

git_comments:

1. !/usr/bin/env bash
 2. add any rocks required for ci_tests to this list lua-curl depends on a libcurl development package (i.e. libcurl4-openssl-dev)
 3. !/usr/bin/env bash
 4. !/usr/bin/env bash
 5. fetch and unpack lua src
 6. civetweb base dir
 7. -L because it's a 302 redirect
 8. this script installs a lua / luarocks environment in .travis/lua this is necessary because travis docker architecture (the fast way) does not permit sudo, and does not contain a useful lua installation
 9. build luarocks
 10. After this script is finished, you can configure your environment to use it by sourcing lua_env.sh
 11. The current versions when this script was written
 12. !/usr/bin/env /bash
 13. fetch and unpack luarocks
 14. add to path required by luarocks installer
 15. directory where this script is located
 16. mv built lua install to target Lua dir
 17. build lua
 18. **comment:** cleanup source dirs
label: code-design
 19. Allocate new message. Caller must hold the lock.
 20. If client is too far behind, return all messages.
 21. We have a message to store. Write-lock the ringbuffer, grab the next message and copy data into it.
 22. A handler for the /ajax/send_message endpoint.
 23. Initialize random number generator. It will be used later on for the session identifier creation.
 24. vim:ts=2:sw=2:
 25. Protects messages, sessions, last_message_id
 26. Get session object for the connection. Caller must hold the lock.
 27. This file is part of the Civetweb project, <http://code.google.com/p/civetweb> It implements an online chat server. For more details, see the documentation on the project web site. To test the application, 1. type "make" in the directory where this file lives 2. point your browser to <http://127.0.0.1:8081>
 28. Redirect user to the login form. In the cookie, store the original URL we came from, so that after the authorization we could redirect back.
 29. In production environment we should ask an authentication system to authenticate the user. Here however we do trivial check that user and password are not empty
 30. Authentication failure, redirect to login.
 31. No suitable handler found, mark as not processed. Civetweb will try to serve the request.
 32. A handler for the /authorize endpoint. Login page form sends user name and password to this endpoint.
 33. Generate session ID. buf must be 33 bytes in size. Note that it is easy to steal session cookies by sniffing traffic. This is why all communication must be SSL-ed.
 34. buf is allocated on stack and hopefully is large enough to hold all messages (it may be too small if the ringbuffer is full and all messages are large. in this case asserts will trigger).
 35. **comment:** TODO(lsm): JSON-encode all text strings
label: code-design
 36. Setup and start Civetweb
 37. Allocate new session object
 38. If "callback" param is present in query string, this is JSONP call. Return 1 in this case, or 0 if "callback" is not specified. Wrap an output in Javascript function call.
 39. Fetch user name and password.
 40. Describes single message sent to a chat. If user is empty (0 length), the message is then originated from the server itself.
 41. Session ID Set user, needed by Javascript code Delete original_url
 42. Return 1 if request is authorized, 0 otherwise.
 43. Get a get of messages with IDs greater than last_id and transform them into a JSON string. Return that string to the caller. The string is dynamically allocated, caller must free it. If there are no messages, NULL is returned.
 44. Message ID User that have sent the message Message text Message timestamp, UTC
 45. Large enough to hold all messages
 46. Authentication success: 1. create new session 2. set session ID token in the cookie 3. remove original_url from the cookie - not needed anymore 4. redirect client back to the original URL The most secure way is to stay HTTPS all the time. However, just to show the technique, we redirect to HTTP after the successful authentication. The danger of doing this is that session cookie can be stolen and an attacker may impersonate the user. Secure application must use HTTPS all the time.
 47. Ringbuffer for messages Current sessions
 48. Empty user indicates server message
 49. Session ID, must be unique Random data used for extra user validation Authenticated user Expiration timestamp, UTC
 50. Always authorize accesses to login page and to authorize URI
 51. Read-lock the ringbuffer. Loop over all messages, making a JSON string.
 52. A handler for the /ajax/get_messages endpoint. Return a list of messages with ID greater than requested.
 53. Wait until enter is pressed, then exit
 54. Return 1 if username/password is allowed, 0 otherwise.
 55. Describes web session.
 56. This file is part of the Civetweb project, <http://sourceforge.net/projects/civetweb/>
 57. <http://sourceforge.net/projects/civetweb/>-->
 58. "},lastModified:{},etag:{},ajax:function(a){function b(){e.success&& e.success.call(k,o,i,x);e.global&&f("ajaxSuccess",[x,e])}function d(){e.complete&&e.complete.call(k,x,i);e.global&&f("ajaxComplete",[x,e]);e.global&&!--c.active&&c.event.trigger("ajaxStop")}function f(q,p){(e.context?c(e.context):c.event).trigger(q,p)}var e=c.extend(true,{},{c.ajaxSettings,a},j,i,o,k=a&&a.context||e,n=e.type.toUpperCase();if(e.data&&e.processData&&typeof e.data=="string")e.data=c.param(e.data,e.traditional);if(e.dataType==="jsonp"){if(n==="GET")N.test(e.url)||((e.url+=(ka.test(e.url)? "&":"?")+(e.jsonp||"callback"))+"=?");else if(!e.data||!N.test(e.data))e.data=(e.data?e.data+"&"":)+(e.jsonp||"callback"))+"=?";e.dataType="json"}if(e.dataType==="json"&&(e.data&&N.test(e.data))||!N.test(e.url))){j=e.jsonpCallback||"jsonp"+sb++;if(e.data)e.data=(e.data+"").replace(N,"="+j+"\$1");e.url=e.url.replace(N,"="+j+"\$1");e.dataType="script";A[j]=A[j]||function(q){o=q;b();d();A[j]=w;try{delete A[j]}catch(p){}z&z.removeChild(C)};if(e.dataType==="script"&&e.cache==null)e.cache=false;if(e.cache== false&&n==="GET"){var r=J(),u=e.url.replace(wb,"\$1_="+r+"\$2");e.url=u+(u==e.url?(ka.test(e.url)? "&":"?")+" "+r:"");if(e.data&&n==="GET")e.url+=ka.test(e.url)? "&":"?");e.data;e.global&&!c.active++&&c.event.trigger("ajaxStart");r=(r=xb.exec(e.url))&&(r[1]&&r[1]==location.protocol||r[2]==location.host);if(e.dataType==="script"&&n==="GET"&&r){var z=s.getElementsByTagName("head")[0]]|s.documentElement,C=s.createElement("script");C.src=e.url;if(e.scriptCharset)C.charset=e.scriptCharset;if(!j){var B=false;C.onload=C.onreadystatechange=function(){if(!B&&(!this.readyState||this.readyState==="loaded")||this.readyState==="complete")){B=true;b();d();C.onload=null;z&&C.parentNode&&z.removeChild(C)}}z.insertBefore(C,z.firstChild);return w}var E=false,x=e.xhr();if(x){e.username?x.open(n,e.url,e.async,e.username,e.password):x.open(n,e.url,e.async);try{if(e.data||a&&a.contentType)x.setRequestHeader("Content-Type",e.contentType);if(e.ifModified){c.lastModified[e.url]&&x.setRequestHeader("If-Modified-Since",c.lastModified[e.url]);c.etag[e.url]&&x.setRequestHeader("If-None-Match",c.etag[e.url])}r||x.setRequestHeader("X-Requested-With","XMLHttpRequest");x.setRequestHeader("Accept",e.dataType&&e.accepts[e.dataType]?e.accepts[e.dataType]+",")}}}}});

59. ! * jQuery JavaScript Library v1.4.2 * http://jquery.com/ * Copyright 2010, John Resig * Dual licensed under the MIT or GPL Version 2 licenses. * http://jquery.org/license * * Includes Sizzle.js * http://sizzlejs.com/ * Copyright 2010, The Dojo Foundation * Released under the MIT, BSD, and GPL Licenses. * * Date: Sat Feb 13 22:33:48 2010 -0500

60. This file is part of the Civetweb project, http://sourceforge.net/projects/civetweb/

61. sourceforge.net/projects/civetweb/ -->

62. Set correct action for the login form. We assume that the SSL port is the next one to insecure one.

63. Note that this page is self-sufficient, it does not load any other CSS or Javascript file. This is done so because only this page is allowed for non-authorized users. If we want to load other files from the frontend, we need to change backend code to allow those for non-authorized users. See chat.c :: must_authorize() function.

64. Keep only chat.maxVisibleMessages, delete older ones.

65. This file is part of Civetweb project, http://sourceforge.net/projects/civetweb/

66. Backend URL, string. 'http://backend.address.com' or " if backend is the same as frontend

67. vim:ts=2:sw=2:et

68. Deselect menu buttons Select clicked button Hide all main windows Show main window

69. input.disabled = true;

70. Stop the server.

71. Always set Content-Length

72. Wait until user hits "enter". Server is running in separate thread. Navigating to http://localhost:8080 will invoke begin_request_handler().

73. Send HTTP reply to the client

74. Start the web server.

75. List of options. Last element must be NULL.

76. Returning non-zero tells civetweb that our function has replied to the client, and civetweb should not send client any more data.

77. Prepare the message we're going to send

78. This function will be called by civetweb on every new request.

79. Prepare callbacks structure. We have only one callback, the rest are NULL.

80. **comment:** Unused
label: code-design

81. User has submitted a form, show submitted data and a variable value

82. Parse form data. input1 and input2 are guaranteed to be NUL-terminated

83. Show HTML form.

84. Mark request as processed

85. Wait until user hits "enter"

86. Send reply to the client, showing submitted form values.

87. Copyright (c) 2014 The Civetweb developers * Copyright (c) 2004-2012 Sergey Lyubka * This file is a part of civetweb project, http://github.com/bel2125/civetweb

88. callback: called after uploading a file is completed

89. Startup options for the server

90. Test server will use this port

91. Display a welcome message

92. Start the server

93. This example is deprecated and no longer maintained. * All relevant parts have been merged into the embedded_c example.

94. ! _WIN32

95. See http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1

96. Mark request as processed

97. Wait until the user hits "enter", then stop the server

98. Main program: Set callbacks and start the server.

99. callback: used to generate all content

100. enctype="multipart/form-data"

101. Show HTML form.

102. close websock

103. chat message

104. init command

105. This example uses deprecated interfaces: global websocket callbacks. They have been superseeded by URI specific callbacks. See examples/embedded_c for an up to date example.

106. **comment:** todo: use mg_start_thread_id instead of mg_start_thread
label: code-design

107. **comment:** todo: wait for the thread instead of a timeout
label: requirement

108. keep alive

109. var connection; var keepAlive = false; function webSockKeepAlive() { if (keepAlive) { connection.send('ping'); // Send the message 'ping' to the server setTimeout("webSockKeepAlive()", 10000); } } function load() { connection = new WebSocket("ws://127.0.0.1/MyWebSock"); connection.onopen = function () { var send = "init " + Math.round(Math.random()*4294967294+1); console.log('Client: ' + send); connection.send(send); keepAlive = true; webSockKeepAlive(); }; connection.onerror = function (error) { keepAlive = false; connection.close(); console.log("WebSocket error: " + error); alert("WebSocket error"); }; connection.onmessage = function (e) { console.log('Server: ' + e.data); if (e.data.substring(0,5) == "title") {window.document.title = e.data.substring(6);} else if (e.data.substring(0,3) == "msg") { var msgStr = document.getElementById('msg'); msgStr.innerHTML = msgStr.innerHTML + e.data.substring(4); } }; } //

110. document_root: The path to the test function websock.htm

111. if the port is changed here, it needs to be changed in websock.htm as well

112. This example uses deprecated interfaces: global websocket callbacks. They have been superseeded by URI specific callbacks. See examples/embedded_c for an up to date example.

113. port: use http standard to match websocket url in websock.htm: ws://127.0.0.1/MyWebSock

114. return 0 to accept every connection

115. WEBSOCKET SERVER

116. Should get the acknowledge message

117. strlen(websocket_acknowledge_msg)

118. Connect client 3

119. * Copyright (c) 2014-2016 the Civetweb developers * Copyright (c) 2014 Jordan Shelley * https://github.com/jshelley * License http://opensource.org/licenses/mit-license.php MIT License

120. First set up a websocket server

121. Won't get any message

122. Simple example program on how to use websocket client embedded C interface.

123. Send websocket welcome message

124. Client 3 should get the websocket welcome message

125. New interface:

126. Obsolete:

127. Send websocket acknowledge message

128. Can not send a websocket goodbye message here - the connection is already * closed

129. return 1 to keep the connection open

130. strlen(websocket_goodbye_msg)

131. Should get the websocket welcome message

132. ****
133. This example is superseeded by other examples, and no longer * actively maintained. * See examples/embedded_c for an up to date example.
134. Now connect a second client
135. Should get the goodbye message
136. WEBSOCKET CLIENT
137. Then connect a first client
138. strlen(websocket_welcome_msg)
139. Client 2 should get the websocket welcome message
140. check for websocket support for Internet Explorer < 10 there are options for websocket support that could be integrated into production code, but for now, we are expecting browser support to be present for this demo
141. websocket connection
142. user connect/disconnect
143. **comment:** Version 0.1 Contributed by William Greathouse 9-Sep-2013 Simple demo of WebSocket connection use. Not a great example of web coding, but it is functional. The meter displays are adapted from CSS-TRICKS Progress Bars by Chris Coyier at <http://css-tricks.com/css3-progress-bars/>
label: code-design
144. Can be anything
145. user turn updates on/off
146. css-tricks.com/css3-progress-bars/
147. time to close the connection
148. simple structure for keeping track of websocket connection
149. second meter 0 to 500, by 10 every 0.5 second
150. Wait until user hits "enter"
151. echo back
152. get simple greeting for the web server
153. we should not send additional messages when close requested/acknowledged
154. websocket_ready_handler() Once websocket negotiation is complete, start a server for the connection
155. websocket_close_handler() When websocket is closed, tell the associated server to shut down
156. Arguments: flags: first byte of websocket frame, see websocket RFC, <http://tools.ietf.org/html/rfc6455>, section 5.2 data, data_len: payload data. Mask, if any, is already applied.
157. end of list
158. third meter 0 to 100, by 10 every 1.0 second
159. up to 16 independent client connections
160. turn on updates
161. client sent PING, respond with PONG
162. first meter 0 to 1000, by 5 every 0.1 second
163. keep connection open
164. ** interpret data as commands here **
165. 0.1 seconds
166. fprintf(stderr, "close handler\n"); /* called for every close, not just websockets */
167. Send initial meter updates
168. time base and structure periodic updates to client for demo
169. received PONG to our PING, no action
170. show the greeting and some basic information
171. if we are closing, server should not send new data
172. Send meter updates
173. **comment:** reset connection information to allow reuse by new client
label: code-design
174. ws_server_thread() Simple demo server thread. Sends periodic updates to connected clients
175. Copyright (c) 2004-2012 Sergey Lyubka This file is a part of civetweb project, <http://github.com/bel2125/civetweb> v 0.1 Contributed by William Greathouse 9-Sep-2013
176. 0.1 second
177. Send periodic PING to assure websocket remains connected, except if we are closing
178. While the connection is open, send periodic updates
179. websocket_connect_handler() On new client connection, find next available server connection and store new connection information. If no more server connections are available tell civetweb to not accept the client request.
180. If client initiated close, respond with close message in acknowledgement
181. turn off updates
182. Add handler for all files with .foo extention
183. Add handler for /close extention
184. Call the form handler
185. Handler may access the request info using mg_get_request_info
186. Add handler for /A* and special handler for /A/B
187. Add HTTP site to open a websocket connection
188. Add handler for /form (serve a file outside the document root)
189. We know the content length in advance
190. * Copyright (c) 2013-2017 the CivetWeb developers * Copyright (c) 2013 No Face Press, LLC * License <http://opensource.org/licenses/mit-license.php> MIT License
191. example from <https://github.com/civetweb/civetweb/issues/347>
192. mg_printf(conn, "<script type='text/javascript'><![CDATA[\n]; ... * xhtml style
193. Check return value:
194. Stop the server
195. We must read until we find the end (chunked encoding * or connection close), indicated my mg_read returning 0
196. Add HTTP site with auth
197. **comment:** MAX_WS_CLIENTS defines how many clients can connect to a websocket at the * same time. The value 5 is very small and used here only for demonstration; * it can be easily tested to connect more than MAX_WS_CLIENTS clients. * A real server should use a much higher number, or better use a dynamic list * of currently connected websocket clients.
label: code-design
198. Set a random password (4 digit number - bad idea from a security * point of view, but this is an API demo, not a security tutorial), * and store it in some directory within the document root (extremely * bad idea, but this is still not a security tutorial). * The reason we create a new password every time the server starts * is just for demonstration - we don't want the browser to store the * password, so when we repeat the test we start with a new password.
199. No valid authorization
200. **comment:** mg_printf(conn, "]]></script>\n"); ... xhtml style
label: code-design
201. Visual Studio 2010+ support llu
202. Add handler for /postresponse example
203. Simple example program on how to use CivetWeb embedded into a C program.
204. List all listening ports
205. Start CivetWeb web server

206. Add application specific SSL initialization
207. Add handler EXAMPLE_URI, to explain the example
208. Add handler for form data
209. IPv4
210. Wait until the server should be closed
211. Send error
212. Add handler for /B, /B/A, /B/B but not for /B*
213. WS site for the websocket connection
214. In this handler, we ignore the req_info and send the file "fileName".
215. Check if libcivetweb has been built with all required features.
216. Add a file upload handler for parsing files on the fly
217. It would be possible to check the request info here before calling * mg_handle_form_request.
218. IPv6
219. Add handler for /cookie example
220. Just tell the user the new password generated for this test.
221. Windows will not work with path > 260 (MAX_PATH), unless we use * the unicode API. However, this is just an example code: A real * code will probably never store anything to D:\\somewhere and * must be adapted to the specific needs anyhow.
222. Handler may access the request info using mg_get_request_info
223. Simple example program on how to use Embedded C++ interface.
224. This handler will handle "everything else", including * requests to files. If this handler is installed, * NO_FILES should be set.
225. Copyright (c) 2013-2017 the Civetweb developers * Copyright (c) 2013 No Face Press, LLC * License http://opensource.org/licenses/mit-license.php MIT License
226. mg_printf(conn, "<script type=\"text/javascript\"><![CDATA[\n]; ... * xhttp style
227. **comment:** mg_printf(conn, "])></script>\n"); ... xhttp style
 label: code-design
228. CivetServer server(options); // <-- C style start <-- C++ style start
229. **comment:** The string is too long and probably truncated. Make sure an * UTF-8 string is never truncated between the UTF-8 code bytes. * This example code must be adapted to the specific needs.
 label: code-design
230. * * getParam(const char *, size_t, const char *, std::string &, size_t) * * Returns a query parameter contained in the supplied buffer. The * occurrence value is a zero-based index of a particular key name. This * should not be confused with the index over all of the keys. * * @param data the - query string (text) * @param data_len - length of the query string * @param name - the key to search for * @param dst - the destination string * @param occurrence - the occurrence of the selected name in the query (0 *based). * @return true if key was found
231. * * removeWebSocketHandler(const std::string &) * * Removes a web socket handler. * * @param uri - the exact URL used in addWebSocketHandler().
232. * * Stores the user provided close handler
233. * * Basic interface for a URI request handler. Handlers implementations * must be reentrant.
234. * * Callback method for GET request. * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise
235. * * Callback method for authorization requests. It is up to the this handler * to generate 401 responses if authorization fails. * * @param server - the calling server * @param conn - the connection information * @returns true if authorization succeeded, false otherwise
236. * * urlEncode(const char *, size_t, std::string &, bool) * * @param src - buffer to be encoded (0 terminated) * @param dst - destination string * @param append - true if string should not be cleared before encoding.
237. * * urlEncode(const char *, size_t, std::string &, bool) * * @param src - buffer to be encoded * @param src_len - length of buffer to be decoded * @param dst - destination string * @param append - true if string should not be cleared before encoding.
238. * * authHandler(struct mg_connection *, void *cbdata) * * Handles the authorization requests. * * @param conn - the connection information * @param cbdata - pointer to the CivetAuthHandler instance. * @returns 1 if authorized, 0 otherwise
239. * * closeHandler(struct mg_connection *) * * Handles closing a request (internal handler) * * @param conn - the connection information
240. * * urlDecode(const std::string &, std::string &, bool) * * @param src - string to be decoded * @param dst - destination string * @param is_form_url_encoded - true if form url encoded * form-url-encoded data differs from URI encoding in a way that it * uses '+' as character for space, see RFC 1866 section 8.2.1 * http://ftp.ics.uci.edu/pub/ietf/html/rfc1866.txt
241. * * getContext() * * @return the context or 0 if not running.
242. forward declaration
243. * * getListeningPorts() * * Returns a list of ports that are listening * * @return A vector of ports
244. * * getParam(struct mg_connection *conn, const char *, std::string &, size_t) * * Returns a query parameter contained in the supplied buffer. The * occurrence value is a zero-based index of a particular key name. This * should not be confused with the index over all of the keys. Note that *this * function assumes that parameters are sent as text in http query string * format, which is the default for web forms. This function will work for * html forms with method="GET" and method="POST" attributes. In other *cases, * you may use a getParam version that directly takes the data instead of *the * connection as a first argument. * * @param conn - parameters are read from the data sent through this *connection * @param name - the key to search for * @param dst - the destination string * @param occurrence - the occurrence of the selected name in the query (0 *based). * @return true if key was found
245. * * Callback method for when websocket handshake is successfully completed, *and connection is ready for data exchange. * * @param server - the calling server * @param conn - the connection information
246. * * urlDecode(const char *, std::string &, bool) * * @param src - buffer to be decoded (0 terminated) * @param dst - destination string * @param is_form_url_encoded - true if form url encoded * form-url-encoded data differs from URI encoding in a way that it * uses '+' as character for space, see RFC 1866 section 8.2.1 * http://ftp.ics.uci.edu/pub/ietf/html/rfc1866.txt
247. * * urlEncode(const std::string &, std::string &, bool) * * @param src - buffer to be encoded * @param dst - destination string * @param append - true if string should not be cleared before encoding.
248. * * getParam(const std::string &, const char *, std::string &, size_t) * * Returns a query parameter contained in the supplied buffer. The * occurrence value is a zero-based index of a particular key name. This * should not be confused with the index over all of the keys. * * @param data - the query string (text) * @param name - the key to search for * @param dst - the destination string * @param occurrence - the occurrence of the selected name in the query (0 *based). * @return true if key was found
249. * * CivetServer * * Basic class for embedded web server. This has an URL mapping built-in.
250. * * urlDecode(const char *, size_t, std::string &, bool) * * @param src - buffer to be decoded * @param src_len - length of buffer to be decoded * @param dst - destination string * @param is_form_url_encoded - true if form url encoded * form-url-encoded data differs from URI encoding in a way that it * uses '+' as character for space, see RFC 1866 section 8.2.1 * http://ftp.ics.uci.edu/pub/ietf/html/rfc1866.txt
251. * * Callback method for HEAD request. * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise
252. * * Callback method for POST request. * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise
253. _CIVETWEB_SERVER_H
254. * * Callback method for when a data frame has been received from the client. * * @param server - the calling server * @param conn - the connection information * @bits: first byte of the websocket frame, see websocket RFC at *http://tools.ietf.org/html/rfc6455, section 5.2 * @data, data_len: payload, with mask (if any) already applied. * @returns true to keep socket open, false to close it
255. * * Callback method for when the connection is closed. * * @param server - the calling server * @param conn - the connection information
256. * * removeAuthHandler(const std::string &) * * Removes an authorization handler. * * @param uri - the exact URL used in addAuthHandler().
257. * * getHeader(struct mg_connection *conn, const std::string &headerName) * @param conn - the connection information * @param headerName - header name to get the value from * @returns a char array which contains the header value as string
258. Copyright (c) 2013-2017 the Civetweb developers * Copyright (c) 2013 No Face Press, LLC * * License http://opensource.org/licenses/mit-license.php MIT License

259. `** addHandler(const std::string &, CivetHandler *)` * Adds a URI handler. If there is existing URI handler, it will * be replaced with this one. * * URI's are ordered and prefix (REST) URI's are supported. * * @param uri - URI to match. * @param handler - handler instance to use.

260. generic user context which can be set/read, the server does nothing with this apart from keep it.

261. `** Callback method for PATCH request.` * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise

262. `** addAuthHandler(const std::string &, CivetAuthHandler *)` * Adds a URI authorization handler. If there is existing URI authorization * handler, it will be replaced with this one. * * URI's are ordered and prefix (REST) URI's are supported. * * @param uri - URI to match. * @param handler - authorization handler instance to use.

263. `** Basic interface for a websocket handler.` Handlers implementations * must be reentrant.

264. `** Callback method for PUT request.` * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise

265. `** Callback method for DELETE request.` * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise

266. `** getCookie(struct mg_connection *conn, const std::string &cookieName, *std::string &cookieValue)` * Puts the cookie value string that matches the cookie name in the *cookieValue destination string. * * @param conn - the connection information * @param cookieName - cookie name to get the value from * @param cookieValue - cookie value is returned using this reference * @returns the size of the cookie value string read.

267. `** Exception class for thrown exceptions within the CivetHandler object.`

268. `** removeHandler(const std::string &)` * Removes a handler. * * @param uri - the exact URL used in addHandler().

269. `** Callback method for when the client intends to establish a websocket *connection, before websocket handshake.` * * @param server - the calling server * * @param conn - the connection information * @returns true to keep socket open, false to close it

270. `** requestHandler(struct mg_connection *, void *cbdata)` * Handles the incoming request. * * @param conn - the connection information * @param cbdata - pointer to the CivetHandler instance. * @returns 0 if implemented, false otherwise

271. `** Destructor`

272. `** Basic interface for a URI authorization handler.` Handler implementations * must be reentrant.

273. `** close()` * * Stops server and frees resources.

274. `** Constructor` * This automatically starts the sever. * It is good practice to call getContext() after this in case there * were errors starting the server. * * Note: CivetServer should not be used as a static instance in a Windows * DLL, since the constructor creates threads and the destructor joins * them again (creating/joining threads should not be done in static * constructors). * * @param options - the web server options. * @param callbacks - optional web server callback methods. * * @throws CivetException

275. `__cplusplus`

276. `** Callback method for OPTIONS request.` * * @param server - the calling server * @param conn - the connection information * @returns true if implemented, false otherwise

277. `** addWebSocketHandler` * Adds a WebSocket handler for a specific URI. If there is existing URI *handler, it will * be replaced with this one. * * URI's are ordered and prefix (REST) URI's are supported. * * @param uri - URI to match. * @param handler - handler instance to use.

278. `** CivetCallbacks` * * wrapper for mg_callbacks

279. This structure may be extended in future versions.

280. Called when civetweb is about to log a message. If callback returns non-zero, civetweb does not log anything.

281. port number

282. Blocks until unique access is obtained to this connection. Intended for use with websockets only. Invoke this before mg_write or mg_printf when communicating with a websocket if your code has server-initiated communication as well as communication in direct response to a message.

283. Send contents of the entire file together with HTTP headers. Parameters: conn: Current connection information. path: Full path to the file to send. mime_type: Content-Type for file. NULL will cause the type to be looked up by the file extension. additional_headers: Additional custom header fields appended to the header. Each header should start with an X-, to ensure it is not included twice. NULL does not append anything.

284. URL-decoded URI (relative). Can be NULL * if the request_uri does not address a * resource at the server host.

285. Called when civetweb initializes SSL library. Parameters: user_data: parameter user_data passed when starting the server. Return value: 0: civetweb will set up the SSL certificate. 1: civetweb assumes the callback already set up the certificate. -1: initializing ssl fails.

286. Skip this field (neither get nor store it). Continue with the * next field.

287. Get the value of particular HTTP header. This is a helper function. It traverses request_info->http_headers array, and if the header is present in the array, returns its value. If it is not present, NULL is returned.

288. redirect all requests: 0 = no, 1 = yes

289. Called when websocket request is received, before websocket handshake. Return value: 0: civetweb proceeds with websocket handshake. 1: connection is closed immediately. This callback is deprecated: Use mg_set_websocket_handler instead.

290. Return information associated with the request. * Use this function to implement a server and get data about a request * from a HTTP/HTTPS client. * Note: Before CivetWeb 1.10, this function could be used to read * a response from a server, when implementing a client, although the * values were never returned in appropriate mg_request_info elements. * It is strongly advised to use mg_get_response_info for clients.

291. Fetch value of certain cookie variable into the destination buffer. Destination buffer is guaranteed to be '\0' - terminated. In case of failure, dst[0] == '\0'. Note that RFC allows many occurrences of the same parameter. This function returns only first occurrence. Return: On success, value length. On error: -1 (either "Cookie:" header is not present at all or the requested parameter is not found). -2 (destination buffer is NULL, zero length or too small to hold the value).

292. User data pointer passed to mg_start()

293. Macros for enabling compiler-specific checks for printf-like arguments.

294. Read data from the remote end, return number of bytes read. Return: 0 connection has been closed by peer. No more data could be read. < 0 read error. No more data could be read from the connection. > 0 number of bytes read into the buffer.

295. URL-decode input buffer into destination buffer. 0-terminate the destination buffer. form-url-encoded data differs from URI encoding in a way that it uses '+' as character for space, see RFC 1866 section 8.2.1 http://ftp.ics.uci.edu/pub/ietf/html/rfc1866.txt Return: length of the decoded data, or -1 if dst buffer is too small.

296. Client's port

297. Stop parsing this request. Skip the remaining fields.

298. Stop the web server. Must be called last, when an application wants to stop the web server and release all associated resources. This function blocks until all Civetweb threads are stopped. Context pointer becomes invalid.

299. utility methods to compare two buffers, case insensitive.

300. Old nomenclature.

301. Store body data into a file.

302. Return values definition for the "field_found" callback in * mg_form_data_handler.

303. Note: The "file in memory" feature is a deletion candidate, since * it complicates the code, and does not add any value compared to * "mg_add_request_handler". * See this discussion thread: * <https://groups.google.com/forum/#!topic/civetweb/h9HT4CmeYqI> * If you disagree, if there is any situation this is indeed useful * and cannot trivially be replaced by another existing feature, * please contribute to this discussion during the next 3 month * (till end of April 2017), otherwise this feature might be dropped * in future releases.

304. Get the value of particular configuration parameter. The value returned is read-only. Civetweb does not allow changing configuration at run time. If given parameter name is not valid, NULL is returned. For valid names, return value is guaranteed to be non-NUL. If parameter is not set, zero-length string is returned.

305. Old nomenclature

306. Called when initializing a new connection object. * Can be used to initialize the connection specific user data * (mg_request_info->conn_data, mg_get_user_connection_data). * When the callback is called, it is not yet known if a * valid HTTP(S) request will be made. * Parameters: * conn: not yet fully initialized connection object * conn_data: output parameter, set to initialize the * connection specific user data * Return value: * must be 0 * Otherwise, the result is undefined

307. Called when civetweb tries to open a file. Used to intercept file open calls, and serve file data from memory instead. Parameters: path: Full path to the file to open. data_len: Placeholder for the file size, if file is served from memory. Return value: NULL: do not serve file from memory, proceed with normal file open. non-NUL: pointer to the file contents in memory. data_len must be initialized with the size of the memory block.

308. Check which features were set when the civetweb library has been compiled. The function explicitly addresses compile time defines used when building the library - it does not mean, the feature has been initialized using a mg_init_library call. mg_check_feature can be called anytime, even before mg_init_library has

been called. Parameters: feature: specifies which feature should be checked The value is a bit mask. The individual bits are defined as: 1 serve files (NO_FILES not set) 2 support HTTPS (NO_SSL not set) 4 support CGI (NO_CGI not set) 8 support IPv6 (USE_IPV6 set) 16 support WebSocket (USE_WEBSOCKET set) 32 support Lua scripts and Lua server pages (USE LUA is set) 64 support server side JavaScript (USE_DUKTAPE is set) 128 support caching (NO_CACHING not set) 256 support server statistics (USE_SERVER_STATS is set) The result is undefined, if bits are set that do not represent a defined feature (currently: feature >= 512). The result is undefined, if no bit is set (feature == 0). Return: If feature is available, the corresponding bit is set If feature is not available, the bit is 0

309. Get a value of particular form variable. Parameters: data: pointer to form-uri-encoded buffer. This could be either POST data, or request_info.query_string. data_len: length of the encoded data. var_name: variable name to decode from the buffer dst: destination buffer for the decoded variable dst_len: length of the destination buffer Return: On success, length of the decoded variable. On error: -1 (variable not found). -2 (destination buffer is NULL, zero length or too small to hold the decoded variable). Destination buffer is guaranteed to be '\0' - terminated if it is not NULL or zero length.

310. Return builtin mime type for the given file name. For unrecognized extensions, "text/plain" is returned.

311. mg_set_websocket_handler Set or remove handler functions for websocket connections. This function works similar to mg_set_request_handler - see there.

312. **comment:** Called when websocket handshake is successfully completed, and connection is ready for data exchange. This callback is deprecated: Use mg_set_websocket_handler instead.

label: code-design

313. **comment:** Send data to the client using printf() semantics. Works exactly like mg_write(), but allows to do message formatting.

label: code-design

314. Handle for the individual connection

315. This structure contains callback functions for handling form fields. It is used as an argument to mg_handle_form_request.

316. Called when civetweb is about to log access. If callback returns non-zero, civetweb does not log anything.

317. Connect to a websocket as a client Parameters: host: host to connect to, i.e. "echo.websocket.org" or "192.168.1.1" or "localhost" port: server port use_ssl: make a secure connection to server error_buffer, error_buffer_size: buffer for an error message path: server path you are trying to connect to, i.e. if connection to localhost/app, path should be "/app" origin: value of the Origin HTTP header data_func: callback that should be used when data is received from the server user_data: user supplied argument Return: On success, valid mg_connection object. On error, NULL. See error_buffer for details.

318. struct mg_websocket_subprotocols * * List of accepted subprotocols

319. Called when civetweb is about to serve Lua server page, if Lua support is enabled. Parameters: lua_context: "lua_State **" pointer.

320. https port: 0 = no, 1 = yes

321. **comment:** TODO: add more data

label: requirement

322. Return array of struct mg_option, representing all valid configuration options of civetweb.c. The array is terminated by a NULL name option.

323. Maximum number of headers

324. Store the field value into a file.

325. Download data from the remote web server. host: host name to connect to, e.g. "foo.com", or "10.12.40.1". port: port number, e.g. 80. use_ssl: whether to use SSL connection. error_buffer, error_buffer_size: error message placeholder. request_fmt,...: HTTP request. Return: On success, valid pointer to the new connection, suitable for mg_read(). On error, NULL. error_buffer contains error message. Example: char ebuf[100]; struct mg_connection *conn; conn = mg_download("google.com", 80, 0, ebuf, sizeof(ebuf), "%s", "GET / HTTP/1.0\r\nHost: google.com\r\n\r\n");

326. Called when data frame has been received from the client. Parameters: bits: first byte of the websocket frame, see websocket RFC at http://tools.ietf.org/html/rfc6455, section 5.2 data, data_len: payload, with mask (if any) already applied. Return value: 1: keep this websocket connection open. 0: close this websocket connection. This callback is deprecated: Use mg_set_websocket_handler instead.

327. Number of HTTP headers

328. Get user data passed to mg_start from context.

329. **comment:** If the "field_found" callback returned FORM_FIELD_STORAGE_GET, * this callback will receive the field data. * * Parameters: * key: Name of the field ("name" property of the HTML input field). * value: Value of the input field. * user_data: Value of the member user_data of mg_form_data_handler * * Return value: * TODO: Needs to be defined.

label: code-design

330. This structure needs to be passed to mg_start(), to let civetweb know which callbacks to invoke. For a detailed description, see <https://github.com/civetweb/civetweb/blob/master/docs/UserManual.md>

331. Deprecated: Use mg_get_valid_options instead.

332. Send data to a websocket server wrapped in a masked websocket frame. Uses mg_lock_connection to ensure that the transmission is not interrupted, i.e., when the application is proactively communicating and responding to a request simultaneously. Send data to a websocket server wrapped in a masked websocket frame. This function is available when civetweb is compiled with -DUSE_WEBSOCKET Return: 0 when the connection has been closed -1 on error >0 number of bytes written on success

333. CIVETWEB_HEADER_INCLUDED

334. Called when civetweb has received new HTTP request. If the callback returns one, it must process the request by sending valid HTTP headers and a body. Civetweb will not do any further processing. Otherwise it must return zero. Note that since V1.7 the "begin_request" function is called before an authorization check. If an authorization check is required, use a request_handler instead. Return value: 0: civetweb will process the request itself. In this case, the callback must not send any data to the client. 1-999: callback already processed the request. Civetweb will not send any data after the callback returned. The return code is stored as a HTTP status code for the access log.

335. Get a formatted link corresponding to the current request Parameters: conn: current connection information. buf: string buffer (out) buflen: length of the string buffer Returns: <0: error >=0: ok

336. Called when civetweb has uploaded a file to a temporary directory as a result of mg_upload() call. Note that mg_upload is deprecated. Use mg_handle_form_request instead. Parameters: file_name: full path name to the uploaded file.

337. Send contents of the entire file together with HTTP headers. * Parameters: * conn: Current connection handle. * path: Full path to the file to send. * mime_type: Content-Type for file. NULL will cause the type to be * looked up by the file extension.

338. HTTP header value

339. Called when civetweb is closing a connection. The per-context mutex is locked when this is invoked. Websockets: Before mg_set_websocket_handler has been added, it was primarily useful for noting when a websocket is closing, and used to remove it from any application-maintained list of clients. Using this callback for websocket connections is deprecated: Use mg_set_websocket_handler instead. Connection specific data: If memory has been allocated for the connection specific user data (mg_request_info->conn_data, mg_get_user_connection_data), this is the last chance to free it.

340. 1 if SSL-ed, 0 if not

341. Allocate maximum headers

342. websocket subprotocol, * accepted during handshake

343. Deprecated: use local_uri instead

344. Convenience function -- create detached thread. Return: 0 on success, non-0 on error.

345. MG_LEGACY_INTERFACE

346. Callback types for websocket handlers in C/C++. mg_websocket_connect_handler Is called when the client intends to establish a websocket connection, before websocket handshake. Return value: 0: civetweb proceeds with websocket handshake. 1: connection is closed immediately. mg_websocket_ready_handler Is called when websocket handshake is successfully completed, and connection is ready for data exchange. mg_websocket_data_handler Is called when a data frame has been received from the client. Parameters: bits: first byte of the websocket frame, see websocket RFC at http://tools.ietf.org/html/rfc6455, section 5.2 data, data_len: payload, with mask (if any) already applied. Return value: 1: keep this websocket connection open. 0: close this websocket connection. mg_connection_close_handler Is called, when the connection is closed.

347. mg_request_handler Called when a new request comes in. This callback is URI based and configured with mg_set_request_handler(). Parameters: conn: current connection information. cbdata: the callback data configured with mg_set_request_handler(). Returns: 0: the handler could not handle the request, so fall through. 1 - 999: the handler processed the request. The return code is stored as a HTTP status code for the access log.

348. **comment:** Print error message to the opened error log stream. This utilizes the provided logging configuration. conn: connection (not used for sending data, but to get parameters) fmt: format string without the line return ...: variable argument list Example: mg_cry(conn, "i like %s", "logging");

label: code-design

349. Get context information. Useful for server diagnosis. Parameters: ctx: Context handle buffer: Store context information here. buflen: Length of buffer (including a byte required for a terminating 0). Return: Available size of system information, excluding a terminating 0. The information is complete, if the return value is

smaller than buflen. The result is a JSON formatted string, the exact content may vary. Note: It is possible to determine the required buflen, by first calling this function with buffer = NULL and buflen = NULL. The required buflen is one byte more than the returned value. However, since the available context information changes, you should allocate a few bytes more.

- 350. Get the field value.
- 351. Check if the current request has a valid authentication token set. * A file is used to provide a list of valid user names, realms and * password hashes. The file can be created and modified using the * mg_modify_passwords_file API function. * Parameters: * conn: Current connection handle. * realm: Authentication realm. If NULL is supplied, the sever domain * set in the authentication_domain configuration is used. * filename: Path and name of a file storing multiple password hashes. * Return: * > 0 Valid authentication * 0 Invalid authentication * < 0 Error (all values < 0 should be considered as invalid * authentication, future error codes will have negative * numbers) * -1 Parameter error * -2 File not found
- 352. Send contents of the entire file together with HTTP headers.
- 353. URL part after '?', not including '?', or NULL
- 354. Start web server. Parameters: callbacks: mg_callbacks structure with user-defined callbacks. options: NULL terminated list of option_name, option_value pairs that specify Civetweb configuration parameters. Side-effects: on UNIX, ignores SIGCHLD and SIGPIPE signals. If custom processing is required for these, signal handlers must be set up after calling mg_start(). Example: const char *options[] = { "document_root", "/var/www", "listening_ports", "80,443s", NULL }; struct mg_context *ctx = mg_start(&my_func, NULL, options); Refer to <https://github.com/civetweb/civetweb/blob/master/docs/UserManual.md> for the list of valid option and their possible values. Return: web server context, or NULL on error.
- 355. mg_set_request_handler Sets or removes a URI mapping for a request handler. This function uses mg_lock_context internally. URI's are ordered and prefixed URI's are supported. For example, consider two URIs: /a/b and /a /a matches /a /a/b matches /a/b /a/c matches /a Parameters: ctx: server context uri: the URI (exact or pattern) for the handler handler: the callback handler to use when the URI is requested. If NULL, an already registered handler for this URI will be removed. The URI used to remove a handler must match exactly the one used to register it (not only a pattern match). cbdata: the callback data to give to the handler when it is called.
- 356. Get user data set for the current connection.
- 357. Send data to the client. Return: 0 when the connection has been closed -1 on error >0 number of bytes written on success
- 358. Client's IP address. Deprecated: use remote_addr instead
- 359. Send a part of the message body, if chunked transfer encoding is set. * Only use this function after sending a complete HTTP request or response * header with "Transfer-Encoding: chunked" set.
- 360. E.g. "1.0", "1.1"
- 361. "GET", "POST", etc
- 362. Close the connection opened by mg_download().
- 363. HTTP header name
- 364. Send HTTP error reply.
- 365. User supplied argument, passed to all callback functions.
- 366. Client certificate information
- 367. **comment:** Initialize this library. This should be called once before any other * function from this library. This function is not guaranteed to be * thread safe. * Parameters: * features: bit mask for features to be initialized. * Return value: * initialized features * 0: error
label: requirement
- 368. Client certificate information (part of mg_request_info)
- 369. Process form data. * Returns the number of fields handled, or < 0 in case of an error. * Note: It is possible that several fields are already handled successfully * (e.g., stored into files), before the request handling is stopped with an * error. In this case a number < 0 is returned as well. * In any case, it is the duty of the caller to remove files once they are * no longer required.
- 370. New nomenclature
- 371. Return array of strings that represent valid configuration options. For each option, option name and default value is returned, i.e. the number of entries in the array equals to number_of_options x 2. Array is NULL terminated.
- 372. Get context from connection.
- 373. Return information associated with a HTTP/HTTPS response. * Use this function in a client, to check the response from * the server.
- 374. Connect to a TCP server as a client (can be used to connect to a HTTP server) Parameters: host: host to connect to, i.e. "www.wikipedia.org" or "192.168.1.1" or "localhost" port: server port use_ssl: make a secure connection to server error_buffer, error_buffer_size: buffer for an error message Return: On success, valid mg_connection object. On error, NULL. See error_buffer for details.
- 375. Opcodes, from <http://tools.ietf.org/html/rfc6455>
- 376. URL-decoded URI (absolute or relative, * as in the request)
- 377. mg_set_auth_handler Sets or removes a URI mapping for an authorization handler. This function works similar to mg_set_request_handler - see there.
- 378. URL-encode input buffer into destination buffer. returns the length of the resulting buffer or -1 is the buffer is too small.
- 379. Return CivetWeb version.
- 380. Authenticated user, or NULL if no auth used
- 381. This callback function is called, if a new field has been found. * The return value of this callback is used to define how the field * should be processed. * * Parameters: * key: Name of the field ("name" property of the HTML input field). * filename: Name of a file to upload, at the client computer. * Only set for input fields of type "file", otherwise NULL. * path: Output parameter: File name (incl. path) to store the file * at the server computer. Only used if FORM_FIELD_STORAGE_STORE * is returned by this callback. Existing files will be * overwritten. * pathlen: Length of the buffer for path. * user_data: Value of the member user_data of mg_form_data_handler * * Return value: * The callback must return the intended storage for this field * (See FORM_FIELD_STORAGE_*)
- 382. Called when a new worker thread is initialized. Parameters: ctx: context handle thread_type: 0 indicates the master thread 1 indicates a worker thread handling client connections 2 indicates an internal helper thread (timer thread)
- 383. New nomenclature.
- 384. E.g. "OK"
- 385. __cplusplus
- 386. **comment:** MD5 hash given strings. Buffer 'buf' must be 33 bytes long. Varargs is a NULL terminated list of ASCIIz strings. When function returns, buf will contain human-readable MD5 hash. Example: char buf[33]; mg_md5(buf, "aa", "bb", NULL);
label: code-design
- 387. Send data to a websocket client wrapped in a websocket frame. Uses mg_lock_connection to ensure that the transmission is not interrupted, i.e., when the application is proactively communicating and responding to a request simultaneously. Send data to a websocket client wrapped in a websocket frame. This function is available when civetweb is compiled with -DUSE_WEBSOCKET Return: 0 when the connection has been closed -1 on error >0 number of bytes written on success
- 388. Called when civetweb is about to send HTTP error to the client. Implementing this callback allows to create custom error pages. Parameters: status: HTTP error status code. Return value: 1: run civetweb error handler. 0: callback already handled the error.
- 389. Copyright (c) 2013-2017 The Civetweb Developers * Copyright (c) 2004-2013 Sergey Lyubka * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
- 390. Get information on the system. Useful for support requests. Parameters: buffer: Store system information as string here. buflen: Length of buffer (including a byte required for a terminating 0). Return: Available size of system information, excluding a terminating 0. The information is complete, if the return value is smaller than buflen. The result is a JSON formatted string, the exact content may vary. Note: It is possible to determine the required buflen, by first calling this function with buffer = NULL and buflen = NULL. The required buflen is one byte more than the returned value.

391. **comment:** If the "field_found" callback returned FORM_FIELD_STORAGE_STORE, * the data will be stored into a file. If the file has been written * successfully, this callback will be called. This callback will * not be called for only partially uploaded files. The * mg_handle_form_request function will either store the file completely * and call this callback, or it will remove any partial content and * not call this callback function. * * Parameters: * path: Path of the file stored at the server. * file_size: Size of the stored file in bytes. * user_data: Value of the member user_data of mg_form_data_handler * * Return value: * TODO: Needs to be defined.
label: code-design
392. Handle for the HTTP service itself
393. Read entire request body and store it in a file "path". Return: < 0 Error >= 0 Number of bytes stored in file "path".
394. Get a value of particular form variable. Parameters: data: pointer to form-uri-encoded buffer. This could be either POST data, or request_info.query_string. data_len: length of the encoded data. var_name: variable name to decode from the buffer dst: destination buffer for the decoded variable dst_len: length of the destination buffer occurrence: which occurrence of the variable, 0 is the first, 1 the second... this makes it possible to parse a query like b=x&a=y&a=z which will have occurrence values b:0, a:0 and a:1 Return: On success, length of the decoded variable. On error: -1 (variable not found). -2 (destination buffer is NULL, zero length or too small to hold the decoded variable). Destination buffer is guaranteed to be '\0' - terminated if it is not NULL or zero length.
395. Length (in bytes) of the request body, can be -1 if no length was given.
396. mg_authorization_handler Callback function definition for mg_set_auth_handler Parameters: conn: current connection information. cbdata: the callback data configured with mg_set_request_handler(). Returns: 0: access denied 1: access granted
397. Called when civetweb context is deleted. Parameters: ctx: context handle
398. Deprecated: Use mg_get_server_ports instead.
399. Lock server context. This lock may be used to protect resources that are shared between different connection/worker threads.
400. This structure contains information about the HTTP request.
401. Connection-specific user data
402. E.g. 200
403. Wait for a response from the server Parameters: conn: connection ebuf, ebuf_len: error message placeholder. timeout: time to wait for a response in milliseconds (if < 0 then wait forever) Return: On success, >= 0 On error/timeout, < 0
404. 1 = IPv4, 2 = IPv6, 3 = both
405. Called when civetweb has finished processing request.
406. Called after civetweb context has been created, before requests are processed. Parameters: ctx: context handle
407. File upload functionality. Each uploaded file gets saved into a temporary file and MG_UPLOAD event is sent. Return number of uploaded files. Deprecated: Use mg_handle_form_request instead.
408. **comment:** Get connection information. Useful for server diagnosis. Parameters: ctx: Context handle idx: Connection index buffer: Store context information here. buflen: Length of buffer (including a byte required for a terminating 0). Return: Available size of system information, excluding a terminating 0. The information is complete, if the return value is smaller than buflen. The result is a JSON formatted string, the exact content may vary. Note: It is possible to determine the required buflen, by first calling this function with buffer = NULL and buflen = NULL. The required buflen is one byte more than the returned value. However, since the available context information changes, you should allocate a few bytes more.
label: code-design
409. Send HTTP digest access authentication request. * Browsers will send a user name and password in their next request, showing * an authentication dialog if the password is not stored. * Parameters: * conn: Current connection handle. * realm: Authentication realm. If NULL is supplied, the sever domain * set in the authentication_domain configuration is used. * Return: * < 0 Error
410. Client's IP address as a string.
411. Set user data for the current connection.
412. Get text representation of HTTP status code.
413. Add, edit or delete the entry in the passwords file. * * This function allows an application to manipulate .htpasswd files on the * fly by adding, deleting and changing user records. This is one of the * several ways of implementing authentication on the server side. For another, * cookie-based way please refer to the examples/chat in the source tree. * * Parameter: * passwords_file_name: Path and name of a file storing multiple passwords * realm: HTTP authentication realm (authentication domain) name * user: User name * password: * If password is not NULL, entry modified or added. * If password is NULL, entry is deleted. * * Return: * 1 on success, 0 on error.
414. Get the list of ports that civetweb is listening on. The parameter size is the size of the ports array in elements. The caller is responsibility to allocate the required memory. This function returns the number of struct mg_server_ports elements filled in, or <0 in case of an error.
415. Un-initialize this library. * Return value: * 0: error
416. !/bin/sh using "pass" for every password
417. remove last build
418. check if the build was successful
419. ! /bin/sh
420. new scan build
421. pack build results for upload
422. check if we use the correct directory
423. copy files to build folder
424. return "ok"
425. data is "var1=val1&var2=val2...". Find variable first
426. Point s to the end of the value
427. No handler found, close connection
428. Add one extra character: in case the post-data is a text, it is required as 0-termination. Do not increment con_len, since the 0 terminating is not part of the content (text or binary).
429. **comment:** malloc may fail for huge requests
label: code-design
430. Happens when a request hits the server before the context is saved
431. No handler found
432. Maximum cookie length as per microsoft is 4096. <http://msdn.microsoft.com/en-us/library/ms178194.aspx>
433. Decode variable into destination buffer
434. get requests do store html <form> field values in the http query_string
435. Point p to variable value
436. Copyright (c) 2013-2017 the Civetweb developers * Copyright (c) 2013 No Face Press, LLC * * License <http://opensource.org/licenses/mit-license.php> MIT License
437. Now update the variable index
438. **comment:** TODO: print error
label: code-design
439. 3. call a "handle everything" callback, if registered
440. Asterisk
441. Get a random number (independent of C rand function)
442. **comment:** Function should never be called
label: code-design
443. range buffer is big enough
444. **comment:** TODO (low): If some data has been sent, a correct error * reply can no longer be sent, so just close the connection
label: code-design
445. Step 1.2: Check websocket protocol version.
446. TODO (mid): Check the mg_send_http_error situations in this function
447. Calculate an estimate for the required space
448. assert(handler == NULL);

449. "Cache-Control: private\r\n" (= default)
450. File name is relative to the webserver working directory * or it is absolute system path
451. Timeout: should retry
452. RFC2616 Section 10.1.2
453. REPORT method only allowed for CGI/Lua/LSP and callbacks.
454. Handler for error group, e.g., 5xx error * handler * for all server errors (500-599)
455. RFC 6585, Section 4
456. Set close flag, so the server thread can exit.
457. Initialize port and len as invalid.
458. File already exists
459. Do not show current dir (but show hidden files as they will * also be removed)
460. Cannot get host from the Host: header. * Fallback to our IP address.
461. <http://en.wikipedia.org/wiki/WebM>
462. File is read only
463. Here we can either generate and send a directory listing, * or send an "access denied" error.
464. RFC2616 Section 10.2.3
465. Pointer to the beginning of the portion of the incoming websocket * message queue. * The original websocket upgrade request is never removed, so the queue * begins after it.
466. Loop over multiple requests sent using the same connection * (while "keep alive").
467. assert(s >= p);
468. Traverse index files list. For each entry, append it to the given * path and see if the file exists. If it exists, break the loop
469. Gobble initial spaces
470. Redirect error code from -1 to -2 (destination buffer too * small).
471. SSL descriptor
472. We are using non-blocking sockets. Thus, the * set_sock_timeout(so.sock, timeout); * call is no longer required.
473. Depending on the SDK, this function uses either * (volatile unsigned int *) or (volatile LONG *), * so whatever you use, the other SDK is likely to raise a warning.
474. This value is the last one
475. Bind to a specific IPv4 address, e.g. 192.168.1.5:8080
476. We need an additional wait loop around this, because in some cases * with TLSwe may get data from the socket but not from SSL_read. * In this case we need to repeat at least once.
477. queue is full
478. 123
479. !NO_CGI
480. AUTH_HANDLER
481. 0 = nowhere, 1 = on disk, 2 = in memory
482. Important: on new connection, reset the receiving buffer. Credit * goes to crule42.
483. success
484. RFC2616 Section 10.4.16
485. ignore leading whitespaces
486. Use "Cache-Control: max-age" instead of "Expires" header. * Reason: see https://www.mnot.net/blog/2007/05/15/expires_max-age
487. Error
488. 100 nsecs
489. Request is directed to another server: The server name is * longer * than * the request name. Drop this case here to avoid overflows * in * the * following checks.
490. Data block
491. If a file should not be cached, do not only send * max-age=0, but also pragmas and Expires headers.
492. 1 is CRYPTO_LOCK
493. keep legacy behavior
494. difftime for struct timespec. Return value is in seconds.
495. OK
496. **comment:** TODO: 1) uri is deprecated; * 2) here, ri.uri is the http response code
 label: code-design
497. Compiler information
498. If CGI file is a script, try to read the interpreter line
499. Is path itself a directory?
500. Read from the socket into the next available location in the * message queue.
501. process HTTP connection
502. 12. Directory uris should end with a slash
503. return 0 on success, just like fclose
504. And disabling them does not work either: * #pragma clang diagnostic ignored "-Wno-error=date-time" * #pragma clang diagnostic ignored "-Wdate-time" * So we just have to disable ALL warnings for some lines * of code.
505. Server did not recv anything -> just close the connection
506. Start master (listening) thread
507. RFC2616 Section 10.2.1
508. EBUSY
509. out of memory
510. Debug: * printf("Missing function: %s\n", fp->name);
511. is still not MIT) and use dynamic binding like
512. body_len is the length of the entire queue in bytes * len is the length of the current message * data_len is the length of the current message's data payload * header_len is the length of the current message's header
513. Get some operating system independent thread id
514. **comment:** File is bad and will be removed anyway.
 label: code-design
515. Note: mg_snprintf will not cause a buffer overflow above. * So, string truncation checks are not required here.
516. open a connection
517. so->sa.sin6.sin6_family = AF_INET6; already set by mg_inet_pton
518. Add 1 to len for the + character we skipped before
519. RFC2616 Section 10.4.18
520. If the path is not complete, skip processing.
521. Access granted
522. Port is specified with a +, bind to IPv6 and IPv4, INADDR_ANY
523. IANA registered MIME types * (<http://www.iana.org/assignments/media-types>) * application types
524. This is the problematic case for CRYPTO_set_id_callback: * The OS pthread_t can not be cast to unsigned long.
525. Server starts *now*
526. SSL context
527. defined(_WIN32) && !defined(__SYMBIAN32__) - \ WINDOWS / UNIX include block
528. PATCH method (RFC 5789)
529. Free Lua state of lua background task
530. chunksize not set correctly

531. Pointer to name (in user addr space).
532. Overflow case
533. Try to read until it succeeds, fails, times out, or the server * shuts down.
534. padding added after data member
535. data is "var1=val1&var2=val2...". Find variable first
536. Wait until everything has stopped.
537. RFC2616 Section 10.5.4
538. Send a chunk, if "Transfer-Encoding: chunked" is used
539. copy from the remainder of the last received chunk
540. Total bytes sent to client
541. Receive error
542. Transfer-Encoding is chunked: * 0 = not chunked, * 1 = chunked, do data read yet, * 2 = chunked, some data read, * 3 = chunked, all data read
543. Public function to check http digest authentication header
544. processed
545. Parameter error
546. next try for a partial match, we will accept uri/something
547. defined(__GNUC__) || defined(__MINGW32__)
548. 11. Handle put/delete/mkcol requests
549. RFC2616 Section 10.2 - Successful 2xx
550. model
551. 6.2.2. Check if put authorization for static files is * available.
552. may be NULL
553. sscanf() is safe here, since send_ssi_file() also uses buffer * of size MG_BUF_LEN to get the tag. So strlen(tag) is * always < MG_BUF_LEN.
554. Publish the content length back to the response info.
555. Parent
556. File properties filled by mg_fopen:
557. CGI error handler should show the status code
558. Fatal error - abort start. However, this situation should never * occur in practice.
559. Size of the accepted socket queue
560. **comment:** This function is deprecated. Use mg_get_valid_options instead.
 label: code-design
561. IPv6
562. **comment:** This is an "in-memory" file, that can not be replaced
 label: code-design
563. RFC 6585, Section 5
564. finally try for pattern match
565. If file name is fishy, reset the file structure and return * error. * Note it is important to reset, not just return the error, cause * functions like is_file_opened() check the struct.
566. **comment:** Appear "unused parameter" warnings
 label: code-design
567. Put the temp file in place of real file
568. Compile-time option to control stack size, * e.g. -DUSE_STACK_SIZE=16384
569. Return True if we should reply 304 Not Modified.
570. Do not compress ranges.
571. 11.1. PUT method
572. !_WIN32 && !_SYMBIAN32_
573. IPv6 address, examples: see above
574. Add user-specified variables
575. Delete is not successful: Return 500 (Server error).
576. On Windows, the file creation time can be higher than the * modification time, e.g. when a file is copied. * Since the Last-Modified timestamp is used for caching * it should be based on the most recent timestamp.
577. Buffer allocation successful. Store the string there.
578. or timeout occurred * => the code must stay in the while loop
579. lock while cert is reloading
580. thread was not created
581. strcat with additional NULL check to avoid clang scan-build warning.
582. Return buffered data
583. MINGW typedefs pid_t to int. Using #define here.
584. If PEM file is not specified and the init_ssl callback * is not specified, skip SSL initialization.
585. closed
586. Each error code path in this function must send an error
587. Forward declaration for Windows
588. Feature check API function
589. Start a thread storing the thread context.
590. RFC2616 Section 10.3 - Redirection 3xx
591. Reallocate the buffer
592. Always put directories on top
593. REPORT method (RFC 3253)
594. Will only work for RFC 952 compliant hostnames, * starting with a letter, containing only letters, * digits and hyphen ('-'). Newer specs may allow * more, but this is not guaranteed here, since it * may interfere with rules for port option lists.
595. ext now points to the path suffix
596. defined(USE_STACK_SIZE) && (USE_STACK_SIZE > 1)
597. assert(conn->data_len <= conn->buf_size);
598. Deprecated: Use mg_get_server_ports instead.
599. Content length is not specified by the client.
600. RFC2616 Section 10.3.1
601. Two \r\n - standard compliant
602. assert(is_delete_request || connect_handler!=NULL || * ready_handler!=NULL || data_handler!=NULL || * close_handler!=NULL);
603. End of civetweb.c
604. We sent an "upgrade" request. For a correct websocket * protocol handshake, we expect a "101 Continue" response. * Otherwise it is a protocol violation. Maybe the HTTP * Server does not know websockets.
605. 14. directories
606. Don't wait if the server is going to be stopped.
607. Values from errno.h in Windows SDK (Visual Studio).
608. Decode variable into destination buffer
609. !EWOULDBLOCK
610. The second word is the URI
611. 1.2. do a https redirect, if required. Do not decode URIs yet.

612. File exists (precondition for calling this function), * but can not be opened by the server.
613. out: file found (directly)
614. mg_stat returns 0 if the file does not exist
615. POST or PUT request without content length set
616. Get Subject and issuer
617. set_ssl_option() function updates this array. * It loads SSL library dynamically and changes NULLs to the actual addresses * of respective functions. The macros above (like SSL_connect()) are really * just calling these functions indirectly via the pointer.
618. No Tag
619. out: websocket connection?
620. Calculate how much to read from the file in the buffer
621. ok = FALSE; already set by init
622. RFC2324 Section 2.3.2
623. file stored on disk
624. If we don't memset stat structure to zero, mtime will have * garbage and strftime() will segfault later on in * print_dir_entry(). memset is required only if mg_stat() * fails. For more details, see * <http://code.google.com/p/mongoose/issues/detail?id=79>
625. Thread local storage index
626. See https://www.chilkatsoft.com/p/p_299.asp
627. For given directory path, substitute it to valid index file. * Return 1 if index file has been found, 0 if not found. * If the file is found, its stats is returned in stp.
628. isgraph or isspace = isprint
629. Read from IO channel - opened file descriptor, socket, or SSL descriptor. * Return value: * >=0 .. number of bytes successfully read * -1 .. timeout * -2 .. error
630. Add data bytes in the current chunk have been read, * so we are expecting \r\n now.
631. 9a. In case the server uses only callbacks, this uri is * unknown. * Then, all request handling ends here.
632. last_modified = now ... assumes the file may change during * runtime, * so every mg_fopen call may return different data
633. HTTP reply status code, e.g. 200
634. Master thread adds accepted socket to a queue
635. shutdown of the socket at client side
636. BEGIN: CLOCK_MONOTONIC = stopwatch (time differences)
637. # is a comment
638. This server does not have any real files, thus the * PUT/DELETE methods are not valid.
639. The headers "Host", "Sec-WebSocket-Key", "Sec-WebSocket-Protocol" and * "Sec-WebSocket-Version" are also required. * Don't check them here, since even an unsupported websocket protocol * request still IS a websocket request (in contrast to a standard HTTP * request). It will fail later in handle_websocket_request.
640. While the delimiter is quoted, look for the next delimiter.
641. Create a temporary file name. Length has been checked before.
642. Size of structures, aligned to 8 bytes
643. Ignore errors for readonly files
644. Process a connection - may handle multiple requests * using the same connection. * Must be called with a valid connection (conn and * conn->ctx must be valid).
645. in: filename (must be valid)
646. PATCH method only allowed for CGI/Lua/LSP and callbacks.
647. Skip over '=' character
648. Avoid CRYPTO_cleanup_all_ex_data(); See discussion: * https://wiki.openssl.org/index.php/Talk:Library_Initialization
649. The first word is the HTTP version
650. Dynamically load SSL library. Set up ctx->ssl_ctx pointer.
651. Mark required libraries
652. Keep reading the input (either opened file descriptor fd, or socket sock, * or SSL descriptor ssl) into buffer buf, until \r\n\r\n appears in the * buffer (which marks the end of HTTP request). Buffer buf may already * have some data. The length of the data is stored in nread. * Upon every read operation, increase nread by the number of bytes read.
653. Send len bytes from the opened file to the client.
654. Message is a valid request
655. !fileno MINGW #defines fileno
656. PROPPATCH, COPY, MOVE, LOCK, UNLOCK (RFC 2518)
657. Set linger option to avoid socket hanging out after close. This * prevent ephemeral port exhaust problem under high QPS.
658. GCC seems to have a flaw with its own socket macros: * <http://www.linuxquestions.org/questions/programming-9/impossible-to-use-gcc-with-wconversion-and-standard-socket-macros-841935/>
659. C++ wants that for INT64_MAX
660. Some of the parameters may be NULL
661. Lua Script
662. not required anyway
663. This is a POST/PUT request, or another request with body data.
664. **comment:** user names may not contain a ':' and may not be empty, * so lines starting with ':' may be used for a special purpose
 label: code-design
665. Copy the data payload into the data pointer for the * callback. Cast to 31 bit is OK, since we * limited data_len
666. !USE_WEBSOCKET
667. Default: don't configure any linger
668. Content-Length header value
669. NO_CACHING
670. Send headers
671. Deallocate context itself
672. Sleep quantum in ms
673. CORS request is asking for additional headers
674. Last time throttled data was sent
675. p must correspond to end_word - 1
676. the ready handler returned false
677. Send 304 "Not Modified" - this must not send any body data
678. last_modified = conn->ctx.start_time; * May be used if the data does not change during runtime. This * allows * browser caching. Since we do not know, we have to assume the file * in memory may change.
679. File exists and is not a directory.
680. Terminate string
681. Print error message to the opened error log stream.
682. **comment:** clock_gettime is not implemented on OSX prior to 10.12
 label: requirement
683. worst case
684. According to RFC 2616, Section 14.21, caching times should not exceed * one year. A year with 365 days corresponds to 31536000 seconds, a * leap * year to 31622400 seconds. For the moment, we just send whatever has * been configured, still the behavior for >1 year should be considered * as undefined.
685. The pre-allocated buffer not large enough.
686. Can it be replaced?
687. HTTP (RFC 2616)
688. Tag is opening
689. draft-tbray-http-legally-restricted-status-05, * Section 3

690. path does not exist and can not be created
691. Propagate the error
692. USE_SERVER_STATS
693. Step 2: If a callback is responsible, call it.
694. **comment:** Alternative alloc_vprintf() for non-compliant C runtimes
 label: code-design
695. **comment:** TODO: Check ssl_verify_peer and ssl_ca_path here. * SSL_CTX_set_verify call is needed to switch off server * certificate checking, which is off by default in OpenSSL and * on in yaSSL.
 label: code-design
696. This is an existing file (not a directory). * Check if write permission is granted.
697. [http://msdn.microsoft.com/en-us/library/ms739165\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms739165(v=vs.85).aspx): * "Note that enabling a nonzero timeout on a nonblocking socket * is not recommended.", so set it to blocking now
698. Use a local copy here, since substitute_index_file will * change the content of the file status
699. 6. authorization check
700. 94
701. Close, if keep alive is not enabled
702. Unknown success code
703. RFC2616 Section 10.5.5
704. Use 64-bit file offsets by default
705. Return OpenSSL error message (from CRYPTO lib)
706. done with OpenSSL
707. **comment:** Protect against directory disclosure attack by removing '..', * excessive '/' and '\' characters
 label: code-design
708. sendfile is only available for Linux
709. For the moment, use option c: We look for a proper file, * but since a file request is not always a script resource, * the authorization check might be different.
710. Request buffers are not pre-allocated. They are private to the * request and do not contain any state information that might be * of interest to anyone observing a server status.
711. This function is called from send_directory() and used for * sorting directory entries by size, or name, or modification time. * On windows, __cdecl specification is needed in case if project is built * with __stdcall convention. qsort always requires __cdels callback.
712. Frame format: <http://tools.ietf.org/html/rfc6455#section-5.2>
713. Partial content
714. The websocket_client context has only this thread. If it runs out, set the stop_flag to 2 (= "stopped").
715. 2. if this ip has limited speed, set it for this connection
716. ssscanf and the option splitting code ensure the following condition
717. RFC2616 Section 10.4.1
718. Open file for read only access
719. look for matching subprotocol
720. 34
721. Publish the content length back to the request info.
722. read error
723. 416 = Range Not Satisfiable
724. Ignore too long entries that may overflow path buffer
725. remove double backslash (check i > 0 to preserve UNC paths, * like \\server\file.txt)
726. DNS is case insensitive, so use case insensitive string compare here
727. The RFC standard version (<https://tools.ietf.org/html/rfc6455>) * requires a Sec-WebSocket-Key header.
728. GetVersion was declared deprecated
729. End of header key (*dp == ':')
730. Parsed Authorization header
731. Linux et al already use unicode. No need to convert.
732. Comma found. Store length and shift the list ptr
733. hpush
734. step 1. completed, the url is known now
735. **comment:** When using -Weverything, clang does not accept its own headers * in a release build configuration. Disable what is too much in * -Weverything.
 label: code-design
736. RFC2518 Section 10.2, RFC4918 Section 11.1
737. Set bits for available features according to API documentation. * This bit mask is created at compile time, according to the active * preprocessor defines. It is a single const value at runtime.
738. RFC says that all initial whitespaces should be ingored
739. Add this thread to cv's waiting list
740. must be a end of line
741. Memory information
742. pollres = 0 means timeout
743. Copy VARIABLE=VALUE\0 string into the free space
744. video
745. CGI scripts may support all HTTP methods
746. All calls to construct_etag use 64 byte buffer
747. POSIX dirent interface
748. Prepare the environment block
749. Size of the request + headers in a buffer
750. Initialize elements.
751. **comment:** TODO (mid): error handling depending on the error code. * These codes are different between Windows and Linux. * Currently there is no problem with failing send calls, * if there is a reproducible situation, it should be * investigated in detail.
 label: code-design
752. Used by mg_(un)lock_connection to ensure * atomic transmissions for websockets
753. see errno
754. File name is relative to the webserver root
755. Timeout occurred, but no data available.
756. If there is anything beyond end_word, copy it.
757. Websocket Lua script
758. NO_SOCKETLEN_T
759. Note: set_sock_timeout is not required for non-blocking sockets. * Leave this function here (commented out) for reference until * CivetWeb 1.9 is tested, and the tests confirm this function is * no longer required.
760. timeout
761. Valid URIs according to RFC 3986 * (<https://www.ietf.org/rfc/rfc3986.txt>) * must only contain reserved characters :?#[@!\$&()'*=; and unreserved characters A-Z a-z 0-9 and -_~ * and % encoded symbols.
762. RFC2616 Section 10.2.5
763. IE uses commas, FF uses * spaces
764. if an error occurred, close the connection

765. Stringify binary data. Output buffer must be twice as big as input, * because each byte takes 2 bytes in string representation
766. Point p to variable value
767. This is the hixie version
768. error
769. set an error, if not yet set
770. HTTP 1.1 default is keep alive
771. Convert in 32 bit words, if data is 4 byte aligned
772. RFC2518 Section 10.1
773. 1 if gzip encoding is accepted
774. Check if the script file is in a path, allowed for script files. * This can be used if uploading files is possible not only for the server * admin, and the upload mechanism does not check the file extension.
775. update existing handler
776. nanoseconds
777. Return HTTP header value, or NULL if not found.
778. Then send length information, chunk and terminating \r\n.
779. Allocate new buffer
780. Initialize thread local storage
781. 1.3. decode url (if config says so)
782. Child
783. Skip over leading LWS
784. timeout: return 0
785. **comment:** TODO: 1) uri is deprecated; * 2) here, ri.uri is the http response code
label: code-design
786. Now read CGI reply into a buffer. We need to set correct * status code, thus we need to see all HTTP headers first. * Do not send anything back to client, until we buffer in all * HTTP headers.
787. Parsing failure.
788. RFC2616 Section 10.4.15
789. See table: <http://wwwcplusplus.com/reference/cctype/>
790. See IANA HTTP status code assignment: * <http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml>
791. Ok
792. seconds
793. Start a thread to read the websocket client connection * This thread will automatically stop when mg_disconnect is * called on the client connection
794. Describes listening socket, or socket which was accept()ed by the master * thread and queued for future handling by the worker thread.
795. BEGIN: unknown clock
796. Should get data, but got EOL
797. socket error - check errno
798. !defined(NO_SSL)
799. RFC2616 Section 10.4.2
800. Protocol violation
801. Allocate a new buffer.
802. Interface function. Parameters are provided by the user, so do * at least some basic checks.
803. _WIN32_WCE
804. Flush current buffer
805. Check config, if index scripts may have sub-resources
806. defined(USE_WEBSOCKET)
807. Step 2.1 check and select subprotocol
808. Step 8: Check if the file exists at the server
809. Is port supposed to redirect everything to SSL * port
810. Step 9: Check for zipped files:
811. Need to retry the function call "later". * See https://linux.die.net/man/3/ssl_get_error * This is typical for non-blocking sockets.
812. **comment:** TODO: for 1.10: restructure how files in memory are handled
label: code-design
813. Return 0 means, path itself is a directory.
814. RFC2616 Section 10.4.5
815. 1. get the request url
816. Not chunked: content length is known
817. This is an SSL specific error
818. Send a response to CORS preflights only if * access_control_allow_methods is not NULL and not an empty string. * In this case, scripts can still handle CORS.
819. RFC2518 Section 10.3, RFC4918 * Section 11.2
820. Environment buffer
821. call init_thread for a worker thread (type 1)
822. Message is a valid response
823. **comment:** __attribute__((unused))
label: code-design
824. **comment:** request not yet handled by a handler or redirect, so the request * is processed here
label: requirement
825. Step 4: Check if gzip encoded response is allowed
826. Flush if buffer is full
827. Copy socket to the queue and increment head
828. could use atomic compare exchange, but this * seems overkill for statistics data
829. Throttling, bytes/sec. <= 0 means no * throttle
830. Stop signal received: somebody called mg_stop. Quit.
831. Client
832. **comment:** TODO: set some error message
label: requirement
833. **comment:** NOTE: If SO_EXCLUSIVEADDRUSE is used, * Windows might need a few seconds before * the same port can be used again in the * same process, so a short Sleep may be * required between mg_stop and mg_start.
label: code-design
834. O_BINARY
835. HCP24: some changes to compare hole var_name
836. This version uses 8 byte body data in a GET request
837. ALTERNATIVE_QUEUE
838. could use SSL_CTX_set_default_passwd_cb_userdata
839. Since we know all sockets, we can shutdown the connections.
840. hope nobody uses this
841. Requests information
842. Mark memory as dead
843. Get system information. It can be printed or stored by the caller. * Return the size of available information.

844. Delete is successful: Return 204 without content.
845. File should be created
846. Allow only if access_control_allow_headers is * not NULL and not an empty string. If this * configuration is set to *, allow everything. * Otherwise this configuration must be a list * of allowed HTTP header names.
847. accepted subprotocols for ws/wss requests.
848. Shut down signal
849. SSL is not supported
850. First satisfy needs of the server
851. <http://msdn.microsoft.com/en-us/library/1w06ktidy.aspx>
852. EINTR can be generated on a socket with a timeout set even * when SA_RESTART is effective for all relevant signals * (see signal(7)). * => stay in the while loop
853. **comment:** Shutdown according to * https://wiki.openssl.org/index.php/Library_Initialization#Cleanup * <http://stackoverflow.com/questions/29845527/how-to-properly-uninitialize-openssl>
label: code-design
854. Darwin prior to 7.0 and Win32 do not have socklen_t
855. Using the given file format, user name and domain must not contain * ':'
856. Flush buffer
857. Perform case-insensitive match of string against pattern
858. Make modifiable copy of the auth header
859. Only accept a full file path, not a Windows short (8.3) path.
860. timeout is handled by the while loop
861. mask flag and opcode
862. Implement the deprecated mg_upload function by calling the new * mg_handle_form_request function. While mg_upload could only handle * HTML forms sent as POST request in multipart/form-data format * containing only file input elements, mg_handle_form_request can * handle all form input elements and all standard request methods.
863. NOTE(lsm): order is important here. should_keep_alive() call * is using parsed request, which will be invalid after * memmove's below. * Therefore, memorize should_keep_alive() result now for later * use in loop exit condition.
864. Run SSL_shutdown twice to ensure completely close SSL connection
865. Step 1.3: Could check for "Host", but we do not really need this * value for anything, so just ignore it.
866. processing
867. A websocket protocol has the following HTTP headers: * * Connection: Upgrade * Upgrade: Websocket
868. Make sure the port is valid and vector ends with 's', 'r' or '
869. Null-terminate the destination
870. assert(connect_handler==NULL && ready_handler==NULL && * data_handler==NULL && close_handler==NULL);
871. Deallocate request handlers
872. Request too long
873. Unknown server error code
874. Set to 1 if the content is gzipped, in which * case we need a "Content-Encoding: gzip" header
875. character is ok
876. destructor not supported for Windows
877. Length of the message being read at the front of the * queue. Cast to 31 bit is OK, since we limited * data_len before.
878. remove existing handler
879. **comment:** Set some extra bits not defined in the API documentation. * These bits may change without further notice.
label: documentation
880. Get a unique thread ID as unsigned long, independent from the data type * of thread IDs defined by the operating system API. * If two calls to mg_current_thread_id return the same value, they calls * are done from the same thread. If they return different values, they are * done from different threads. (Provided this function is used in the same * process context and threads are not repeatedly created and deleted, but * CivetWeb does not do that). * This function must match the signature required for SSL id callbacks: * CRYPTO_set_id_callback
881. **comment:** TODO: check if it is still used in response_info
label: code-design
882. **comment:** TODO: handle non-local request (PROXY)
label: code-design
883. Convert the nonce from the client to a number.
884. Linear congruential generator
885. ASN1_digest is deprecated. Do the calculation manually, * using EVP_Digest.
886. Open failed
887. if clock_gettime is declared, then __CLOCK_AVAILABILITY will be defined
888. Regard empty password as no password - remove user record.
889. data points to the place where the message is stored when passed to * the * websocket_data callback. This is either mem on the stack, or a * dynamically allocated buffer if it is too large.
890. This function only checks if the uri is valid, not if it is * addressing the current server. So civetweb can also be used * as a proxy server.
891. Check if the request is directed to a different server.
892. URI: /path_to_script/ ... using index.cgi
893. This file can not be sent using sendfile. * This might be the case for pseudo-files in the * /sys/ and /proc/ file system. * Use the regular user mode copy code instead.
894. Send chunk of data that may have been read after the headers
895. The body data is completed when the connection * is closed.
896. Allocate a mutex for this connection to allow communication both * within the request handler and from elsewhere in the application
897. Ignore SIGPIPE signal, so if browser cancels the request, it * won't kill the whole process.
898. Do not delete anything shorter
899. Check for quotechar
900. **comment:** Avoid warnings for Xcode 7. It seems it does no longer exist in Xcode 8
label: code-design
901. Step 5: The websocket connection has been accepted
902. BEGIN: CLOCK_PROCESS = CPU usage of process
903. Assume the data may change during runtime by setting * last_modified = now
904. END: unknown clock
905. **comment:** TODO: get timeout def
label: code-design
906. The converted number corresponds to the time the nounce has been * created. This should not be earlier than the server start.
907. Parse HTTP request, fill in mg_request_info structure. * This function modifies the buffer by NUL-terminating * HTTP request components, header names and header values. * Parameters: * buf (in/out): pointer to the HTTP header to parse and split * len (in): length of HTTP header buffer * re (out): parsed header as mg_request_info * buf and ri must be valid pointers (not NULL), len>0. * Returns <0 on error.
908. !NO_SSL
909. Check for a valid HTTP version key
910. Step 10: Script resources may handle sub-resources
911. Find end of HTTP header
912. Return stringified MD5 hash for list of strings. Buffer must be 33 bytes.
913. standard case if called from close_socket_gracefully

914. Look at the "path" extension and figure what mime type it has. * Store mime type in the vector.
915. Step 1: Set all initially unknown outputs to zero
916. RFC7238 Section 3
917. Name/Pattern of the URI.
918. **comment:** TODO: specific error message
 label: code-design
919. Request is directed to another server: It could be a * substring * like notmyserver.com
920. Next would be the HTTP version
921. Get SSL client certificate information (if set)
922. If any ACL is set, deny by default
923. For every system, "(sizeof(int) == sizeof(void *))" is either always * true or always false. One of the two branches is unreachable in any case. * Unfortunately the C standard does not define a way to check this at * compile time, since the #if preprocessor conditions can not use the sizeof * operator as an argument.
924. USE_LUA
925. **comment:** TODO: check for openSSL 1.1
 label: requirement
926. For sockets, wait for the socket using select
927. Script was in an illegal path
928. Make up and send the status line
929. Check end of word
930. RFC 2774, Section 7
931. Check if the nonce is too high, so it has not (yet) been used by the * server.
932. Reset buffer, so we can always use strcat.
933. Connection structure has been pre-allocated
934. Convert to Unicode and back. If doubly-converted string does not * match the original, something is fishy, reject.
935. Wrap pointers if needed
936. SSL context for client connections
937. deallocate system name string
938. Fatal error - abort start. However, this situation should * never occur in practice.
939. **comment:** Enable unused function warning again
 label: code-design
940. Store the ssl library handle.
941. RFC2616 Section 10.4.13
942. Get context information. It can be printed or stored by the caller. * Return the size of available information.
943. mg_stat must fill all fields, also for files in memory
944. Put so socket structure into the queue
945. **comment:** FIXME(lsm): fix this.
 label: code-design
946. pthread cond not available
947. SCRIPT_NAME
948. Start the server
949. none found
950. RFC2616 Section 10.1.1
951. NOTE(lsm): due to a bug in MSIE, we do not compare the URI
952. sf_sent<0 means error, thus fall back to the classic way
953. **comment:** not defined as a preprocessor macro, replacing with '0' for '#if/#elif'
 label: code-design
954. Now we know that our FIN is ACK-ed, safe to close
955. End of initial operating system specific define block.
956. If truncated, add "..."
957. Helper function for deprecated mg_upload.
958. Request to create a directory has been fulfilled successfully. * No need to put a file.
959. add mask
960. **comment:** TODO: (void)mg_snprintf(NULL, dst, dst_size, "implement strftime() * for WinCE");
 label: code-design
961. Step 5: If there is no root directory, don't look for files.
962. put_dir returns -1 if the path is too long
963. **comment:** Signal mg_stop() that we're done. * WARNING: This must be the very last thing this * thread does, as ctx becomes invalid after this line.
 label: code-design
964. Both read and write were successful, adjust counters
965. Try to delete it.
966. Compile-time option to control stack size, e.g. * -DUSE_STACK_SIZE=16384
967. Windows SO_REUSEADDR lets many procs binds to a * socket, SO_EXCLUSIVEADDRUSE makes the bind fail * if someone already has the socket -- DTL
968. ORDERPATCH (RFC 3648)
969. 1 if connection must be closed
970. !NO_POPEN
971. linked list of uri handlers
972. Mark handles that should not be inherited. See * <https://msdn.microsoft.com/en-us/library/windows/desktop/ms682499%28v=vs.85%29.aspx>
973. Request/response has content length set
974. **comment:** We should subtract the time used in select from remaining * "milliseconds", in particular if called from mg_poll with a * timeout quantum. *
 Unfortunately, the remaining time is not stored in "tv" in all * implementations, so the result in "tv" must be considered undefined. * See
 <http://man7.org/linux/man-pages/man2/select.2.html>
 label: code-design
975. Could be a hostname
976. asterisk
977. GCC does not realize one branch is unreachable, so it raises some * pointer cast warning within the unreachable branch.
978. The first word has to be the HTTP method
979. Send host, port, uri and (if it exists) ?query_string
980. 8.3: If the request target is a directory, there could be * a substitute file (index.html, index.cgi, ...).
981. Tag to long for buffer
982. This is an IO error. Look at errno.
983. RFC2616 Section 10.4.6
984. Context has been created - init user libraries
985. error callint poll
986. According to <https://en.wikipedia.org/wiki/Symbian#History>, * Symbian is no longer maintained since 2014-01-01. * Recent versions of CivetWeb are no longer tested for Symbian. * It makes no sense, to support an abandoned operating system. * All remaining "#ifdef __SYMBIAN__" cases will be dropped from * the code sooner or later.
987. Note that root == NULL is a regular use case here. This occurs, * if all requests are handled by callbacks, so the WEBSOCKET_ROOT * config is not required.
988. Currently gz files can not be scripts.

989. A file should be created or overwritten.
990. Some ANSI #includes are not available on Windows CE
991. assert(fp != NULL);
992. Update lsa port in case of random free ports
993. If Range: header specified, act accordingly
994. Deallocate config parameters
995. EAGAIN/EWOULDBLOCK: * standard case if called from close_socket_gracefully * => should return -1
996. IPv4
997. Strings returned from bn_bn2hex must be freed using OPENSSL_free, * see https://linux.die.net/man/3/bn_bn2hex
998. call the init_connection callback if assigned
999. Create substitutes for POSIX functions in Win32.
1000. Unified socket address. For IPv6 support, add IPv6 address structure in * the * union u.
1001. Step 4: Check if there is a responsible websocket handler.
1002. For every compiler, either "sizeof(pthread_t) > sizeof(unsigned long)" * or not, so one of the two conditions will be unreachable by construction. * Unfortunately the C standard does not define a way to check this at * compile time, since the #if preprocessor conditions can not use the sizeof * operator as an argument.
1003. Check if the method is known to the server. The list of all known * HTTP methods can be found here at * <http://www.iana.org/assignments/http-methods/http-methods.xhtml>
1004. RFC2616 Section 10.4 - Client Error 4xx
1005. Step 6: Call the ready handler
1006. See CONTEXT_* above
1007. Store the <
1008. Set default value if needed
1009. There should be already an error message
1010. Timeout in s:
1011. **comment:** Print message to buffer. If buffer is large enough to hold the message, * return buffer. If buffer is to small, allocate large enough buffer on * heap, * and return allocated buffer.
label: code-design
1012. Adjust length for trailing LWS
1013. 1.1.1970 in filedate
1014. put_dir returns 0 if path is a directory
1015. Point s to the end of the value
1016. **comment:** This standard handler is only used for real files. * Scripts should support the OPTIONS method themselves, to allow a * maximum flexibility. * Lua and CGI scripts may fully support CORS this way (including * preflights).
label: code-design
1017. Could not parse the CGI response. Check if some error message on * stderr.
1018. Log is written to a file and/or a callback. If both are not set, * executing the rest of the function is pointless.
1019. Buffer size
1020. Yes it does, break the loop
1021. **comment:** va_copy should always be a macro, C99 and C++11 - DTL
label: code-design
1022. Send all current and obsolete cache opt-out directives.
1023. Return 1 on success. Always initializes the ah structure.
1024. Calculate SHA1 fingerprint and store as a hex string
1025. Yes, they are optional
1026. RFC 6585, Section 6
1027. _WIN32
1028. Parse HTTP headers from the given buffer, advance buf pointer * to the point where parsing stopped. * All parameters must be valid pointers (not NULL). * Return <0 on error.
1029. The master thread ID
1030. Threads have different return types on Windows and Unix.
1031. Free client certificate info
1032. Return 1 if request is authorised, 0 otherwise.
1033. ignore leading whitespaces
1034. Open file for writing, create and append
1035. Check if the server may write this file
1036. Scan user-defined mime types first, in case user wants to * override default mime types.
1037. Get serial number
1038. Here stop_flag is 1 - Initiate shutdown.
1039. **comment:** All file operations need to be rewritten to solve #246.
label: code-design
1040. **comment:** Legacy INIT, if mg_start is called without mg_init_library. * Note: This may cause a memory leak
label: code-design
1041. Return NULL or the relative uri at the current server
1042. value of request_timeout is in seconds, config in milliseconds
1043. Lua_State for a websocket connection
1044. Encode 'path' which is assumed UTF-8 string, into UNICODE string. * wbuf and wbuf_len is a target buffer and its length.
1045. The worker thread IDs
1046. Take connection [idx]. This connection is not locked in * any way, so some other thread might use it.
1047. "The masking key is a 32-bit value chosen at random by the client." * <http://tools.ietf.org/html/draft-ietf-hybi-thewebsOCKETprotocol-17#section-5>
1048. Out of memory
1049. Set socket family to IPv6, do not use IPV6_V6ONLY
1050. Server version
1051. Other request
1052. Signaled when socket is produced
1053. Call consume_socket() even when ctx->stop_flag > 0, to let it * signal sq_empty condvar to wake up the master waiting in * produce_socket()
1054. ready
1055. Set close flag, so keep-alive loops will stop
1056. RFC2616 Section 10.1 - Informational 1xx
1057. 6.3. This is either a OPTIONS, GET, HEAD or POST request, * or it is a PUT or DELETE request to a resource that does not * correspond to a file. Check authorization.
1058. Timeout
1059. Set to 1 if mg_stat is called for a directory
1060. No option known to set thread name for MinGW
1061. Do not open file (used in is_file_in_memory)
1062. CGI is not supported
1063. assert(option != NULL); assert(option[0] != '\0');
1064. 11.4. PATCH method * This method is not supported for static resources, * only for scripts (Lua, CGI) and callbacks.
1065. to expose flockfile and friends in stdio.h

1066. Check the user's password, return 1 if OK
1067. This is always the case, if sf_file is not a "normal" file, * e.g., for sending data from the output of a CGI process.
1068. See also https://www.mnot.net/cache_docs/
1069. mg_init_library counter
1070. End of the list
1071. If it is a directory, print directory entries too if Depth is not 0
1072. **comment:** TODO: Remove this unlock as well, when lock is moved.
 label: code-design
1073. This is not a valid field.
1074. The handler did NOT handle the request.
1075. for setgroups()
1076. User supplied argument for the handler function.
1077. Set IPv6 only option, but don't abort on errors.
1078. The second word is the status as a number
1079. This is a valid CORS preflight, and the server is configured * to * handle it automatically.
1080. char **envp
1081. IPv6 hex string is 46 chars
1082. Send user defined error pages, if defined
1083. Cross-origin resource sharing (CORS).
1084. Prepare connection data structure
1085. fail early, don't waste time checking other header * fields
1086. if it's not the end of line, there must be a next word
1087. Protects nonce_count
1088. filep->is_directory = 0; filep->gzipped = 0; .. already done by * memset
1089. Use memcpy for alignment
1090. control characters and spaces are invalid
1091. RFC 2295, Section 8.1
1092. Do not add truncated strings to the environment
1093. This happens, e.g., in calls from parse_auth_header, * if the user name contains a " character.
1094. HTTPS connection
1095. Define the initial recursion depth for processesing htpasswd files that * include other htpasswd * (or even the same) files. It is not difficult to provide a file or files * s.t. they force civetweb * to infinitely recurse and then crash.
1096. ebuf is set by connect_socket, * free all memory and return NULL;
1097. Value is either quote-delimited, or ends at first comma or space.
1098. Reset all outputs
1099. Initialize thread local storage before calling any callback
1100. WINDOWS / UNIX include block
1101. Directory entry
1102. No more data to read
1103. **comment:** WinCE-TODO: define stat, remove, rename, _rmdir, _lseeki64
 label: code-design
1104. file stored in memory
1105. Do not show current dir and hidden files
1106. Size was set by the callback
1107. **comment:** TODO (mid): Add a webdav unit test first, before changing * anything here.
 label: test
1108. Prepare Etag, Date, Last-Modified headers. Must be in UTC, * according to * <http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3>
1109. **comment:** Show no warning in case system functions are not used.
 label: code-design
1110. Visual Studio 6 does not know __func__ or __FUNCTION__ * The rest of MS compilers use __FUNCTION__, not C99 __func__ * Also use _strtoi64 on modern M\$ compilers
1111. RFC2616 Section 10.3.5
1112. The 'path' given to us points to the directory. Remove all trailing * directory separator characters from the end of the path, and * then append single directory separator character.
1113. Close files
1114. Combining two pseudo-random number generators and a high resolution * part * of the current server time will make it hard (impossible?) to guess * the * next number.
1115. RFC2616 Section 10.4.7
1116. If Content-Length is not set for a request with body data * (e.g., a PUT or POST request), we do not know in advance * how much data should be read.
1117. **comment:** Return fake connection structure. Used for logging, if connection * is not applicable at the moment of logging.
 label: code-design
1118. Return -1 in an error case
1119. Set reuse option, but don't abort on errors.
1120. 16-bit length field
1121. Does it exist?
1122. Check if the config_options and the corresponding enum have compatible * sizes.
1123. Finally check if the server corresponds to the authentication * domain of the server (the server domain). * Allow full matches (like <http://mydomain.com/path/file.ext>), and * allow subdomain matches (like <http://www.mydomain.com/path/file.ext>), * but do not allow substrings (like * <http://notmydomain.com/path/file.ext> * or <http://mydomain.com.fake/path/file.ext>).
1124. pthread_t may be any data type, so a simple cast to unsigned long * can rise a warning/error, depending on the platform. * Here memcpy is used as an anything-to-anything cast.
1125. Ignore truncation in access log
1126. This code uses static_assert to check some conditions. * Unfortunately some compilers still do not support it, so we have a * replacement function here.
1127. stdin read
1128. Truncate here and set the key name
1129. All SSI tags start with <!--#
1130. END: CLOCK_REALTIME
1131. Note that C callbacks are no longer called when Lua is * responsible, so C can no longer filter callbacks for Lua.
1132. Legacy before split into local_uri and request_uri
1133. Convert month to the month number. Return -1 on error, or month number
1134. **comment:** TODO (mid): This check does not seem to make any sense !
 label: code-design
1135. Append VARIABLE=VALUE\0 string to the buffer, and add a respective * pointer into the vars array. Assumes env != NULL and fmt != NULL.
1136. Reallocate existing block
1137. key == pthread_getspecific(sTlsKey);
1138. restore the string
1139. Number of requests handled by this connection
1140. Behaves like realloc(), but frees original pointer on failure

1141. Reference counter for crypto library.
1142. 8.2: Check if it is a script type.
1143. Do nothing, callback has served the request. Store * then return value as status code for the log and discard * all data from the client not used by the callback.
1144. The connection struct, pre- * allocated for each worker
1145. Chunked encoding: 3=chunk read completely * completely
1146. _WIN32 && !_SYMBIAN32
1147. For Windows, change all slashes to backslashes in path names.
1148. Depending on the OpenSSL version, the callback may be * 'void (*)(SSL *, int, int)' or 'void (*)(const SSL *, int, int)' * yielding in an "incompatible-pointer-type" warning for the other * version. It seems to be "unclear" what is correct: * <https://bugs.launchpad.net/ubuntu/+source/openssl/+bug/1147526> *
<https://www.openssl.org/docs/man1.0.2/ssl/ssl.html> * <https://www.openssl.org/docs/man1.1.0/ssl/ssl.html> *
<https://github.com/openssl/openssl/blob/1d97c8435171a7af575f73c526d79e1ef0ee5960/ssl/ssl.h#L1173> * Disable this warning here. * Alternative would be a version dependent ssl_info_callback and * a const-cast to call 'char *SSL_get_app_data(SSL *ssl)' there.
1149. defined(_GNUC_)
1150. Protects (max|num)_threads
1151. Discard all buffered data for this request
1152. Set the time the request was received. This value should be used for * timeouts.
1153. **comment:** unused if NO_FILES is defined
 label: code-design
1154. connected with IPv6
1155. assert(conn->request_len < 0 || conn->data_len >= conn->request_len);
1156. Server start time, used for authentication * and for diagnostics.
1157. Tag is closing
1158. stdout write
1159. INO_CACHING
1160. Set linger timeout
1161. Reserved for future use, must be zero.
1162. BEGIN: CLOCK_THREAD = CPU usage of thread
1163. Deprecated function mg_upload - use mg_handle_form_request instead.
1164. assert(ebuf[0] != '0');
1165. **comment:** TODO: Define mg_response_info and implement parsing
 label: code-design
1166. Parse all HTTP headers
1167. Ignore if a file in memory is inside a folder
1168. We got ready to write. Don't check the timeout * but directly go back to write again.
1169. Error, no bytes read
1170. NO_SSL_DL
1171. Random number generator will initialize at the first call
1172. For a given PUT path, create all intermediate subdirectories. * Return 0 if the path itself is a directory. * Return 1 if the path leads to a file. * Return -1 for if the path is too long. * Return -2 if path can not be created.
1173. <http://www.webdav.org/specs/rfc4918.html>, 9.1: * Some PROPFIND results MAY be cached, with care, * as there is no cache validation mechanism for * most properties. This method is both safe and * idempotent (see Section 9.1 of [RFC2616]).
1174. Apply mask if necessary
1175. A helper function for checking if a comma separated list of values * contains * the given option (case insensitively). * 'header' can be NULL, in which case false is returned.
1176. A helper function for traversing a comma separated list of values. * It returns a list pointer shifted to the next value, or NULL if the end * of the list found. * Value is stored in val vector. If value has form "x=y", then eq_val * vector is initialized to point to the "y" part, and val vector length * is adjusted to point only to "x".
1177. Unknown redirection code
1178. join worker thread
1179. Remote socket address
1180. Read frame payload from the first message in the queue into * data and advance the queue by moving the memory in place.
1181. Not initialized yet
1182. C runtime is not standard compliant, vsnprintf() returned -1. * Switch to alternative code path that uses incremental * allocations.
1183. ignore error on read only file
1184. Parse a buffer: * Forward the string pointer till the end of a word, then * terminate it and forward till the begin of the next word.
1185. Windows return values: see * <https://msdn.microsoft.com/en-us/library/windows/desktop/ms738520%28v=vs.85%29.aspx>
1186. This is the end of the header
1187. 5.2. check if the request will be handled by a callback
1188. Handle to algorithm used for fingerprint
1189. text
1190. Calculate Sec-WebSocket-Accept reply from Sec-WebSocket-Key.
1191. Step 1.1: Check Sec-WebSocket-Key.
1192. This is just a quick fix if the client offers multiple * protocols. The handler should have a list of accepted * protocols on his own * and use it to select one protocol among those the client * has * offered.
1193. It could be an absolute uri:
1194. Just a single protocol -> accept it.
1195. Delete was successful: Return 204 without content.
1196. Must be 0x1000.
1197. Calculate number of bytes added to the environment
1198. Parse UTC date-time string, and return the corresponding time_t value.
1199. is_file_in_memory returned true
1200. Step 3.2: Lua is responsible: call it.
1201. Time (since system start) when the request * was received
1202. CGI must be executed in its own directory. 'dir' must point to the * directory containing executable program, 'p' must point to the * executable program name relative to 'dir'.
1203. http://en.wikipedia.org/wiki/Advanced_Audio_Coding
1204. Allocation failed, exit the loop and then close the * connection
1205. Features
1206. Socket already closed by client/peer, close socket without linger
1207. Buffer is big enough
1208. Alternative queue is well tested and should be the new default
1209. ok
1210. Use a const cast here and modify the string. * We are going to restore the string later.
1211. callback already processed the request. Store the return value as a status code for the access log.
1212. RFC2616 Section 10.5.3
1213. The RFC version (<https://tools.ietf.org/html/rfc6455>) is 13.
1214. audio
1215. pthread_setname_np first appeared in glibc in version 2.12
1216. DEBUG

1217. Note that since V1.7 the "begin_request" function is called * before an authorization check. If an authorization check is * required, use a request_handler instead.
1218. RFC2616 Section 10.3.4
1219. However, the reasonable default is to not accept a nonce from a * previous start, so if anyone changed the access rights between * two restarts, a new login is required.
1220. Include the header file here, so the CivetWeb interface is defined for the * entire implementation, including the following forward definitions.
1221. Allocate context and initialize reasonable general case defaults.
1222. Linux uses -1 on error, Windows NULL.
1223. could use atomic compare exchange, but this * seems overkill for statistics data
1224. not recognized on older compiler versions
1225. Signaled when socket is consumed
1226. out: file status structure
1227. Send header
1228. This is a file in memory, but we cannot store the * properties * now. * Called from "is_file_in_memory" function.
1229. Connected client
1230. Wakeup workers that are waiting for connections to handle.
1231. 1 if in handler for user defined error * pages
1232. done
1233. RFC 6585, Section 3
1234. **comment:** TODO: retry condition
 label: code-design
1235. No more data to write
1236. **comment:** Data type of linger structure elements may differ, * so we don't know what cast we need here. * Disable type conversion warnings.
 label: code-design
1237. How many bytes of content have been read
1238. Windows file systems are not case sensitive, but you can still use * uppercase and lowercase letters (on all modern file systems). * The server can check if the URI uses the same upper/lowercase * letters an the file system, effectively making Windows servers * case sensitive (like Linux servers are). It is still not possible * to use two files with the same name in different cases on Windows * (like /a and /A) - this would be possible in Linux. * As a default, Windows is not case sensitive, but the case sensitive * file name check can be activated by an additional configuration.
1239. Return 0 on success, non-zero if an error occurs.
1240. Connection object will be null if something goes wrong
1241. Handler for http/https or authorization requests.
1242. stdin write
1243. We are in a tag, either SSI tag or html tag
1244. **comment:** Keep stderr and stdout in two different pipes. * Stdout will be sent back to the client, * stderr should go into a server error log.
 label: code-design
1245. The previous code must not add any header starting with X- to make * sure no one of the additional_headers is included twice
1246. String truncation in buf may only occur if * error_handler is too long. This string is * from the config, not from a client.
1247. RFC2616 Section 10.4.12
1248. file is in memory
1249. RFC2616 Section 10.2.4
1250. **comment:** TODO (Feature): Add config value for allowed script path. * Default: All allowed.
 label: code-design
1251. **comment:** Here we are at a dead end: * According to URI matching, a callback should be * responsible for handling the request, * we called it, but the callback declared itself * not responsible. * We use a goto here, to get out of this dead end, * and continue with the default handling. * A goto here is simpler and better to understand * than some curious loop.
 label: code-design
1252. assert(connect_handler==NULL && ready_handler==NULL && * data_handler==NULL && close_handler==NULL);
1253. Thread ID (-1=caller thread).
1254. RFC2616 Section 10.3.6
1255. Check if the uri is valid. * return 0 for invalid uri, * return 1 for *, * return 2 for relative uri, * return 3 for absolute uri without port, * return 4 for absolute uri with port
1256. Step 2: Check if the request attempts to modify the file system
1257. 0 = undef, numerical value may change in different * versions. For the current definition, see * mg_get_connection_info_impl
1258. protocol found, port set
1259. Is keep alive allowed by the server
1260. assert(conn->data_len >= 0);
1261. For client connections, mg_context is fake. Since we need to set a * callback function, we need to create a copy and modify it.
1262. If we can't find the actual file, look for the file * with the same name but a .gz extension. If we find it, * use that and set the gzipped flag in the file struct * to indicate that the response need to have the content- * encoding: gzip header. * We can only do this if the browser declares support.
1263. Open the given file and temporary file
1264. If new user, just add it
1265. large enough, so there will be no overflow
1266. ws/wss client context
1267. No body allowed. Close the connection.
1268. Print properties for the requested resource itself
1269. If a callback has been specified, call it.
1270. in/out: request (must be valid)
1271. If the queue is empty, wait. We're idle at this point.
1272. free context
1273. Step 3: Check if it is a websocket request, and modify the document * root if required
1274. User-defined callback function
1275. We have returned all buffered data. Read new data from the remote * socket.
1276. Two newline, no carriage return - not standard compliant, * but * it * should be accepted
1277. DEBUG_TRACE
1278. 125
1279. This error code is unknown. This should not happen.
1280. 1.1. split into url and query string
1281. call the connection_close callback if assigned
1282. Step 8: Call the close handler
1283. no handler to set, this was a remove request to a non-existing * handler
1284. Everything else but a 3 digit code is invalid
1285. TRACE method (RFC 2616) is not supported for security reasons
1286. Linux, ... (not Windows)
1287. NOTE: Maybe some data has already been sent.
1288. NO_SSL
1289. Value has form "x=y", adjust pointers and lengths * so that val points to "x", and eq_val points to "y".
1290. Check explicit wish of the client
1291. port remains 0 if the protocol is not found

1292. USE_DUKTAPE
1293. Perform redirect and auth checks before calling begin_request() * handler. * Otherwise, begin_request() would need to perform auth checks and * redirects.
1294. GetProcAddress() returns pointer to function
1295. Unknown informational status code
1296. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>
1297. Establish the client connection and request upgrade
1298. 8.1: File exists.
1299. Reset attributes. DO NOT TOUCH is_ssl, remote_ip, remote_addr, * remote_port
1300. Do not read more than the total content length of the * request.
1301. **comment:** Disable spurious conversion warning for GCC
 label: code-design
1302. HAVE_POLL
1303. Use of fopen here is OK, since fname is only ASCII
1304. However, Linux does not return 0 on success either.
1305. abort scan as soon as one malformed character is found
1306. Config option name, config types, default value
1307. normal end
1308. URI: /path_to_script/script.cgi
1309. **comment:** appease "unused variable" warning for release builds
 label: code-design
1310. Init variables
1311. Reset user data, after close callback is called. * Do not reuse it. If the user needs a destructor, * it must be done in the connection_close callback.
1312. in: size of filename buffer
1313. The request addresses a CGI resource, Lua script or * server-side javascript. * The URI corresponds to the script itself (like * /path/script.cgi), and there is no additional resource * path (like /path/script.cgi/something). * Requests that modify (replace or delete) a resource, like * PUT and DELETE requests, should replace/delete the script * file. * Requests that read or write from/to a resource, like GET and * POST requests, should call the script and return the * generated response.
1314. Internal function. Assumes conn is valid
1315. Send read bytes to the client, exit the loop on error
1316. HTTP 1.1 assumes keep alive if "Connection:" header is not set * This function must tolerate situations when connection info is not * set up, for example if request parsing failed.
1317. The rest of the line is the value
1318. Use read() instead of fread(), because if we're reading from the * CGI pipe, fread() may block until IO buffer is filled up. We * cannot afford to block and must pass all read bytes immediately * to the client.
1319. Move the queue forward len bytes
1320. Control characters are not allowed, including zero
1321. Skip all following slashes, backslashes and double-dots
1322. nonce is from a previous start of the server and no longer valid * (replay attack?)
1323. **comment:** Initialize this library. This function does not need to be thread safe.
 label: requirement
1324. RFC3229 Section 10.4.1
1325. END: CLOCK_MONOTONIC
1326. POSIX return values: see * <http://pubs.opengroup.org/onlinepubs/9699919799/functions/freeaddrinfo.html>
1327. server context
1328. vsnprintf must not be used in any system, * \\ \\ * but this define only works well for Windows.
1329. NOTE: override filep->size only on success. Otherwise, it * might * break constructs like if (!mg_stat() || !mg_fopen()) ...
1330. RFC2616 Section 10.5.1
1331. RFC 2817 Section 4
1332. 1.4. clean URIs, so a path like allowed_dir/../forbidden_file is * not possible
1333. RFC2616 Section 10.4.8
1334. RFC2518 Section 10.6, RFC4918 * Section 11.5
1335. **comment:** TODO: deal with legacy
 label: requirement
1336. Using filename_buf_len - 1 because memmove() for PATH_INFO may shift * part of the path one byte on the right.
1337. The mg_upload function is superseded by mg_handle_form_request.
1338. 5.1. first test, if the request targets the regular http(s)::/* protocol namespace or the websocket ws(s)::/* protocol namespace.
1339. Step 6: Determine the local file path from the root path and the * request uri.
1340. Handler for specific error, e.g. 404 error
1341. Copy the stuff to temporary file
1342. Print first entry - link to a parent directory
1343. Not required for non-blocking sockets. set_sock_timeout(conn->client.sock, timeout);
1344. Sanity check the offset
1345. Mask for all nonce values
1346. Allocation failed. Return -1 as "out of memory" error.
1347. dummy for SSL argument to push/pull
1348. inline 7-bit length field
1349. No truncation check for ebuf
1350. Copy socket from the queue and increment tail
1351. pthread mutex not available
1352. Adjust number of bytes to read.
1353. OPENSSL_API_1_1
1354. Unknown request error code
1355. assert(auth_handler == NULL);
1356. Other status codes, not shown in the IANA HTTP status code * assignment. * E.g., "de facto" standards due to common use, ...
1357. Loop continuously, reading messages from the socket, invoking the * callback, and waiting repeatedly until an error occurs.
1358. out: handled by a script?
1359. Lua server page: an SSI like page containing mostly plain * html * code * plus some tags with server generated contents.
1360. SSL callback documentation: * https://www.openssl.org/docs/man1.1.0/ssl/SSL_set_info_callback.html * https://linux.die.net/man/3/ssl_set_info_callback
1361. fetch a new chunk
1362. Error sending the body data
1363. put_dir returns -2 if the directory can not be created
1364. reset all members of fileacc
1365. non-constant aggregate initializer: issued due to missing C99 support
1366. pem is not NULL here
1367. Execution time information
1368. 13. Handle other methods than GET/HEAD
1369. Ifsr will be only 0 if has not been initialized, * so this code is called only once.
1370. unspecified - may change with the next version

1371. First check if it is a list or just a single value.
1372. could be deprecated global callback
1373. SSL function name
1374. Substitute file is a script file
1375. Ignore any empty entries.
1376. Skip the characters until one of the delimiters characters found. * 0-terminate resulting word. Skip the delimiter and following whitespaces. * Advance pointer to buffer to the next word. Return found 0-terminated * word. * Delimiters can be quoted with quotechar.
1377. USE_TIMERS
1378. **comment:** Some proper reactions would be: * a) close the connections without sending anything * b) send a 404 not found * c) try if there is a file matching the URI * It would be possible to do a, b or c in the callback * implementation, and return 1 - we cannot do anything * here, that is not possible in the callback. **
TODO: What would be the best reaction here? * (Note: The reaction may change, if there is a better *idea.)
label: code-design
1379. Request is directed to a different port
1380. Copy the mask before we shift the queue and destroy it
1381. Errors 1xx, 204 and 304 MUST NOT send a body
1382. next handler in a linked list
1383. WinCE has no pipes
1384. Prepare full path to the index file
1385. We use (signed) cast below because MSVC 6 compiler cannot * convert unsigned __int64 to double. Sigh.
1386. -1 for path too long, * -2 for path can not be created.
1387. Invalid request
1388. Use mg_cry here, since gpass has been configured.
1389. Quick fix (for 1.9.x):
1390. defined(_WIN32) && !defined(__SYMBIAN32__) - \ WINDOWS / UNIX include block
1391. ssl already initialized
1392. Unread data from the last chunk
1393. 13.2. Handle OPTIONS for files
1394. Set linger option according to configuration
1395. End of headers reached.
1396. BEGIN: CLOCK_REALTIME = wall clock (date and time)
1397. RFC2616 Section 10.5.2
1398. RFC2616 Section 10.3.8
1399. Thread index within ctx
1400. Error while parsing headers
1401. If there is no substitute file, the server could return * a directory listing in a later step
1402. DTL -- including winsock2.h works better if lean and mean
1403. USE_WEBSOCKET
1404. Local file path and name, corresponding to requested URI * is now stored in "filename" variable.
1405. failed
1406. Convert time_t to a string. According to RFC2616, Sec 14.18, this must be * included in all responses other than 100, 101, 5xx.
1407. Translate serial number to a hex string
1408. SEARCH (RFC 5323)
1409. File properties filled by mg_stat:
1410. handler type
1411. image
1412. Important: using local struct mg_file to test path for * is_directory flag. If filep is used, mg_stat() makes it * appear as if auth file was opened. * TODO(mid): Check if this is still required after rewriting * mg_stat
1413. first try for an exact match
1414. see CONNECTION_TYPE_* above
1415. Opcode == 8, connection close
1416. **comment:** TODO: handle error
label: requirement
1417. For fseeko(), ftello()
1418. Yes, they are mandatory
1419. gai_strerror could be used to convert gai_ret to a string
1420. defined(USE_STACK_SIZE) && USE_STACK_SIZE > 1
1421. Support PATH_INFO for CGI scripts.
1422. Copyright (c) 2013-2013 the Civetweb developers * Copyright (c) 2004-2013 Sergey Lyubka ** Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: ** The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. ** THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
1423. Handler for all errors
1424. Reject wrong versions
1425. Include '\n'
1426. A relative uri starts with a / character
1427. Find end of status text
1428. No error, but 0 bytes sent. May be EOF?
1429. File does not exist. This is not always a problem here.
1430. On the fly compression allowed
1431. RFC2616 Section 10.5.6
1432. If we compiled with Mac OSX 10.12 or later, then clock_gettime will be * declared but it may be NULL at runtime. So we need to check before using * it.
1433. **comment:** not registered types * (<http://reference.sitepoint.com/html/mime-types-full>, * http://www.hansenb.pdx.edu/DMKB/dict/tutorials/mime_typ.php, ..)
label: documentation
1434. Open file for writing, create and overwrite
1435. RFC2616 Section 10.4.4
1436. Read the server config to check how long a file may be cached. * The configuration is in seconds.
1437. file is on disk
1438. Server is to be stopped.
1439. Listening socket
1440. This is an unknown version
1441. Delete not successful (file locked).
1442. find a free worker slot and signal it
1443. WinCE does not support CGI pipes

1444. **comment:** TODO (low): Replace "membuf" implementation by a "file in memory" * support library. Use some struct mg_file_in_memory *mf; instead of * membuf char pointer.
label: code-design

1445. File is already compressed. No "on the fly" compression.

1446. Copy remaining data

1447. Not an SSI tag

1448. some data has been read, or no data was requested

1449. This structure helps to create an environment for the spawned CGI * program. * Environment is an array of "VARIABLE=VALUE\0" ASCIIZ strings, * last element must be NULL. * However, on Windows there is a requirement that all these * VARIABLE=VALUE\0 * strings must reside in a contiguous buffer. The end of the buffer is * marked by two '\0' characters. * We satisfy both worlds: we create an envp array (which is vars), all * entries are actually pointers inside buf.

1450. handle request to local server

1451. This thread has been removed from cv's waiting list

1452. Step 7: URI rewriting

1453. one worker thread will be created

1454. Space available in buf

1455. dlsym() on UNIX returns void *. ISO C forbids casts of data * pointers to function pointers. We need to use a union to make a * cast.

1456. Disable deprecation warning in VS2005

1457. Use the global passwords file, if specified by auth_gpass option, * or search for .htpasswd in the requested directory.

1458. RFC2616 Section 10.4.3

1459. 16 = Max. thread length in Linux/OSX/..

1460. Check license (zlib has a permissive license, but

1461. __SYMBIAN32__

1462. 6.2.1. thus, the server must have real files

1463. Request is directed to another server: * The server name is different.

1464. Call ductape to generate the page

1465. The chain of threads

1466. Bind to IPv4 only, since IPv6 is not built in.

1467. Return at least a category according to RFC 2616 Section 10.

1468. Accepted sockets

1469. defined for tdm-gcc so we can use getnameinfo

1470. The max request size

1471. RFC2518 Section 10.4, RFC4918 Section 11.3

1472. assert(conn->consumed_content == 0);

1473. The request sent by the client could not be understood by * the server, or it was incomplete or a timeout. Send an * error message and close the connection.

1474. Implementation of POSIX opendir/closedir/readdir for Windows.

1475. Do not lock when getting the callback value, here and below. * I suppose this is fine, since function cannot disappear in the * same way string option can.

1476. The pre-allocated buffer is large enough. * Use it to store the string and return the address.

1477. Initialize output string

1478. 6.2. this request is a PUT/DELETE to a real file

1479. WinCE has no popen

1480. Poll returned either success (1) or error (-1). * Forward both to the caller.

1481. output: socket, must not be NULL

1482. The number of configured worker threads.

1483. RFC2616 Section 10.4.10

1484. Full request not yet received

1485. 6.1. a custom authorization handler is installed

1486. Deallocate SSL context

1487. Cannot determine if socket is already closed. This should * not occur and never did in a test. Log an error message * and continue.

1488. Is valid

1489. function has been selected for automatic inline expansion

1490. Initialize locking callbacks, needed for thread safety. * <http://www.openssl.org/support/faq.html#PROG1>

1491. file does not exist and will not be created

1492. 3 indicates a websocket client thread

1493. GCC_VERSION >= 40500

1494. This will fail file is the file is in memory

1495. Make sure child closes all pipe descriptors. It must dup them to 0,1

1496. Using fmt as a non-literal is intended here, since it is mostly called * indirectly by mg_snprintf

1497. Note that POSIX/Winsock's send() is threadsafe * <http://stackoverflow.com/questions/1981372/are-parallel-calls-to-send-recv-on-the-same-socket-valid> * but mongoose's mg_printf/mg_write is not (because of the loop in * push(), although that is only a problem if the packet is large or * outgoing buffer is full).

1498. C callback has returned non-zero, do not proceed with * handshake.

1499. Apple OSX section

1500. **comment:** Convert variable name into uppercase, and change - to _
label: code-design

1501. Reset ip_version to 0 of there is an error

1502. on linux we can use the old prctl function

1503. **comment:** Todo: check if cors_origin is in cors_orig_cfg. * Or, let the client check this.
label: requirement

1504. TCP_USER_TIMEOUT/RFC5482 (<http://tools.ietf.org/html/rfc5482>): * max. time waiting for the acknowledged of TCP data before the connection * will be forcefully closed and ETIMEDOUT is returned to the application. * If this option is not set, the default timeout of 20-30 minutes is used.

1505. Default is "no".

1506. Since we use a sleep quantum of some seconds to check for recv * timeouts, we will just wait a few seconds in mg_join_thread.

1507. Free existing block

1508. conditional expression is constant: introduced by FD_SET(..)

1509. Verify given socket address against the ACL. * Return -1 if ACL is malformed, 0 if address is disallowed, 1 if allowed.

1510. 10. Request is handled by a script

1511. Handle SSI tag

1512. Not yet a valid request/response

1513. This legacy interface only works for the IPv4 case

1514. **comment:** Join all worker threads to avoid leaking threads.
label: code-design

1515. Cross-origin resource sharing (CORS), see * <http://www.html5rocks.com/en/tutorials/cors/>, * http://www.html5rocks.com/static/images/cors_server_flowchart.png * - * preflight is not supported for files.

1516. RFC7540 Section 9.1.2

1517. Incomplete request

1518. We need to get the client thread out of the select/recv call * here.

1519. Get connection information. It can be printed or stored by the caller. * Return the size of available information.

1520. !_WIN32_WCE

1521. File did not exist before fopen was called. * Maybe it has been created now. Get stat info * like creation time now.

1522. if this file is in fact a pre-gzipped file, rewrite its filename * it's important to rewrite the filename after resolving * the mime type from it, to preserve the actual file's type
1523. convert 1-3 remaining bytes if ((dataLen % 4) != 0)
1524. Initialize SSL library
1525. TODO: SSL_CTX_set_verify(conn->client_ssl_ctx, * SSL_VERIFY_PEER, verify_ssl_server);
1526. Response code not even within reasonable range
1527. Terminate and forward to the next word
1528. absolute uri (with/without port)
1529. There is no callback, and Lua is not responsible either.
1530. 11.2. DELETE method
1531. _MSC_VER
1532. For files, just wait a fixed time, * maybe an average disk seek time.
1533. Unsupported WEBDAV Methods:
1534. 4. Check for CORS preflight requests and handle them (if configured). * https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
1535. Valid request
1536. Forward until a space is found - use isgraph here
1537. If callback returns 0, civetweb sets up the SSL certificate. * If it returns 1, civetweb assumes the callback already did this. * If it returns -1, initializing ssl fails.
1538. forward_body_data failed. * The error code has already been sent to the client, * and conn->status_code is already set.
1539. Return null terminated string of given maximum length. * Report errors if length is exceeded.
1540. Data information
1541. assert(discard_len >= 0);
1542. Should we support client certificates?
1543. Create the file if does not exist
1544. Buffer for received data
1545. Read and discard pending incoming data. If we do not do that and * close * the socket, the data in the send buffer may be discarded. This * behaviour is seen on Windows, when client keeps sending data * when server decides to close the connection; then when client * does recv() it gets no data back.
1546. Terminate string and forward buf to next line
1547. **comment:** TODO: not thread safe
label: requirement
1548. **comment:** TODO: add interface to compression module
label: code-design
1549. Send the rest of buffered data
1550. Currently CivetWeb does not need read+write access.
1551. closing
1552. assert(buffered_len >= 0);
1553. **comment:** TODO: Check if this lock should be moved to user land. * Currently the server sets this lock for websockets, but * not for any other connection. It must be set for every * conn read/written by more than one thread, no matter if * it is a websocket or regular connection.
label: code-design
1554. 0 means "do not cache". All values <0 are reserved * and may be used differently in the future.
1555. If we're stopping, sq_head may be equal to sq_tail.
1556. Reply with a 404 Not Found. We are still at a standard * HTTP request here, before the websocket handshake, so * we can still send standard HTTP error replies.
1557. <inttypes.h> wants this for C++
1558. Check whether full request is buffered. Return: * -1 if request or response is malformed * 0 if request or response is not yet fully buffered * >0 actual request length, including last \r\n\r\n
1559. Use global passwords file
1560. Close write end fdin and read end fdout and fderr
1561. output: socket address, must not be NULL
1562. **comment:** TODO: POSIX returns either EAGAIN or EWOULDBLOCK in both cases, * if the timeout is reached and if the socket was set to non- * blocking in close_socket_gracefully, so we can not distinguish * here. We have to wait for the timeout in both cases for now.
label: code-design
1563. this index file is a script
1564. Try to create intermediate directory
1565. All threads exited, no sync is needed. Destroy thread mutex and * condvars
1566. Linear feedback shift register
1567. Local socket address
1568. Function pointer
1569. init
1570. According to the HTTP standard * <http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.1.2> * URI can be an asterisk (*) or should start with slash (relative uri), * or it should start with the protocol (absolute uri).
1571. Determine 32/64 bit data mode. * see https://en.wikipedia.org/wiki/64-bit_computing
1572. -2 means "3rd party thread"
1573. in ms
1574. Search for \n from p till the end of stream
1575. Valid listening port specification is: [ip_address:]port[s] * Examples for IPv4: 80, 443s, 127.0.0.1:3128, 192.0.2.3:8080s * Examples for IPv6: [::]:80, [::]:80, * [2001:0db8:7654:3210:FEDC:BA98:7654:3210]:443s * see <https://tools.ietf.org/html/rfc3513#section-2.2> * In order to bind to both, IPv4 and IPv6, you can either add * both ports using 8080,[::]:8080, or the short form +8080. * Both forms differ in detail: 8080,[::]:8080 create two sockets, * one only accepting IPv4 the other only IPv6. +8080 creates * one socket accepting IPv4 and IPv6. Depending on the IPv6 * environment, they might work differently, or might not work * at all - it must be tested what options work best in the * relevant network environment.
1576. Bytes sent this second
1577. <http://sourceforge.net/p/predef/wiki/Compilers/>
1578. If the queue is full, wait
1579. Used nonces, used for authentication
1580. 2147479552 (0x7FFF000) is a limit found by experiment on * 64 bit Linux (2^31 minus one memory page of 4k?).
1581. common use
1582. Check for recursion
1583. Fill in IP, port info early so even if SSL setup below fails, * error handler would have the corresponding info. * Thanks to Johannes Winkelmann for the patch.
1584. Send FIN to the client
1585. defined(_WIN32_WCE)
1586. Read file, byte by byte, and look for SSI include tags
1587. Only in case n=0 (timeout), repeat calling the write function
1588. <http://www.webdav.org/specs/rfc4918.html>, 9.1: * When MKCOL is invoked without a request body, * the newly created collection SHOULD have no * members. A MKCOL request message may contain * a message body. The precise behavior of a MKCOL * request when the body is present is undefined, * ... ==> We do not support MKCOL with body data. * This method is idempotent, but not safe (see * Section 9.1 of [RFC2616]). Responses to this * method MUST NOT be cached.
1589. This is an already existing directory, * so there is nothing to do for the server.
1590. path too long
1591. If a file is in memory, set all "stat" members and the membuf pointer of * output filep and return 1, otherwise return 0 and don't modify anything.
1592. Try to find .htpasswd in requested directory.

1593. PATH_INFO part of the URL
1594. MacOS needs that. If we do not zero it, subsequent bind() will fail. * Also, all-zeroes in the socket address means binding to all addresses * for both IPv4 and IPv6 (INADDR_ANY and IN6ADDR_ANY_INIT).
1595. Similar array as ssl_sw. These functions could be located in different * lib.
1596. **comment:** Disable bogus compiler warning -Wdate-time
 label: code-design
1597. stdout read
1598. After exec, all signal handlers are restored to their default * values, with one exception of SIGCHLD. According to * POSIX.1-2001 and Linux's implementation, SIGCHLD's handler will * leave unchanged after exec if it was set to be ignored. Restore * it to default action.
1599. RFC2616 Section 10.3.2
1600. EINVAL - Clock ID is unknown
1601. non-ascii characters must be % encoded
1602. No custom error page. Send default error page.
1603. !_WIN32
1604. end of list
1605. Translate subject and issuer to a string
1606. Number of variables available in var
1607. current mask and opcode, overwritten by * memmove()
1608. SSL called from an unknown thread: Create some thread index.
1609. **comment:** Short name is used.
 label: code-design
1610. The maximum length of the path to the password file is limited
1611. e.g., def from https://zlib.net/zlib_how.html
1612. Number of variables stored in var
1613. Only for memory statistics
1614. 64-bit length field
1615. 124
1616. Writes PROPFIND properties for a collection element
1617. 13.3. everything but GET and HEAD (e.g. POST)
1618. #define TCP_USER_TIMEOUT (18)
1619. Parse authorization header
1620. **comment:** We don't use a lock here. Calling mg_stop with the same ctx from * two threads is not allowed.
 label: code-design
1621. Not breaking the loop, process next websocket frame.
1622. non-script files will not have sub-resources
1623. Server side nonce check is valuable in all situations but one: * if the server restarts frequently, but the client should not see * that, so the server should accept nonces from previous starts.
1624. Remote user name
1625. Add data to buffer
1626. **comment:** Do not allow control characters like newline in user name and domain. * Do not allow excessively long names either.
 label: code-design
1627. 96
1628. While getaddrinfo on Windows will work with [::1], * getaddrinfo on Linux only works with ::1 (without []).
1629. File is open for writing. If fclose fails, there was probably an * error flushing the buffer to disk, so the file on disk might be * broken. Delete it and return an error to the caller.
1630. RFC2616 Section 10.4.9
1631. Ignore errors. We can't call * mg_cry here anyway ;-)
1632. RFC2616 Section 10.4.17
1633. Is port SSL-ed
1634. Out of range
1635. Lua in-server module script: a CGI like script used to * generate * the * entire reply.
1636. out: filename
1637. !defined(NO_FILES)
1638. Close so child gets an EOF.
1639. Index block
1640. 'type cast': conversion from 'int' to 'HANDLE' of greater size
1641. Free the request buffer.
1642. Other arguments must not be empty
1643. Poll returned timeout (0).
1644. Make a pointer to the free space int the buffer
1645. fonts
1646. to close
1647. If a Lua background script has been configured, start it.
1648. It could be the hixie draft version * (<http://tools.ietf.org/html/draft-hixie-thewebsccketprotocol-76>).
1649. HTTP 1.0 (and earlier) default is to close the connection
1650. Allocate new block
1651. What operating system is running
1652. Head of the socket queue
1653. may be null
1654. no function prototype given: converting '()' to '(void)'
1655. 60
1656. We are not in a tag yet.
1657. regular end of content
1658. first time reading from this connection
1659. Windows is not standard-compliant, and vsnprintf() returns -1 if * buffer is too small. Also, older versions of msrvct.dll do not have * _vscprintf(). However, if size is 0, vsnprintf() behaves correctly. * Therefore, we make two passes: on first pass, get required message * length. * On second pass, actually print the message.
1660. Windows and Visual Studio Compiler
1661. Read the rest of CGI output and send to the client
1662. This is the heart of the Civetweb's logic. * This function is called when the request is read, parsed and validated, * and Civetweb must decide what action to take: serve a file, or * a directory, or call embedded function, etcetera.
1663. Describes a string (chunk of memory).
1664. Multiple protocols -> accept the last one.
1665. Read the nonce from the response.
1666. Substitute files have already been handled above.
1667. ACL (RFC 3744)
1668. User-defined data
1669. must be a end of a word, but not a line
1670. #define SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS 0x0001 ssl->s3->flags |= SSL3_FLAGS_NO_RENEGOTIATE_CIPHERS;

1671. First check if the port is the same (IPv4 and IPv6).
1672. Space taken in buf
1673. ws/wss client
1674. For PHP
1675. request is authorized or does not need authorization
1676. mg_fopen will open a file either in memory or on the disk. * The input parameter path is a string in UTF-8 encoding. * The input parameter mode is MG_FOPEN_MODE_* * On success, either fp or membuf will be set in the output * struct file. All status members will also be set. * The function returns 1 on success, 0 on error.
1677. Do an unsigned comparison in some conditions below
1678. some intermediate directory has an index file
1679. System info
1680. illegal character for chunk length
1681. HCP24: now check is it a substring or a full cookie name
1682. RFC2616 Section 10.5 - Server Error 5xx
1683. If SSL is loaded dynamically, dlopen/dlclose is required.
1684. not consumed
1685. assert(is_delete_request || (handler!=NULL));
1686. everything is invalid for the moment (might change in the * future)
1687. The support for duktape is still in alpha version state. * The name of this config option might change.
1688. connected with IPv4
1689. SSL loaded dynamically from DLL. * I put the prototypes here to be independent from OpenSSL source * installation.
1690. Connections information
1691. 62
1692. Step 3.1: Check if Lua is responsible.
1693. assert(auth_handler != NULL);
1694. Store the crypto library handle.
1695. In contrast to OpenSSL, wolfSSL does not support certificate * chain files that contain private keys and certificates in * SSL_CTX_use_certificate_chain_file. * The CivetWeb-Server used pem-Files that contained both information. * In order to make wolfSSL work, it is split in two files. * One file that contains key and certificate used by the server and * an optional chain file for the ssl stack.
1696. Sort and print directory entries
1697. A http to https forward port has been specified, * but no https port to forward to.
1698. The protocol is a comma separated list of names.
1699. **comment:** Who on earth came to the conclusion, using __DATE__ should rise * an "expansion of date or time macro is not reproducible" * warning. That's exactly what was intended by using this macro. * Just disable this nonsense warning.
label: code-design
1700. NOTE(lsm): order is important here. SSL certificates must * be initialized before listening ports. UID must be set last.
1701. **comment:** TODO
label: requirement
1702. stderr read
1703. all file related outputs have already been set to 0, just return
1704. Write data to the IO channel - opened file descriptor, socket or SSL * descriptor. * Return value: * >=0 .. number of bytes successfully written * -1 .. timeout * -2 .. error
1705. If there is a connection header from the client, obey
1706. If no index file exists, restore directory path
1707. in: request (must be valid)
1708. SSL functions may fail and require to be called again: * see https://www.openssl.org/docs/manmaster/ssl/SSL_get_error.html * Here "func" could be SSL_connect or SSL_accept.
1709. Callback for the master thread (type 0)
1710. If only port is specified, bind to IPv4, INADDR_ANY
1711. **comment:** filename and conn could be unused, if all preprocessor conditions * are false (no script language supported).
label: code-design
1712. Time (wall clock) when connection was * established
1713. Deallocate worker thread ID array
1714. 1 if in read_websocket
1715. Free certificate memory
1716. will not overflow
1717. fclose failed. This might have different reasons, but a likely * one is "no space on disk", http 507.
1718. max 64 headers
1719. Important! When having connections with and without auth * would cause double free and then crash
1720. 92
1721. + MicroSoft extensions * <https://msdn.microsoft.com/en-us/library/aa142917.aspx>
1722. since the return value is * int, we may not read more * bytes
1723. Client sent an "Expect: xyz" header and xyz is not 100-continue.
1724. We do not set a "Cache-Control" header here, but leave the default. * Since browsers do not send an OPTIONS request, we can not test the * effect anyway.
1725. Use some UID as session context ID.
1726. Set stop flag, so all threads know they have to exit.
1727. **comment:** Call poll, but only for a maximum time of a few seconds. * This will allow to stop the server after some seconds, instead * of having to wait for a long socket timeout.
label: code-design
1728. 11. File does not exist, or it was configured that it should be * hidden
1729. the file is cached in memory
1730. Read from file, exit the loop on error
1731. civetweb should process the request
1732. Use case sensitive compare function
1733. State
1734. Exit the loop if callback signals to exit (server side), * or "connection close" opcode received (client side).
1735. RFC2616 Section 10.2.7
1736. Lua rejected the new client
1737. If send failed, wait before retry
1738. current mask and opcode
1739. See <http://www.cplusplus.com/reference/cctype/>
1740. Worker threads take accepted socket from the queue
1741. File name is relative to the current document
1742. Prepended CONFIG_ to avoid conflict with the * socket option typedef TCP_NODELAY.
1743. Authorize against the opened passwords file. Return 1 if authorized.
1744. __WIN32 / else
1745. Use default of the standard
1746. Handler for ws/wss (websocket) requests.

1747. client context: loops must end
1748. **comment:** conn is currently unused
 label: code-design
1749. Append a pointer to the added string into the envp array
1750. !NO_FILES
1751. **comment:** Avoid unused warning if NO_SSL is set and DEBUG_TRACE is not used
 label: code-design
1752. First store the length information in a text buffer.
1753. Tag is still open
1754. This included all leading \r and \n (isspace)
1755. Step 1: Check websocket protocol version.
1756. See #199 (<https://github.com/civetweb/civetweb/issues/199>)
1757. Parent closes only one side of the pipes. * If we don't mark them as closed, close() attempt before * return from this function throws an exception on Windows. * Windows does not like when closed descriptor is closed again.
1758. Request info
1759. Connection info
1760. RFC2616 Section 10.4.14
1761. RFC2616 Section 10.4.11
1762. actually, range requests don't play well with a pre-gzipped * file (since the range is specified in the uncompressed space)
1763. Civetweb configuration parameters
1764. **comment:** TODO (low): check if this is still required
 label: requirement
1765. No handler set - assume "OK"
1766. The rest of the line is the status text
1767. This following lines are just meant as a reminder to use the mg-functions * for memory management
1768. RFC5842 Section 7.1
1769. RFC2616 Section 10.3.3
1770. No need to continue processing files once we have a * match, since nothing will reset it back * to 0.
1771. The "file in memory" feature is a candidate for deletion. * Please join the discussion at * <https://groups.google.com/forum/#topic/civetweb/h9HT4CmeYql>
1772. unknown content length
1773. CGI needs it as REMOTE_USER
1774. RFC2616 Section 10.2.2
1775. Valid response
1776. Next character after the port number
1777. State as string
1778. Tail of the socket queue
1779. Calculate how much space is left in the buffer
1780. We already know there is no more data buffered in conn->buf * but there is more available in the SSL layer. So don't poll * conn->client.sock yet.
1781. NOTE(lsm): on QNX, poll() returns POLLRDNORM after the * successful poll, and POLLIN is defined as * (POLLRDNORM | POLLRDBAND) * Therefore, we're checking pfd[i].revents & POLLIN, not * pfd[i].revents == POLLIN.
1782. HPUX defines socklen_t incorrectly as size_t which is 64bit on * Itanium. Without defining _XOPEN_SOURCE or _XOPEN_SOURCE_EXTENDED * the prototypes use int* rather than socklen_t* which matches the * actual library expectation. When called with the wrong size arg * accept() returns a zero client inet addr and check_acl() always * fails. Since socklen_t is widely used below, just force replace * their typedef with int. - DTL
1783. No conn
1784. It was defined for WEBDAV in RFC 3253, Sec. 3.6 * (<https://tools.ietf.org/html/rfc3253#section-3.6>), but seems * to be useful for REST in case a "GET request with body" is * required.
1785. WEBDAV (RFC 2518)
1786. !defined(NO_CGI) || defined(USE_LUA) || defined(USE_DUKTAPE)
1787. Retrieve requested HTTP header multiple values, and return the number of * found occurrences
1788. stderr write
1789. Build date
1790. Should we stop event loop
1791. process HTTPS connection
1792. filep is initialized in mg_stat: all fields with memset to, * some fields like size and modification date with values
1793. Destroy other context global data structures mutex
1794. 8. handle websocket requests
1795. 5.2.1. A callback will handle this request. All requests * handled * by a callback have to be considered as requests to a script * resource.
1796. no can do
1797. **comment:** Depending on USE_WEBSOCKET and NO_SSL, some of the protocols might be * not supported. Clang raises an "unreachable code" warning for parts of ?: * unreachable, but splitting into four different #ifdef clauses here is more * complicated.
 label: code-design
1798. Loop over passwords file
1799. 13.1. Handle PROPFIND
1800. !defined(NO_THREAD_NAME)
1801. **comment:** GCC unused function attribute seems fundamentally broken. * Several attempts to tell the compiler "THIS FUNCTION MAY BE USED * OR UNUSED" for individual functions failed. * Either the compiler creates an "unused-function" warning if a * function is not marked with __attribute__((unused)). * On the other hand, if the function is marked with this attribute, * but is used, the compiler raises a completely idiotic * "used-but-marked-unused" warning - and * #pragma GCC diagnostic ignored "-Wused-but-marked-unused" * raises error: unknown option after #pragma GCC diagnostic*. * Disable this warning completely, until the GCC guys sober up * again.
 label: code-design
1802. Step 7: Enter the read loop
1803. 11.3. MKCOL method
1804. linked list of shared lua websockets
1805. **comment:** clean zombies
 label: code-design
1806. Read the first line of the script into the buffer
1807. RFC2518 Section 10.5, RFC4918 * Section 11.4
1808. Check after term
1809. **comment:** The "non blocking" property should already be * inherited from the parent socket. Set it for * non-compliant socket implementations.
 label: code-design
1810. Substitute file found. Copy stat to the output, then * check if the file is a script file
1811. 5. interpret the url to find out how the request must be handled
1812. **comment:** TODO(lsm, low): propagate an error to the caller
 label: code-design
1813. Start worker threads
1814. relative uri
1815. **comment:** Unused.
 label: code-design

1816. Don't use mg_cry here, but only a trace, since this is * a typical case. It will occur for every directory * without a password file.
1817. Close, if civetweb framework needs to close
1818. HPUX 11 does not have monotonic, fall back to realtime
1819. No more data left to read
1820. Start WinSock for Windows
1821. Increase priority of the master thread
1822. Windows specific
1823. 5.2.2. No callback is responsible for this request. The URI * addresses a file based resource (static content or Lua/cgi * scripts in the file system).
1824. Get the next step of both random number generators.
1825. RFC2616 Section 10.2.6
1826. Un-initialize this library.
1827. Wait for a thread to finish.
1828. DTL add for SO_EXCLUSIVE
1829. Add all headers as HTTP_* variables
1830. NOTE(lsm): this enum should be in sync with the config_options below.
1831. conn is assumed to be valid in this internal function
1832. not OPENSSL_API_1_1
1833. Test for different ways to format this string
1834. Allocate space to hold websocket payload
1835. Not connected
1836. URI: /path_to_script/script.cgi/path_info
1837. Step 3: No callback. Check if Lua is responsible.
1838. **comment:** TODO: check if conn->ctx can be set
 label: requirement
1839. For flockfile() on Linux
1840. Implementation of API function for HTTP clients
1841. Mark the queue as advanced
1842. END: CLOCK_PROCESS
1843. Total size of data in a buffer
1844. 9b. This request is either for a static file or resource handled * by a script file. Thus, a DOCUMENT_ROOT must exist.
1845. Handler for authorization requests
1846. first try to find an existing handler
1847. If pem is NULL and conn->ctx->callbacks.init_ssl is not, * refresh_trust still can not work.
1848. Request is directed to this server - full name match.
1849. the value is not required
1850. No content
1851. + 11 methods from RFC 3253
1852. **comment:** goto spawn_cleanup;
 label: code-design
1853. 7. check if there are request handlers for this uri
1854. **comment:** TODO(Feature): this is no longer a boolean, but yes/no/optional
 label: code-design
1855. END: CLOCK_THREAD
1856. **comment:** TODO (mid): Define proper return values - maybe return length? * For the first test use <0 for error and >0 for OK
 label: code-design
1857. Set the thread name for debugging purposes in Visual Studio * <http://msdn.microsoft.com/en-us/library/xcb2z8hs.aspx>
1858. Set TCP keep-alive. This is needed because if HTTP-level * keep-alive * is enabled, and client resets the connection, server won't get * TCP FIN or RST and will keep the connection open forever. With * TCP keep-alive, next keep-alive handshake will figure out that * the client is down and will close the server end.
 Thanks to Igor Klopov who suggested the patch.
1859. Substitute file is a regular file
1860. **comment:** Disable TCP Nagle's algorithm. Normally TCP packets are coalesced * to effectively fill up the underlying IP packet payload and * reduce the overhead of sending lots of small buffers. However * this hurts the server's throughput (ie. operations per second) * when HTTP 1.1 persistent connections are used and the responses * are relatively small (eg. less than 1400 bytes).
 label: code-design
1861. out: put/delete a file?
1862. Check for a valid http method
1863. The server must only return one value from this list.
1864. Windows happily opens files with some garbage at the end of file name. * For example, fopen("a.cgi ", "r") on Windows successfully opens * "a.cgi", despite one would expect an error back. * This function returns non-0 if path ends with some garbage.
1865. If name=" is found, search for the closing "
1866. Read new data
1867. There are three ways to encode data from a HTML form: * 1) method: GET (default) * The form data is in the HTTP query string. * 2) method: POST, enctype: "application/x-www-form-urlencoded" * The form data is in the request body. * The body is url encoded (the default encoding for POST). * 3) method: POST, enctype: "multipart/form-data". * The form data is in the request body of a multipart message. * This is the typical way to handle file upload from a form.
1868. If the boundary is already in the buffer, get the address, * otherwise next will be NULL.
1869. GET request: form data is in the query string.
1870. Get the mandatory name="..." part of the Content-Disposition * header.
1871. **comment:** TODO: check Content-Type
 label: requirement
1872. No query string.
1873. Copy boundary string to variable "boundary"
1874. **comment:** Sanity check: The algorithm can not work if bl >= sizeof(buf), * and it will not work effectively, if the buf is only a few byte * larger than bl, or if buf can not hold the multipart header * plus the boundary. * Check some reasonable number here, that should be fulfilled by * any reasonable request from every browser. If it is not * fulfilled, it might be a hand-made request, intended to * interfere with the algorithm.
 label: code-design
1875. No body data, but not a GET request. * This is not a valid form request.
1876. **comment:** Sanity check: A boundary string of less than 4 bytes makes * no sense either.
 label: code-design
1877. Store the content of the buffer.
1878. Read body data and split it in keys and values. * The encoding is like in the "GET" case above: a=1&b=c=3&c=4. * Here we use "POST", and read the data from the request body. * The data read on the fly, so it is not required to buffer the * entire request in memory before processing it.
1879. **comment:** The initial sanity check * (bl + 800 > sizeof(buf)) * is no longer required, since sizeof(buf) == 1024 * * Original comment:
 label: code-design
1880. Malformed request (the filename field is optional, but if * it exists, it needs to be terminated correctly).
1881. Malformed request
1882. Unknown Content-Type
1883. Call callback
1884. }

1885. **comment:** And others complain, the result is unused.
label: code-design
1886. All parts handled
1887. We must do a binary search here, not a string search, since the buffer * may contain '\x00' bytes, if binary data is transferred.
1888. Copyright (c) 2016-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
1889. RFC 2616 Sec. 2.2 defines a list of allowed * separators, but many of them make no sense * here, e.g. various brackets or slashes. * If they are used, probably someone is * trying to attack with curious hand made * requests. Only ; , space and tab seem to be * reasonable here. Ignore everything else.
1890. End of handle_form.inl
1891. If the boundary is quoted, trim the quotes
1892. Next, we need to get the part header: Read until \r\n\r\n
1893. TODO: Create a function to get "all_data_read" from * the conn object. All data is read if the Content-Length * has been reached, or if chunked encoding is used and * the end marker has been read, or if the connection has * been closed.
1894. RFC 2046 permits the boundary string to be quoted.
1895. Store the content to a file
1896. if (field_storage == FORM_FIELD_STORAGE_READ) {
1897. In theory, it could be possible that someone crafts * a request like name=filename=xyz. Check if name and * filename do not overlap.
1898. Log error message and stop parsing the form data.
1899. Skip \r\n\r\n
1900. There has to be a BOUNDARY definition in the Content-Type header
1901. No data
1902. name= without quotes is also allowed
1903. Try the same without quotes
1904. Out of memory
1905. In every "field_found" callback we ask what to do with the * data ("field_storage"). This could be: * FORM_FIELD_STORAGE_SKIP (0) ... ignore the value of this field * FORM_FIELD_STORAGE_GET (1) ... read the data and call the get * callback function * FORM_FIELD_STORAGE_STORE (2) ... store the data in a file * FORM_FIELD_STORAGE_READ (3) ... let the user read the data * (for parsing long data on the fly) * (currently not implemented) * FORM_FIELD_STORAGE_ABORT (flag) ... stop parsing
1906. Split data in a=1&b=xy&c=3&c=4 ...
1907. The form data is in the request body data, encoded in key/value * pairs.
1908. avoid warning
1909. Must not be smaller than ~900 - see sanity check
1910. read error
1911. **comment:** The idea of "field_storage=read" is to let the API user read * data chunk by chunk and to some data processing on the fly. * This should avoid the need to store data in the server: * It should neither be stored in memory, like * "field_storage=get" does, nor in a file like * "field_storage=store". * However, for a "GET" request this does not make any much * sense, since the data is already stored in memory, as it is * part of the query string.
label: code-design
1912. Loop to read values larger than sizeof(buf)-keylen-2
1913. Log error message and skip this field.
1914. Remove from the buffer
1915. The entire data has already been loaded, so there is no need to * call mg_read. We just need to split the query string into key-value * pairs.
1916. init here, to a avoid a false positive "uninitialized variable used" warning
1917. Equivalent to "upload" callback of "mg_upload".
1918. Every part must end with \r\n, if there is another part. * The end of the request has an extra --
1919. End of the request
1920. According to the RFC, every part has to have a header field like: * Content-Disposition: form-data; name="..."
1921. It could be somethingname= instead of name=
1922. Stop parsing the request
1923. Proceed to next entry
1924. Skip all spaces between MULTIPART/FORM-DATA; and BOUNDARY=
1925. Find boundary
1926. Content-Type: application/octet-stream
1927. Get the optional filename="..." part of the Content-Disposition * header.
1928. **comment:** This line is not required, but otherwise some compilers * generate spurious warnings.
label: code-design
1929. Set "towrite" to the number of bytes available * in the buffer
1930. TODO: Create a function to get "all_data_read" * from the conn object. All data is read if the * Content-Length has been reached, or if chunked * encoding is used and the end marker has been * read, or if the connection has been closed.
1931. **comment:** we do not need mg_cry here, so conn is currently unused
label: code-design
1932. This callback will deliver partial contents
1933. **comment:** Do some sanity checks for boundary lengths
label: code-design
1934. It could be somethingfilename= instead of filename=
1935. stored successfully
1936. The form data is in the request body data, encoded as multipart * content (see <https://www.ietf.org/rfc/rfc1867.txt>, * <https://www.ietf.org/rfc/rfc2388.txt>).
1937. Call callback for new field
1938. From RFC 2046: * Boundary delimiters must not appear within the * encapsulated material, and must be no longer * than 70 characters, not counting the two * leading hyphens.
1939. Subtract the boundary length, to deal with * cases the boundary is only partially stored * in the buffer.
1940. If filename=" is found, search for the closing "
1941. Should we use a config file ?
1942. Initialize the dialog elements
1943. **comment:** Unused code from "string duplicate with escape"
label: code-design
1944. Errno will still hold the error from fopen.
1945. invalid number
1946. Show usage if -h or --help options are specified
1947. option set correctly
1948. Trim additional spaces between option name and value - then * (line+j) contains the option value
1949. Set by init_system_info()

1950. Get content of input line
1951. Dialog proc for input dialog
1952. Attach menu to the status bar
1953. Run this app as agent
1954. UNC path, e.g. \\server\dir
1955. initWithTitle:[NSString stringWithFormat:@"%@%s", server_name]
1956. Make extra verification for certain options
1957. mg_start copies all options to an internal buffer. * The options data field here is not required anymore.
1958. This should also be sufficient for "realpath", according to * <http://man7.org/linux/man-pages/man3/realpath.3.html>, but in * reality it does not seem to work.
1959. strip UTF-8 BOM
1960. Disable deprecation warning in VS2005
1961. Required for some functions (tray icons, ...)
1962. invalid boolean
1963. defined(_WIN32) && !defined(__SYMBIAN32__) - WINDOWS / UNIX include \ block
1964. For __MINGW32/64_MAJOR/MINOR_VERSION define
1965. Setup signal handler: quit on Ctrl-C
1966. integer number >= 0, e.g. number of threads
1967. defined(_WIN32) && !defined(__SYMBIAN32__) - WINDOWS / UNIX include \ block
1968. If option is already set and it is an absolute path, leave it as it is -- it's already absolute.
1969. for taskbar creation
1970. Remove user
1971. Edit passwords file: Remove user, if -R option is specified
1972. WINDOWS / UNIX include block
1973. Update config based on command line arguments
1974. Advance to next character
1975. No config file set. Path to exe found in arg[0]. * Use default file name next to the executable.
1976. Required for unit testing
1977. If value is the same as default, skip it
1978. Win32 has a small GUI. * Define some GUI elements and Windows message handlers.
1979. Set by init_server_name()
1980. **comment:** TODO (low, api makeover): options should be an array of key-value-pairs, * like * struct {const char * key, const char * value} options[] * but it currently is an array with * options[2*i] = key, options[2*i + 1] = value * (probably with a MG_LEGACY_INTERFACE definition)
 label: code-design
1981. backup config file
1982. Start option -I: * Show system information and exit * This is very useful for diagnosis.
1983. <inttypes.h> wants this for C++
1984. Do not open a password dialog, if the username is empty
1985. Option already set. Overwrite
1986. LPARAM pointer passed to WM_INITDIALOG
1987. any text
1988. Add status bar menu
1989. Make sure we have absolute paths for files and directories
1990. Initialize user data
1991. Add version menu item
1992. Invalid esc sequence
1993. Load config file settings first
1994. Line is only \
1995. Add configuration menu item
1996. Separator
1997. Start Civetweb
1998. Option not set yet. Add new option
1999. Create the dialog
2000. Loop over the lines in config file
2001. Ignore empty lines and comments
2002. alternative: SendMessage(hDlg, WM_COMMAND, ID_RESET_FILE, 0);
2003. Set dialog handle for the caller
2004. **comment:** Path to executable plus magic argument
 label: code-design
2005. C++ wants that for INT64_MAX
2006. Dialog proc for password dialog
2007. Get handle of input line
2008. _WIN32
2009. lines of text, separated by carriage return line feed
2010. Use same defines as in civetweb.c before including system headers.
2011. Add quit menu item
2012. Terminate the string - then the string at (line+i) contains the * option name
2013. Copyright (c) 2013-2017 the Civetweb developers * Copyright (c) 2004-2013 Sergey Lyubka * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
2014. Handle command line flags. They override config file and default settings.
2015. Create a password suggestion
2016. Absolutize the path, and set the option
2017. Set by start_civetweb()
2018. invalid ESC sequence
2019. Call Lua with additional CivetWeb specific Lua functions, if -L option * is specified
2020. Only allow one instance of this dialog to be open.
2021. list of text items, separated by ,
2022. Add delegate to process menu item actions
2023. Add connect menu item
2024. Return the WM_QUIT value.
2025. **comment:** TODO: define what to do
 label: code-design

2026. Otherwise: CivetWeb can work without a config file
2027. **comment:** TODO: This is not really user friendly
 label: code-design
2028. Skip spaces, \r and \n at the end of the line
2029. ignore result
2030. boolean value, yes or no
2031. Edit passwords file: Add user or change password, if -A option is * specified
2032. Advance mbl characters (+1 is below)
2033. unknown option
2034. try alternate config file
2035. Reload field with this ID
2036. If mg_start fails, it returns NULL
2037. Window proc for taskbar icon
2038. no break
2039. Not absolute. Use the directory where civetweb executable lives be the relative directory for everything. Extract civetweb executable directory into path.
2040. Load current string
2041. Dialog proc for settings dialog
2042. We've just sent our own quit message, with proper hwnd.
2043. Call Duktape, if -E option is specified
2044. Modify password
2045. Set by process_command_line_arguments()
2046. for tdm-gcc so we can use getconsolewindow
2047. Passed to mg_start() by start_civetweb()
2048. valid characters are 32 to 126
2049. Set option
2050. list of patterns, separated by |
2051. For PATH_MAX on linux
2052. Open the config file
2053. For fseeko(), ftello()
2054. The first command line parameter is a config file name.
2055. Main loop should exit
2056. Input dialog is not empty.
2057. If we're under MacOS and started by launchd, then the second argument is process serial number, -psn_..... In this case, don't process arguments at all.
2058. E.g. X:\dir
2059. This is an input dialog
2060. !CONFIG_FILE
2061. Store hWnd in a parameter accessible by the parent, so we can * bring this window to front if required.
2062. Find the space character between option name and value
2063. Make buffer available for input dialog
2064. In case this causes a problem, disable the warning: * #pragma GCC diagnostic ignored "-Wimplicit-function-declaration" * #pragma clang diagnostic ignored "-Wimplicit-function-declaration"
2065. Text boxes for files have "..." buttons to open file browser. These buttons have IDs that are ID_FILE_BUTTONS_DELTA higher than associated text box ID.
2066. Run the app
2067. Invalid multi byte character
2068. **comment:** TODO (low): check this option when it is set, instead of calling * verify_existence later
 label: code-design
2069. read all configurations from a config file
2070. 18 bytes
2071. No config file set. No path in arg[0] found. * Use default file name in the current path.
2072. Check whether option is already set
2073. Init dialog with active settings
2074. There is no input line in this dialog.
2075. If config file was set in command line and open failed, die.
2076. Create config file if it is not present yet
2077. Get dialog parameters
2078. Failed to open the file. Keep errno for the caller.
2079. Use 64-bit file offsets by default
2080. All dynamically created text boxes for options have IDs starting from ID_CONTROLS, incremented by one.
2081. Set dialog name
2082. Add user
2083. This option is evaluated by main.c, not civetweb.c - just skip it * and return OK
2084. 0xfffff5bb1
2085. 0xd4ef3085
2086. accumulate block
2087. 0x8771f681
2088. 0xfffffa3942
2089. Define storage for little-endian or both types of CPUs.
2090. Let [abcd k s t] denote the operation $a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s)$.
2091. 0xd9d4d039
2092. Let [abcd k s t] denote the operation $a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)$.
2093. message length in bits, lsw first
2094. Copyright (C) 1999, 2000, 2002 Aladdin Enterprises. All rights reserved. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution. L. Peter Deutsch ghost@aladdin.com
2095. 0xf7537e82
2096. (dynamic only)
2097. 32-bit word
2098. Process full blocks.
2099. 0xf4d50d87
2100. data are properly aligned, a direct assignment is possible
2101. Append a string to the message.
2102. 0xe8c7b756
2103. 0xc040b340
2104. **comment:** 1 = big-endian, -1 = little-endian, 0 = unknown
 label: code-design

2105. 0xc4ac5665
2106. 0xfd987193
2107. 0xfffff47d
2108. 0xf6bb4b60
2109. Define storage only for big-endian CPUs.
2110. 0xfe2ce6e0
2111. 0xe6db99e5
2112. \$Id: md5.h,v 1.4 2002/04/13 19:20:28 lpd Exp \$
2113. 0xfc93a039
2114. 0xa8304613
2115. Do the following 16 operations.
2116. (static only)
2117. cast through a (void *) should avoid a compiler warning, see <https://github.com/bel2125/civetweb/issues/94#issuecomment-98112861>
2118. dynamic little-endian
2119. 0xf61e2562
2120. **comment:** Independent implementation of MD5 (RFC 1321). This code implements the MD5 Algorithm defined in RFC 1321, whose text is available at <http://www.ietf.org/rfc/rfc1321.txt>. The code is derived from the text of the RFC, including the test suite (section A.5) but excluding the rest of Appendix A. It does not include any code or documentation that is identified in the RFC as being copyrighted. The original and principal author of md5.h is L. Peter Deutsch <ghost@aladdin.com>. Other authors are noted in the change history that follows (in reverse chronological order): 2002-04-13 lpd Removed support for non-ANSI compilers; removed references to Ghostscript; clarified derivation from RFC 1321; now handles byte order either statically or dynamically. 1999-11-04 lpd Edited comments slightly for automatic TOC extraction. 1999-10-18 lpd Fixed typo in header comment (ansi2knr rather than md5); added conditionalization for C++ compilation from Martin Purschke <porschke@bnl.gov>. 1999-05-03 lpd Original version.
label: documentation
2121. 0xab9423a7
2122. 0xc33707d6
2123. 0xf4292244
2124. Save the length before padding.
2125. **comment:** * This package supports both compile-time and run-time determination of CPU * byte order. If ARCH_IS_BIG_ENDIAN is defined as 0, the code will be * compiled to run only on little-endian CPUs; if ARCH_IS_BIG_ENDIAN is * defined as non-zero, the code will be compiled to run only on big-endian * CPUs; if ARCH_IS_BIG_ENDIAN is not defined, the code will be compiled to * run on either big- or little-endian CPUs, but will run slightly less * efficiently on either one than if ARCH_IS_BIG_ENDIAN is defined.
label: code-design
2126. * On big-endian machines, we must arrange the bytes in the * right order.
2127. Process a final partial block.
2128. 0xd8a1e681
2129. 0xe7d3fbc8
2130. 0x98badcfe
2131. 0xc1bdceee
2132. 0xea127fa
2133. 0xa4beea44
2134. 0x8f0ccc92
2135. 8-bit byte
2136. 0xd62f105d
2137. Update the message length.
2138. **comment:** Independent implementation of MD5 (RFC 1321). This code implements the MD5 Algorithm defined in RFC 1321, whose text is available at <http://www.ietf.org/rfc/rfc1321.txt>. The code is derived from the text of the RFC, including the test suite (section A.5) but excluding the rest of Appendix A. It does not include any code or documentation that is identified in the RFC as being copyrighted. The original and principal author of md5.c is L. Peter Deutsch <ghost@aladdin.com>. Other authors are noted in the change history that follows (in reverse chronological order): 2002-04-13 lpd Clarified derivation from RFC 1321; now handles byte order either statically or dynamically; added missing #include <string.h> in library. 2002-03-11 lpd Corrected argument list for main(), and added int return type, in test program and T value program. 2002-02-21 lpd Added missing #include <stdio.h> in test program. 2000-07-03 lpd Patched to eliminate warnings about "constant is unsigned in ANSI C, signed in traditional"; made test program self-checking. 1999-11-04 lpd Edited comments slightly for automatic TOC extraction. 1999-10-18 lpd Fixed typo in header comment (ansi2knr rather than md5). 1999-05-03 lpd Original version.
label: code-design
2139. 0xa679438e
2140. Let [abcd k s i] denote the operation $a = b + ((a + F(b,c,d) + X[k] + T[i]) \ll\ll s)$.
2141. Pad to 56 bytes mod 64.
2142. 0xd76aa478
2143. * This an amalgamation of md5.c and md5.h into a single file * with all static declaration to reduce linker conflicts * in Civetweb. ** The MD5_STATIC declaration was added to facilitate static * inclusion. * No Face Press, LLC
2144. end extern "C"
2145. digest buffer
2146. 0xf57c0faf
2147. Round 4.
2148. End of md5.inl
2149. 0xebefbfc70
2150. **comment:** * Determine dynamically whether this is a big-endian or * little-endian machine, since we can use a more efficient * algorithm on the latter.
label: code-design
2151. Round 2.
2152. Process an initial partial block.
2153. Let [abcd k s i] denote the operation $a = b + ((a + G(b,c,d) + X[k] + T[i]) \ll\ll s)$.
2154. Round 3.
2155. dynamic big-endian
2156. Append the length.
2157. 0xeb86d391
2158. 0x895cd7be
2159. 0x8b44f7af
2160. 0xfd469501
2161. \$Id: md5.c,v 1.6 2002/04/13 19:20:28 lpd Exp \$
2162. 0xbd3af235
2163. 0x85845dd1
2164. Round 1.
2165. [64]
2166. Initialize the algorithm.
2167. 0xe9b6c7aa
2168. 0xfde5380c
2169. Finish the message and return the digest.
2170. 0x8d2a4c8a
2171. Define the state of the MD5 Algorithm.

2172. * On little-endian machines, we can process properly aligned * data without copying it.
2173. 0xefcdab89
2174. md5_INCLUDED
2175. not aligned
2176. 0xa9e3e905
2177. little-endian
2178. 0xa3014314
2179. big-endian
2180. 0xfcfa3f8
2181. Then perform the following additions. (That is increment each of the four registers by the value it had before this block was started.)
2182. add function conn.read
2183. call the table "civetweb"
2184. subfolder "conn.http_headers"
2185. 1 = nargs
2186. Add "conn" object
2187. Add "civetweb" object
2188. 0 = nargs
2189. call the table "conn"
2190. probably never reached, but satisfies static code analysis
2191. add function conn.write
2192. duk_push_global_stash(duk_ctx); duk_get_prop_string(duk_ctx, -1, civetweb_conn_id); conn = (struct mg_connection *)duk_to_pointer(duk_ctx, -1);
2193. Note: This is only experimental support, so the API may still change.
2194. ignore result
2195. add string conn.r...
2196. Create Duktape interpreter state
2197. create a new table/object ("conn")
2198. For evaluation purposes, currently only "send" is supported. * All other ~50 functions will be added later.
2199. Script is called "protected" (duk_peval_file), so script errors should * never yield in a call to this function. Maybe calls prior to executing * the script could raise a fatal error.
2200. This file is part of the CivetWeb web server. * See <https://github.com/civetweb/civetweb/> * (C) 2015-2017 by the CivetWeb authors, MIT license.
2201. Alternative: redefine a new, clean API from scratch (instead of using mg), * or at least do not add problematic functions.
2202. **comment:** TODO: the mg context should be added to duktape as well
 label: code-design
2203. End of mod_duktape.inl
2204. mg.include: Include another .lp file
2205. No truncation check for ebuf
2206. Forward declarations
2207. just one text: send it to all client
2208. client id, opcode string and message text
2209. mg.set_timeout for websockets
2210. Variable found: return value to Lua
2211. Get connection info for connection idx
2212. path_type==NULL is the legacy use case with 1 argument
2213. Do not closesocket(*psock): here, close it in __gc
2214. Return the current "keep_alive" state. This may be false, even if * keep_alive(true) has been called.
2215. Assume the script does not support keep_alive. The script may change this * by calling mg.keep_alive(true).
2216. Register mg module
2217. Lua server pages store the depth of mg.include, in order * to detect recursions and prevent stack overflows.
2218. mg.write for websockets
2219. This function may be called with one parameter (boolean) to set the keep_alive state. Or without a parameter to just query the current keep_alive state.
2220. Get option according to argument
2221. **comment:** TODO (high): ip version
 label: requirement
2222. **comment:** unused
 label: code-design
2223. Syntax error or OOM. Error message is pushed on * stack.
2224. mg.get_var
2225. Stack of includes
2226. Allocate dynamically, so there is no internal limit for get_var
2227. "relative" = file name is relative to the * current document
2228. The civetweb internal random number generator will generate * a 64 bit random number.
2229. remote_ip is deprecated, use remote_addr instead
2230. mg.redirect: Redirect the request (internally).
2231. mg.base64_encode
2232. **comment:** TODO: dont use "bits" but fields with a meaning according to <http://tools.ietf.org/html/rfc6455>, section 5.2
 label: code-design
2233. Preload
2234. **comment:** not used
 label: code-design
2235. client id, opcode number and message text
2236. lua_Number may be used as 52 bit integer
2237. zero or one return value
2238. mg.write: Send data to the client
2239. **comment:** TODO (low): handle OOM
 label: code-design
2240. Error when loading the file (e.g. file not found, * out of memory, ...)
2241. **comment:** TODO: Delete lua_websock_data and remove it from the websocket list. This must only be done, when all connections are closed, and all asynchronous operations and timers are completed/expired.
 label: code-design
2242. This file is part of the CivetWeb web server. * See <https://github.com/civetweb/civetweb/>
2243. Execute a plain Lua script.
2244. Correct number of arguments
2245. mg.get_info
2246. Register default mg.onerror function
2247. mg.send_file
2248. Remove from ws connection list.
2249. **comment:** Variable not found (TODO (mid)): may be string too long
 label: code-design

2250. **comment:** TODO (low): Test if this could be used as a replacement for bit32. * Check again with Lua 5.3 later.
label: test

2251. mmap failed

2252. The script file is loaded, now call it

2253. **comment:** Lua error handler should show the status code
label: code-design

2254. **comment:** unused parameters
label: code-design

2255. File not found or not accessible

2256. mg.get_time

2257. opcode number and message text

2258. call add

2259. reg_int64: content_length

2260. Get info according to argument

2261. call mg.write()

2262. Export mg.request_info

2263. "virtual" = relative to document root.

2264. mg.set_interval for websockets

2265. Store script name and increment depth

2266. **comment:** TODO(lsm, low): leave the stack balanced
label: code-design

2267. Get context info for server context

2268. Store context in the registry

2269. no arguments

2270. mg.url_decode

2271. **comment:** TODO (low): This is an incomplete implementation of mmap for windows. * Currently it is sufficient, but there are a lot of unused parameters. * Better use a function "mg_map" which only has the required parameters, * and implement it using mmap in Linux and CreateFileMapping in Windows. * Noone should expect a full mmap for Windows here.
label: code-design

2272. Silently stop processing chunks.

2273. client id and message text

2274. UUID library and function pointer

2275. mg.keep_alive: Allow Lua pages to use the http keep-alive mechanism

2276. opcode string and message text

2277. add ws to list

2278. "absolute" = file name is relative to the * webserver working directory * or it is absolute system path.

2279. mg.get_response_code_text

2280. handle_lsp_request returned an error code, meaning an error * occurred in the included page and mg.onerror returned non-zero. * Stop processing.

2281. mg_fopen opens the file and sets the size accordingly

2282. Get context

2283. **comment:** TODO (mid): not implemented yet
label: requirement

2284. **comment:** shared_websock_list unused (see open TODO)
label: code-design

2285. mg.cry: Log an error. Default value for mg.onerror.

2286. End of mod_lua.inl

2287. Export connection specific info

2288. mg.random - might be better than math.random on some systems

2289. Lua "number" is a IEEE 754 double precision float: * https://en.wikipedia.org/wiki/Double-precision_floating-point_format * Thus, mask with 2^53-1 to get an integer with the maximum * precision available.

2290. mg.md5

2291. mg.get_mime_type

2292. mg.uuid

2293. Get context info for NULL context

2294. Success loading chunk. Call it.

2295. TODO (mid): Check if list entry and Lua state needs to be deleted * (see websocket_close).

2296. mg.url_encode

2297. mg.read: Read data from the client (e.g., from a POST request)

2298. Get LSP include history table

2299. Error when executing the script

2300. **comment:** <?= ?> means a variable is enclosed and its value should be * printed
label: code-design

2301. Lua state is ready to use

2302. Syntax error

2303. init ws list element

2304. Map file in memory (size is known).

2305. Script executed

2306. Get system info

2307. We're not sending HTTP headers here, Lua page must do it.

2308. name only used for debugging

2309. inc ref count

2310. Methods and meta-methods supported by the object returned by connect. * For meta-methods, see <http://lua-users.org/wiki/MetatableEvents>

2311. **comment:** Keep the "connect" method for compatibility, * but do not backport it to Lua 5.1. * TODO: Redesign the interface.
label: code-design

2312. reg_conn_function(L, "send_file", lsp_send_file, conn);

2313. If the first argument is a number, convert it to the corresponding text.

2314. Lua uses 1 based index, C uses 0 based index

2315. errors as strint return value

2316. check if ws already in list

2317. mg.get_cookie

2318. lock list (mg_context global)

2319. lua_close(L); must be done somewhere else

2320. mg.get_option

2321. Add padding and return the message digest.

2322. SHA-1 in C By Steve Reid <sreid@sea-to-sky.net> 100% Public Domain ----- Modified 7/98 By James H. Brown <jbrown@burgoyne.com> Still 100% Public Domain Corrected a problem which generated improper hash values on 16 bit machines Routine SHA1Update changed from void SHA1Update(SHA_CTX* context, unsigned char* data, unsigned int len) to void SHA1Update(SHA_CTX* context, unsigned char* data, unsigned long len) The 'len' parameter was declared an int which works fine on 32 bit machines. However, on 16 bit machines an int is too small for the shifts being done against it. This

caused the hash function to generate incorrect values if len was greater than 8191 (8K - 1) due to the 'len << 3' on line 3 of SHA1Update(). Since the file IO in main() reads 16K at a time, any file 8K or larger would be guaranteed to generate the wrong hash (e.g. Test Vector #3, a million "a"s). I also changed the declaration of variables i & j in SHA1Update to unsigned long from unsigned int for the same reason. These changes should make no difference to any 32 bit implementations since an int and a long are the same size in those environments. -- I also corrected a few compiler warnings generated by Borland C. 1. Added #include <process.h> for exit() prototype 2. Removed unused variable 'j' in SHA1Final 3. Changed exit(0) to return(0) at end of main. ALL changes I made can be located by searching for comments containing 'JHB' ----- Modified 8/98 By Steve Reid <sreid@sea-to-sky.net> Still 100% public domain 1- Removed #include <process.h> and used return() instead of exit() 2- Fixed overwriting of finalcount in SHA1Final() (discovered by Chris Hall) 3- Changed email address from steve@edmweb.com to sreid@sea-to-sky.net ----- Modified 4/01 By Saul Kravitz <Saul.Kravitz@celera.com> Still 100% PD Modified to run on Compaq Alpha hardware. ----- Modified 07/2002 By Ralph Giles <giles@ghostscript.com> Still 100% public domain modified for use with stdint types, autoconf code cleanup, removed attribution comments switched SHA1Final() argument order for consistency use SHA1_ prefix for public api move public api to sha1.h

2323. little endian / intel byte order

2324. I got the idea of expanding during the round function from SSLeay

2325. Endian independent

2326. **comment:** 11/2016 adapted for CivetWeb: include sha1.h in sha1.c, rename to sha1.inl remove unused #ifdef sections make endian independent align buffer to 4 bytes remove unused variable assignments

label: code-design

2327. 4 rounds of 20 operations each. Loop unrolled.

2328. Should cause a SHA1_Transform()

2329. blk0() and blk() perform the initial expand.

2330. Wipe variables

2331. SHA1 initialization constants

2332. Copy context->state[] to working vars

2333. Test Vectors (from FIPS PUB 180-1) "abc" A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

"abcdcbcdecdefdefgefghfghighijhijkjkljklmklmnlmmnopnopq" 84983E44 1C3BD26E BAAE4AA1 F95129E5 E54670F1 A million repetitions of "a"
34AA973C D4C4DAA4 F61EEB2B DBAD2731 6534016F

2334. Hash a single 512-bit block. This is the core of the algorithm.

2335. SHA1Init - Initialize new context

2336. (R0+R1), R2, R3, R4 are the different operations used in SHA1

2337. Must use an aligned, read/write buffer

2338. Add the working vars back into context.state[]

2339. End of sha1.inl

2340. / capacity of current token

2341. / stores number of special character codes

2342. Tokenizer_print(tok);

2343. set metatable:

2344. set __tostring metamethod

2345. / stores next token, if already determined

2346. / pointer to current token

2347. * LuaXML License LuaXml is licensed under the terms of the MIT license reproduced below, the same as Lua itself. This means that LuaXml is free software and can be used for both academic and commercial purposes at absolutely no cost. Copyright (C) 2007-2013 Gerald Franz, eludi.net Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2348. / stores current read position

2349. / stores size of string to be tokenized

2350. read elements

2351. register default codes:

2352. / stores code table for special characters

2353. / stores current read context

2354. / stores string to be tokenized

2355. parse tag header

2356. new tag found

2357. / size of current token

2358. --- internal tokenizer -----

2359. void Tokenizer_print(Tokenizer* tok) { printf(" @%u %s\n", tok->i, !tok->m_token ? "(null)" : (tok->m_token[0]==ESC)?"(esc)":(tok->m_token[0]==OPN)?"(open)":(tok->m_token[0]==CLS)?"(close)": tok->m_token); fflush(stdout); }

2360. / stores currently allocated capacity for special character codes

2361. regular attribute

2362. this tag has no content, only attributes

2363. trim whitespace

2364. strip comments

2365. --- local variables -----

2366. / size of next token

2367. interpet CDATA

2368. parse tag and content: use index 0 for storing the tag

2369. --- auxiliy functions -----

2370. strip meta information

2371. --- public methods -----

2372. previous tag is over pop current table

2373. Lua 5.1 detected

2374. not supported

2375. This header is intended to support Lua 5.1, Lua 5.2 and Lua 5.3 in the same * C source code.

2376. Lua 5.2 detected

2377. Must use luaL_newstate() for 64 bit target

2378. Copyright (c) 2015-2017 The Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF

CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

2379. #fnfdef CIVETWEB LUA_H

2380. Lua 5.3 detected

2381. Shared helper to generate DUK_OPT_xxx and DUK_USE_xxx documentation. XXX: unfinished placeholder

2382. Add a header snippet for detecting presence of DUK_OPT_xxx feature options which will be removed in Duktape 2.x.

2383. Add a header snippet for providing a __OVERRIDE_DEFINES__ section.

2384. XXX: unsigned constants?

2385. Globals holding scanned metadata, helper snippets, etc

2386. verbatim text for the entire line

2387. **comment:** Derived defines (DUK_USE_INTEGER_LE, etc) from DUK_USE_BYTEORDER. Duktape internals currently rely on the derived defines. This is after sanity checks because the derived defines are marked removed.

label: code-design

2388. Add original snippets. This fills in the required nodes recursively.

2389. **comment:** DUK_OPT_xxx always come from outside

label: code-design

2390. print('#snippet ' + sub_fn)

2391. already present

2392. **comment:** XXX: more checks

label: code-design

2393. Assume these provides come from outside.

2394. Feature selection, system include, Date provider Most #include statements are here

2395. Remaining tags in alphabetic order

2396. Automatic DUK_OPT_xxx feature option handling

2397. key is known

2398. Byte order and alignment defines are allowed to be missing, a fill-in will handle them. This is necessary because for some architecture byte order and/or alignment may vary between targets and may be software configurable.

2399. integer value

2400. Figure out fill-ins by looking for snippets not in original list and without any unserialized dependent nodes.

2401. **comment:** Autogeneration of option documentation

label: documentation

2402. duk_config.h generation

2403. print(repr(graph)) print(repr(snlist)) print('Resolved helper defines: %r' % resolved)

2404. Detect and reject 'fast math'

2405. **comment:** XXX: conditional warning, happens in some normal cases print('WARNING: define %r required, not provided so far' % k)

label: code-design

2406. Helper for building a text file from individual lines, injected files, etc. Inserted values are converted to Snippets so that their provides/requires information can be tracked. When non-C outputs are created, these will be bogus but ignored.

2407. Don't allow e.g. DUK_USE_ which results from matching DUK_USE_xxx

2408. NOTE: careful with Python equality, e.g. "0 == False" is true.

2409. Metadata to scan from config files.

2410. Generate a duk_config.h where platform, architecture, and compiler are all either autodetected or specified by user. Autodetection is based on a configured list of supported platforms, architectures, and compilers. For example, platforms.yaml defines the supported platforms and provides a helper define (DUK_F_xxx) to use for detecting that platform, and names the header snippet to provide the platform-specific definitions. Necessary dependencies (DUK_F_xxx) are automatically pulled in. Automatic "fill ins" are used for mandatory platform, architecture, and compiler defines which have a reasonable portable default. This reduces e.g. compiler-specific define count because there are a lot compiler macros which have a good default.

2411. **comment:** XXX: require automatic detection to be signaled? e.g. define DUK_USE_ALIGN_BY -1 define DUK_USE_BYTE_ORDER -1

label: code-design

2412. Development time helper: add DUK_ACTIVE which provides a runtime C string indicating what DUK_USE_xxx config options are active at run time. This is useful in genconfig development so that one can e.g. diff the active run time options of two headers. This is intended just for genconfig development and is not available in normal headers.

2413. Default value is false, and caller has emitted an unconditional #undef, so don't emit a duplicate

2414. Platform, architecture, compiler fillins. These are after all detection so that e.g. DUK_SPRINTF() can be provided by platform or compiler before trying a fill-in.

2415. Main

2416. **comment:** XXX: indicate feature option support, sanity checks enabled, etc in general summary of options, perhaps genconfig command line?

label: code-design

2417. sort under primary tag

2418. **comment:** for line in stripped_lines: print(line)

label: code-design

2419. **comment:** XXX: better to lookup compilers metadata

label: code-design

2420. Compiler files must provide at least these (additional checks in validate_compiler_file()). Fill-ins provide missing optionals.

2421. Header snippet representation: lines, provides defines, requires defines.

2422. **comment:** Find a header snippet which provides the missing define. Some DUK_F_xxx files provide multiple defines, so we don't necessarily know the snippet filename here.

label: code-design

2423. signal to fillin

2424. print('Tags: %r' % use_tags_list)

2425. must be #define'd

2426. Compatibility with Duktape 1.3

2427. compiling Duktape from a single source file (duktape.c) version compiling Duktape (not user application) artifact, include guard

2428. Forced options from multiple sources are gathered into a shared list so that the override order remains the same as on the command line.

2429. Avoid ambiguous hex escapes

2430. Snippet provides its own require; omit

2431. print('config option %s not covered by manual snippets, emitting default automatically' % k)

2432. **comment:** XXX: rst or other format

label: code-design

2433. C string value

2434. Platform files must provide at least these (additional checks in validate_platform_file()). Fill-ins provide missing optionals.

2435. shallow copy

2436. Architecture files must provide at least these (additional checks in validate_architecture_file()). Fill-ins provide missing optionals.

2437. Generate a duk_config.h header with platform, compiler, and architecture either autodetected (default) or specified by user. Support for autogenerated DUK_OPT_xxx flags is also selected by user.

2438. **comment:** XXX: aware of target version

label: code-design

2439. Preferred tags first

2440. snippet list

2441. **comment:** XXX: better to lookup architectures metadata

label: code-design

2442. graph[A] = [B, ...] <-> B, ... provide something A requires.

2443. x is a Snippet

2444. Helper headers snippets.

2445. lines of text and/or snippets map from define to 'True' for now map from define to 'True' for now

2446. **comment:** Insert missing define dependencies into index 'idx_deps' repeatedly until no unsatisfied dependencies exist. This is used to pull in the required DUK_F_xxx helper defines without pulling them all in. The resolution mechanism also ensures dependencies are pulled in the correct order, i.e. DUK_F_xxx helpers may depend on each other (as long as there are no circular dependencies). XXX: this can be simplified a lot

label: code-design

2447. strip last newline to avoid empty line

2448. **comment:** DUK_USE_xxx are internal and they should not be 'requirements'

label: code-design

2449. This is a fallback in case config option metadata is wrong.

2450. byteorder provided by all architecture files alignment provided by all architecture files packed tval provided by all architecture files

2451. print('Autoscanning snippet: %s' % fn)

2452. for printing only

2453. **comment:** XXX: detect and handle loops cleanly

label: code-design

2454. Emit a default #define / #undef for an option based on a config option metadata node (parsed YAML doc).

2455. Object layout

2456. Careful with order, snippet may self-reference its own defines in which case there's no outward dependency. (This is not 100% because the order of require/provide matters and this is not handled now.) Also, some snippets may #undef/#define another define but they don't "provide" the define as such. Such redefinitions are marked /* redefine */ in the snippets. They're best avoided (and not currently needed in Duktape 1.4.0).

2457. **comment:** Add a header snippet for checking consistency of DUK_USE_xxx config options, e.g. inconsistent options, invalid option values.

label: code-design

2458. Helpers for duk_config.h generation

2459. must be #define'd may be #undef'd, as long as provided may be #undef'd, as long as provided may be #undef'd, as long as provided

2460. Snippet may have further unresolved provides; add recursively

2461. For some options like DUK_OPT_PACKED_TVAL the default comes from platform definition.

2462. **comment:** XXX: assume no newlines etc

label: code-design

2463. Date provider snippet is after custom header and overrides, so that the user may define e.g. DUK_USE_DATE_NOW_GETTIMEofday in their custom header.

2464. **comment:** /usr/bin/env python2 Process Duktape option metadata and produce various useful outputs: - duk_config.h with specific or autodetected platform, compiler, and architecture; forced options; sanity checks; etc - option documentation for Duktape 1.x feature options (DUK_OPT_xxx) - option documentation for Duktape 1.x/2.x config options (DUK_USE_xxx) Genconfig tries to build all outputs based on modular metadata, so that managing a large number of config options (which is hard to avoid given the wide range of targets Duktape supports) remains maintainable. Genconfig does *not* try to support all exotic platforms out there. Instead, the goal is to allow the metadata to be extended, or to provide a reasonable starting point for manual duk_config.h tweaking. For Duktape 1.3 release the main goal was to autogenerate a Duktape 1.2 compatible "autodetect" header from legacy snippets, with other outputs being experimental. For Duktape 1.4 duk_config.h is always created from modular sources.

label: code-design

2465. **comment:** Not exact but close enough. Doesn't handle string literals etc, but these are not a concrete issue for scanning preprocessor #define references. Comment contents are stripped of any DUK_ prefixed text to avoid incorrect requires/provides detection. Other comment text is kept; in particular a /* redefine */ comment must remain intact here. (The 'redefine' hack is not actively needed now.) Avoid Python 2.6 vs. Python 2.7 argument differences.

label: code-design

2466. If manually-edited snippets don't #define or #undef a certain config option, emit a default value here. This is useful to fill-in for new config options not covered by manual snippets (which is intentional).

2467. position where to emit DUK_F_xxx dependencies

2468. **comment:** XXX: platform/compiler could provide types; if so, need some signaling defines like DUK_F_TYPEDEFS_DEFINED

label: code-design

2469. **comment:** Compilers need a lot of defines; missing defines are automatically filled in with defaults (which are mostly compiler independent), so the requires define list is not very large.

label: code-design

2470. **comment:** C preprocessor '#warning' is often supported

label: code-design

2471. print('Resolving %r' % k)

2472. Miscellaneous helpers

2473. If a related feature option exists, it can be used to force enable/disable the target feature. If neither feature option (DUK_OPT_xxx or DUK_OPT_NO_xxx) is given, revert to default.

2474. print('Deleting temporary directory: %r' % dirname)

2475. at least one other node provides 'k'

2476. Preferred tag order for generated C header files.

2477. **comment:** XXX: better plumbing for lookup path

label: code-design

2478. Add automatic DUK_OPT_XXX and DUK_OPT_NO_XXX handling for backwards compatibility with Duktape 1.2 and before.

2479. DUK_SETJMP, DUK_LONGJMP, DUK JMPBUF_TYPE are optional, fill-in provides if none defined.

2480. for trailing newline

2481. Forced options, last occurrence wins (allows a base config file to be overridden by a more specific one).

2482. DUK_F_xxx snippets

2483. emit tag heading only if there are subsections

2484. **comment:** XXX: placeholder, need to decide on markup conventions for YAML files

label: code-design

2485. Number types

2486. print('REQUIRES: %r' % m)

2487. **comment:** XXX: better to lookup platforms metadata

label: code-design

2488. verbatim value

2489. **comment:** uppercase only, don't match DUK_USE_xxx for example

label: code-design

2490. **comment:** XXX: assume no newlines etc XXX: support compiler specific warning mechanisms

label: code-design

2491. print('Effective tag order: %r' % tags)

2492. **comment:** Don't allow e.g. DUK_USE_ which results from matching DUK_USE_xxx print('PROVIDES: %r' % m.group(1))

label: code-design

2493. DUK_F_UCLIBC is special because __UCLIBC__ is provided by an #include file, so the check must happen after platform includes. It'd be nice for this to be automatic (e.g. DUK_F_UCLIBC.h.in could indicate the dependency somehow).

2494. Emit forced options. If a corresponding option is already defined by a snippet above, #undef it first.

2495. DLL build affects visibility attributes on Windows but unfortunately cannot be detected automatically from preprocessor defines or such. DLL build status is hidden behind DUK_F_DLL_BUILD and there are two ways for that to be set: - Duktape 1.3 backwards compatible DUK_OPT_DLL_BUILD - Genconfig --dll

option

2496. Preferred tag order for option documentation.

2497. XXX: check that YAML parses

2498. Pretty print a debugger command.

2499. Duktape internal class numbers, must match C headers

2500. Write a string into a buffer interpreting codepoints U+0000...U+00FF * as bytes. Drop higher bits.

2501. **comment:** XXX: add constants inline to preformatted output (e.g. for strings, add a short escaped snippet as a comment on the line after the compact argument list).

label: code-design

2502. absFn -> true

2503. Errors

2504. true

2505. We shouldn't come here, but if we do, JSON is a reasonable default.

2506. Could emit a 'debug-value' event here, but that's not necessary because the receiver will just collect statistics which can also be done using the finished message.

2507. **comment:** XXX: move this to the web UI so that the UI can control what locals are listed (or perhaps show locals for all levels with an expandable tree view).

label: code-design

2508. There is no close() or destroy() for a passthrough stream, so just close the outputParser which will cancel timers etc.

2509. Raw bytes Unicode string

2510. Still waiting for version identification to complete.

2511. **comment:** Pending flags are used to avoid requesting the same thing twice while a previous request is pending. The flag-based approach is quite awkward. Rework to use promises.

label: code-design

2512. 2-byte string

2513. undefined

2514. * JSON debug proxy

2515. msg.varname is a proper Unicode strings here, and needs to be converted into a protocol string (U+0000...U+00FF).

2516. new object, old may be in circulation for a while

2517. list of command names, merged client/target

2518. XXX: normalize last newline (i.e. force a newline if contents don't end with a newline)?

2519. accumulate data accumulated message until EOM

2520. A PutVar call quite possibly changes the local variables so always re-read locals afterwards. We don't need to wait for PutVar to complete here; the requests will pipeline automatically and be executed in order.

2521. 2-byte buffer

2522. ./foo/bar.js' -> 'foo/bar.js'

2523. whole

2524. Commands initiated by Duktape

2525. Not all streams will emit this.

2526. Pretty print a dvalue. Useful for dumping etc.

2527. Could also check for an empty request queue, but that's probably too strict?

2528. Technically we should wait for each delbreak reply but because target processes the requests in order, it doesn't matter.

2529. Protocol version handling. When dumping an output stream, the caller gives a non-null protocolVersion so we don't read one here.

2530. **comment:** Check if 'msg' would encode to the same JSON which was previously sent to the web client. The caller then avoid resending unnecessary stuff.

label: code-design

2531. Up-to-date list of breakpoints on target

2532. **comment:** XXX: do pretty printing in debug client for now

label: code-design

2533. console.log(JSON.stringify(msg));

2534. Duktape heapHdr type constants, must match C headers

2535. * Debugger implementation * * A debugger instance communicates with the debug target and maintains * persistent debug state so that the current state can be resent to the * socket.io client (web UI) if it reconnects. Whenever the debugger state * is changed an event is generated. The socket.io handler will listen to * state change events and push the necessary updates to the web UI, often * in a rate limited fashion or using a client pull to ensure the web UI * is not overloaded. * * The debugger instance assumes that if the debug protocol connection is * re-established, it is always to the same target. There is no separate * abstraction for a debugger session.

2536. error

2537. Decode and normalize source file contents: UTF-8, tabs to 8, * CR LF to LF.

2538. **comment:** unused/none

label: code-design

2539. * Command line parsing and initialization

2540. "notify" can be a string or "true"

2541. Shouldn't come here.

2542. 4-byte signed integer

2543. For the unquoted version we don't need to escape single or double quotes.

2544. unsigned

2545. msg.input is a proper Unicode strings here, and needs to be converted into a protocol string (U+0000...U+00FF).

2546. nop: no callback

2547. https://github.com/ryanmcgrath/wrench.js

2548. varvalue is JSON parsed by the web UI for now, need special string encoding here.

2549. Command line options (defaults here, overwritten if necessary)

2550. truncate higher bits

2551. No actual requests sent by the target right now (just notifies).

2552. Encode an ordinary Unicode string into a dvalue compatible format, i.e. * into a byte array represented as codepoints U+0000...U+00FF. Concretely, * encode with UTF-8 and then represent the bytes with U+0000...U+00FF.

2553. console.log('Received json proxy input line: ' + line.toString('utf8'));

2554. * Miscellaneous helpers

2555. indicate 'v' is actually set

2556. Dump effective options. Also provides a list of option names.

2557. Although the underlying transport may not have a close() or destroy() method or even a 'close' event, this method is always available and will generate a 'transport-close'. The caller is responsible for closing the underlying stream if that is necessary.

2558. A byline-parser is simple and good enough for now (assume compact JSON with no newlines).

2559. override

2560. update run state now that we're paused

2561. duk_tval number, any IEEE double decode into hex big endian ieee double

2562. just trigger a sync, gets rate limited

2563. **comment:** Convert a buffer into a string using Unicode codepoints U+0000...U+00FF. * This is the NodeJS 'binary' encoding, but since it's being deprecated, * reimplement it here. We need to avoid parsing strings as e.g. UTF-8: * although Duktape strings are usually UTF-8/CESU-8 that's not always the * case, e.g. for internal strings. Buffer values are also represented as * strings in the debug protocol, so we must deal accurately with arbitrary * byte arrays.

label: code-design

2564. Marker objects for special protocol values

2565. Fetch basic info right away

2566. **comment:** Pretty print a dvalue for UI usage. Everything comes out as a ready-to-use * string. * * XXX: Currently the debug client formats all values for UI use. A better * solution would be to pass values in typed form and let the UI format them, * so that styling etc. could take typing into account.

label: code-design

2567. An eval call quite possibly changes the local variables so always re-read locals afterwards. We don't need to wait for Eval to complete here; the requests will pipeline automatically and be executed in order.

2568. Commands initiated by the debug client (= us)

2569. res.status(200).json(val);

2570. raw identification string

2571. Create debugger and web UI singletons, tie them together and start them.

2572. * Debug protocol parser * * The debug protocol parser is an EventEmitter which parses debug messages * from an input stream and emits 'debug-message' events for completed * messages ending in an EOM. The parser also provides debug dumping, stream * logging functionality, and statistics gathering functionality. * * This parser is used to parse both incoming and outgoing messages. For * outgoing messages the only function is to validate and debug dump the * messages we're about to send. The downside of dumping at this low level * is that we can't match request and reply/error messages here. * * <http://www.sitepoint.com/nodejs-events-and-eventemitter/>

2573. Pretty print a dvalue string (bytes represented as U+0000...U+00FF) * for UI usage without quotes.

2574. Loose matching is tempting but counterproductive: filenames must match 1:1 between the debug client and the debug target for e.g. breakpoints to work as expected. Note that a breakpoint may be assigned by selecting a file from a dropdown populated by scanning the filesystem for available sources and there's no way of knowing if the debug target uses the exact same name.

2575. web UI singleton transport connection to target dummy passthrough for message dumping parser for incoming debug messages parser for outgoing debug messages (stats, dumping)

2576. **comment:** XXX: make the client do this?

label: code-design

2577. // This fails with "RangeError: Maximum call stack size exceeded" for some // reason, so use a much slower variant. for (i = 0, n = buf.length; i < n; i++) { cp[i] = buf[i]; } return String.fromCharCode.apply(String, cp);

2578. We require a destroy() method from the actual target stream

2579. Debug protocol integer

2580. shift forces unsigned

2581. Send a status request to trigger a status notify, result is ignored: target sends a status notify instead of a meaningful reply

2582. See doc/debugger.rst for format description.

2583. Pretty print a dvalue string (bytes represented as U+0000...U+00FF) * for UI usage. Try UTF-8 decoding to get a nice Unicode string (JSON * encoded) but if that fails, ensure that bytes are encoded transparently. * The result is a quoted string with a special quote marker for a "raw" * string when UTF-8 decoding fails. Very long strings are optionally * clipped.

2584. console.dir(argv);

2585. Intentional invalid command

2586. When we fall back to representing bytes, indicate that the string is "raw" with a 'r"' prefix (a somewhat arbitrary convention). U+0022 = ", U+0027 = '

2587. number (IEEE double), big endian

2588. We want the fileMap to contain the filename relative to the search dir root.

2589. Initial debugger connection

2590. Represented signed 32-bit integers as plain integers. Debugger code expects this for all fields that are not duk_tval representations (e.g. command numbers and such).

2591. Right now bytecode is a string containing a direct dump of the bytecode in target endianness. Decode here so that the web UI doesn't need to.

2592. **comment:** The value key should not be used programmatically, it is just there to make the dumps more readable.

label: code-design

2593. msg.varname and msg.varvalue are proper Unicode strings here, they need to be converted into protocol strings (U+0000...U+00FF).

2594. Note: typeof null === 'object', so null special case explicitly

2595. Parse complete dvalues (quite inefficient now) by trial parsing. Consume a value only when it's fully present in 'buf'. See doc/debugger.rst for format description.

2596. 0x80...0xbf: integers 0-63

2597. pointer

2598. * Source file manager * * Scan the list of search directories for Ecmascript source files and * build an index of them. Provides a mechanism to find a source file * based on a raw 'fileName' property provided by the debug target, and * to provide a file list for the web UI. * * NOTE: it's tempting to do loose matching for filenames, but this does * not work in practice. Filenames must match 1:1 with the debug target * so that e.g. breakpoints assigned based on filenames found from the * search paths will match 1:1 on the debug target. If this is not the * case, breakpoints won't work as expected.

2599. not cached, send (cache already updated)

2600. CUSTOMTRANSPORT: to use a custom transport, change this.targetStream to use your custom transport.

2601. Resend all debugger state for new client clear client state cache

2602. Parse arguments.

2603. not negative zero

2604. stats and dumping

2605. map from (merged) command name to number

2606. stats for current debug connection

2607. "request" can be a string or "true"

2608. Use a PassThrough stream to debug dump and get stats for output messages. Simply write outgoing data to both the targetStream and this passthrough separately.

2609. object

2610. * Debugger output formatting

2611. Read final, effective breakpoints from the target

2612. cached

2613. null

2614. **comment:** XXX: unprintable characters etc? In some UI cases we'd want to e.g. escape newlines and in others not.

label: code-design

2615. 4-byte string

2616. String map for commands (debug dumping). A single map works (instead of separate maps for each direction) because command numbers don't currently overlap. So merge the YAML metadata.

2617. nop

2618. lightfunc

2619. false

2620. stream is closed/broken, don't parse anymore

2621. trailing "" intentionally missing

2622. heappt

2623. Explicit rate limiter because this is a source of a lot of traffic.

2624. * Minimal debug web console for Duktape command line tool * * See debugger/README.rst. * * The web UI socket.io communication can easily become a bottleneck and * it's important to ensure that the web UI remains responsive. Basic rate * limiting mechanisms (token buckets, suppressing identical messages, etc) * are used here now. Ideally the web UI would pull data on its own terms * which would provide natural rate limiting. * * Promises are used to structure callback chains. * * <https://github.com/petkaantonov/bluebird> * <https://github.com/petkaantonov/bluebird/blob/master/API.md> *

<https://github.com/petkaantonov/bluebird/wiki/Promise-anti-patterns>

2625. Pretty print a debugger message given as an array of parsed dvalues. * Result should be a pure ASCII one-liner.

2626. Constants

2627. Not possible in practice.

2628. * Express setup and socket.io
2629. Represent non-integers as IEEE double dvalues
2630. UTF-8
2631. **comment:** console logging is done at a higher level to match request/response
 label: code-design
2632. Ignore unknown notify messages
2633. console.log(msg);
2634. 0x60...0x7f: strings with length 0-31
2635. Delete matching breakpoints in reverse order so that indices remain valid. We do this for all operations so that duplicates are eliminated if present.
2636. 0xc0...0xff: integers 0-16383
2637. close previous target connection
2638. 4-byte buffer
2639. inform web UI
2640. XXX: signal success to UI?
2641. Bytecode opcode/extraop metadata
2642. Token bucket rate limiter for a given callback. Calling code calls * trigger() to request 'cb' to be called, and the rate limiter ensures * that 'cb' is not called too often.
2643. Poll various state items when running
2644. pc is preincremented before adding
2645. cache to avoid resending identical data
2646. Here utf8.decode() is better than decoding using NodeJS buffer operations because we want strict UTF-8 interpretation.
2647. Right now the implementation is setInterval-based, but could also be made timerless. There are so few rate limited resources that this doesn't matter in practice.
2648. Value could be a Node.js buffer directly, but we prefer all dvalues to be JSON compatible
2649. this.targetStream.destroy();
2650. nop
2651. Don't add a 'value' key to numbers.
2652. preformatted dvalues
2653. used to flag special values like undefined
2654. **comment:** Pretty print a number for UI usage. Types and values should be easy to * read and typing should be obvious. For numbers, support Infinity, NaN, * and signed zeroes properly.
 label: code-design
2655. no bias in LDINTX
2656. debugger singleton current socket (or null)
2657. Try to detach cleanly, timeout if no response
2658. 4-byte signed integer
2659. heapptr
2660. plain buffer
2661. * Target binary connection handler
2662. number (IEEE double), big endian
2663. Encode arguments into dvalues.
2664. **comment:** XXX: write a notify to target?
 label: code-design
2665. disconnect JSON client too (if not already disconnected)
2666. * JSON proxy server * * Accepts an incoming JSON proxy client and connects to a debug target, * tying the two connections together. Supports both a single connection * and a persistent mode.
2667. true
2668. truncate higher bits
2669. * JSON connection handler
2670. Represented signed 32-bit integers as plain integers. Debugger code expects this for all fields that are not duk_tval representations (e.g. command numbers and such).
2671. Not possible in practice.
2672. * Main
2673. indicate 'v' is actually set
2674. **comment:** The value key should not be used programmatically, it is just there to make the dumps more readable.
 label: code-design
2675. 2-byte string
2676. Once we're connected to the target, start read both binary and JSON input. We don't want to read JSON input before this so that we can always translate incoming messages to dvalues and write them out without queueing. Any pending JSON messages will be queued by the OS instead.
2677. Generic handshake format: only relies on initial version field.
2678. undefined
2679. 0x80...0xbff: integers 0-63
2680. pointer
2681. 0x60...0x7f: strings with length 0-31
2682. Represent non-integers as IEEE double dvalues.
2683. 0xc0...0xff: integers 0-16383
2684. Trial parse dvalue(s) and debug messages.
2685. default logger log.l = 0; // enable debug and trace logging
2686. 4-byte buffer
2687. **comment:** for manual testing of binary/json parsing robustness
 label: code-design
2688. 2-byte buffer
2689. Parse message type, determine initial marker for binary message.
2690. Note that typeof null === 'object'.
2691. * Config
2692. * Misc helpers
2693. **comment:** * JSON debug proxy written in DukLuv * * This single file JSON debug proxy implementation is an alternative to the * Node.js-based proxy in duk_debug.js. DukLuv is a much smaller dependency * than Node.js so embedding DukLuv in a debug client is easier.
 label: code-design
2694. Feed the data one byte at a time when torture testing.
2695. When this is invoked the proxy connection and the target connection have both been closed.
2696. Add an EOM, and write out the dvalues to the debug target.
2697. * Detect missing 'var' declarations
2698. whole
2699. not negative zero
2700. Receive data into 'incoming', resizing as necessary.
2701. explicit flag for e.g. v === undefined
2702. disconnect target too (if not already disconnected)
2703. **comment:** XXX: it'd be nice to log remote peer host:port
 label: code-design

2704. make a copy, ensuring there's no slice offset get underlying plain buffer
2705. Trial parse JSON message(s).
2706. This is still pretty awkward in Duktape 1.4.x. Argument may be a "slice" and we want a copy of the slice (not the full underlying buffer).
2707. **comment:** XXX: Code assumes uv.write() will write fully. This is not necessarily true; should add support for partial writes (or at least failing when a partial write occurs).
label: code-design
2708. Prevent new bindings on global object. This detects missing 'var' declarations, e.g. "x = 123;" in a function without declaring it.
2709. object
2710. lightfunc
2711. skip dukluv and script name
2712. null
2713. More detailed v1 handshake line.
2714. **comment:** unnecessary but just in case
label: code-design
2715. 4-byte string
2716. **comment:** unused/none
label: code-design
2717. don't register any sockets/timers etc to exit
2718. **comment:** In lenient mode if JSON parse fails just send back an _Error and ignore the line (useful for initial development). In non-lenient mode drop the connection here; if the failed line was a request the client is expecting a reply/error message back (otherwise it may go out of sync) but we can't send a synthetic one (as we can't parse the request).
label: code-design
2719. Trial parse handshake unless done.
2720. false
2721. !/usr/bin/env python2 Merge debugger YAML metadata files and output a merged JSON metadata file.
2722. #exec-status
2723. #center-area
2724. #part-middle
2725. #part-footer
2726. ?
2727. No source loaded
2728. #about-dialog
2729. #part-header
2730. #left-area
2731. #right-area
2732. **comment:** XXX: how to minimize the chance we'll further communicate with the server or reconnect to it? socket.reconnection()
label: code-design
2733. **comment:** XXX: prevent retry of no-such-file by negative caching?
label: code-design
2734. Eval may take seconds to complete so indicate it is pending.
2735. Bytecode dialog highlight
2736. If previous update is pending, abort and start a new one.
2737. http://diveintohtml5.info/storage.html
2738. Note: loadedFileName can be either from target or from server, but they must match exactly. We could do a loose match here, but exact matches are needed for proper breakpoint handling anyway.
2739. Remove eval button "pulsating" glow when we get a result
2740. may be null
2741. Remove pending highlight once we're no longer running.
2742. Update the "console" output based on lines sent by the server. The server rate limits these updates to keep the browser load under control. Even better would be for the client to pull this (and other stuff) on its own.
2743. * Duktape debugger web client * * Talks to the NodeJS server using socket.io. * * http://unixpapa.com/js/key.html
2744. If we just became paused, check for eval watch
2745. Make copies of the requested file/line so that we have the proper values in case they've changed.
2746. * UI update handling when exec-status update arrives
2747. Active breakpoints follow
2748. Exact match is required.
2749. **comment:** XXX: any faster way; elems doesn't have e.g. indexOf()
label: code-design
2750. **comment:** XXX: ignore issue with last empty line for now
label: code-design
2751. returns a Manager
2752. Update interval for custom source highlighting.
2753. Duktape now restricts execution status updates quite effectively so there's no need to rate limit UI updates now.
2754. **comment:** XXX: hacky transition, make source change visible
label: code-design
2755. puff slide, puff
2756. onclick handler for exec status text
2757. Source is updated periodically. Other code can also call doSourceUpdate() directly if an immediate update is needed.
2758. * AJAX request handling to fetch source files
2759. **comment:** XXX: error transition here
label: requirement
2760. If we're executing the file shown, highlight current line
2761. **comment:** Scroll to requested line. This is not very clean, so a better solution should be found: https://developer.mozilla.org/en-US/docs/Web/API/Element.scrollIntoView http://erraticdev.blogspot.fi/2011/02/jquery-scroll-into-view-plugin-with.html
http://flesler.blogspot.fi/2007/10/jqueryscrollto.html
label: code-design
2762. Make source window grey when running for a longer time, use a small delay to avoid flashing grey when stepping.
2763. **comment:** Source view file that we want to be loaded in source view scroll to line once file has been loaded line that we want to highlight (if any) currently loaded (shown) file currently loaded file line count true if currFileName (loosely) matches loadedFileName if set, scroll loaded file to requested line highlight line line numbers which have been modified (added classes etc, tracked for removing) timer for updating source view current AJAX request for fetching a source file (if any) hack to reset button states bytecode dialog active index of currently highlighted line (or null) index to first line of bytecode instructions
label: code-design
2764. Remove previously added custom classes
2765. Not really interesting in the UI \$(#server-info').text(new Date() + ': ' + JSON.stringify(msg));
2766. Not 100% reliable if callstack has several functions of the same name
2767. If no-one requested us to scroll to a specific line, finish.
2768. Update buttons

2769. <http://stackoverflow.com/questions/14918787/jquery-scroll-to-bottom-of-div-even-after-it-updates> Stop queued animations so that we always scroll quickly to bottom
2770. First line is special
2771. AJAX request for the source.
2772. * Initialization
2773. About dialog, shown automatically on first startup.
2774. Enter handling for eval input <https://forum.jquery.com/topic/bind-html-input-to-enter-key-keypress>
2775. Eval watch handling
2776. We'd like to window.close() here but can't (not allowed from scripts). Alert is the next best thing.
2777. float
2778. Bytecode dialog
2779. Force source view to match currFileName only when running or when just became paused (from running or detached).
2780. nop
2781. Force line update (scrollTop) only when running or just became paused. Otherwise let user browse and scroll source files freely.
2782. Not worth alerting about because source fetch errors happen all the time, e.g. for dynamically evaluated code.
2783. Execution state previous execution state ('paused', 'running', etc) previous debugger attached state (true, false, null) current filename being executed current function name being executed current line being executed current bytecode PC being executed current execution state ('paused', 'running', 'detached', etc) current debugger attached state (true or false) current local variables current callstack (from top to bottom) current breakpoints timestamp when last started running (if running) (used to grey out the source file if running for long enough)
2784. * Source view periodic update handling
2785. If we just started running, store a timestamp so we can grey out the source view only if we execute long enough (i.e. we're not just stepping).
2786. **comment:** The variable value is parsed as JSON right now, but it'd be better to also be able to parse buffer values etc.
 label: code-design
2787. * Init socket.io and add handlers
2788. don't clear eval input
2789. * AJAX request for fetching the source list
2790. This is worth alerting about as the UI is somewhat unusable if we don't get a source list.
2791. Update current execution state
2792. Add breakpoints
2793. * Helpers
2794. Pause may take seconds to complete so indicate it is pending.
2795. This must fit into the smallest pool entry.
2796. Define to enable some debug printfs.
2797. Use 'st' as udata.
2798. **comment:** #define DUK_ALLOC_HYBRID_DEBUG
 label: code-design
2799. **comment:** * Example memory allocator with pool allocation for small sizes and * fallback into malloc/realloc/free for larger sizes or when the pools * are exhausted. * * Useful to reduce memory churn or work around a platform allocator * that doesn't handle a lot of small allocations efficiently.
 label: code-design
2800. Still fits, no shrink support.
2801. 'ptr' cannot be NULL.
2802. DUK_ALLOC_HYBRID_H_INCLUDED
2803. **comment:** The double value in the union is there to ensure alignment is * good for IEEE doubles too. In many 32-bit environments 4 bytes * would be sufficiently aligned and the double value is unnecessary.
 label: code-design
2804. Handle the ptr-NULL vs. size-zero cases explicitly to minimize * platform assumptions. You can get away with much less in specific * well-behaving environments.
2805. **comment:** * Example memory allocator with machine parseable logging. * * Also sizes for reallocs and frees are logged so that the memory * behavior can be essentially replayed to accurately determine e.g. * optimal pool sizes for a pooled allocator. * * Allocation structure: * * [alloc_hdr | user area] * * ^ ^ * | `--- pointer returned to Duktape * `--- underlying malloc ptr
 label: code-design
2806. Suppress warning.
2807. **comment:** DUK_ALLOC_LOGGING_H_INCLUDED
 label: code-design
2808. Note: ajduk doesn't log oldsize (uses -1 instead)
2809. **comment:** !/usr/bin/env python2 Analyze allocator logs and write total-bytes-in-use after every operation to stdout. The output can be gnuplotted as: \$ python log2gnuplot.py </tmp/duk-alloc-log.txt >/tmp/output.txt \$ gnuplot > plot "output.txt" with lines
 label: code-design
2810. A ptr/NULL/FAIL size R ptr/NULL size oldsize ptr/NULL/FAIL newsize
2811. **comment:** * Example torture memory allocator with memory wiping and check for * out-of-bounds writes. * * Allocation structure: * * [alloc_hdr | red zone before | user area | red zone after] * * ^ ^ * | `--- pointer returned to Duktape * `--- underlying malloc ptr
 label: code-design
2812. Handle the ptr-NULL vs. size-zero cases explicitly to minimize * platform assumptions. You can get away with much less in specific * well-behaving environments.
2813. **comment:** The double value in the union is there to ensure alignment is * good for IEEE doubles too. In many 32-bit environments 4 bytes * would be sufficiently aligned and the double value is unnecessary.
 label: code-design
2814. Suppress warning.
2815. Force address change on every realloc.
2816. DUK_ALLOC_TORTURE_H_INCLUDED
2817. * Enter interactive mode if options indicate it
2818. [...] bytecode_filename src_data src_len filename]
2819. * Main
2820. Example of sending an application specific debugger notification.
2821. 'p' points to a NUL (p == p_end) or a period.
2822. avoid SIGPIPE killing process
2823. Full completion, add a period, e.g. input 'Math' -> 'Math.'2824. Here we know the eval arg exists but check anyway
2825. DUK_CMDLINE_RLIMIT
2826. * Misc helpers
2827. A virtual '/tmp' exists by default: * <https://gist.github.com/evanw/e6be28094f34451bd5bd#file-temp-js-L3806-L3809>
2828. DUK_CMDLINE_LINENOISE_COMPLETION
2829. Callback should avoid errors for now, so use * duk_check_stack() rather than duk_require_stack().
2830. Manual test for bytecode dump/load cycle: dump and load before * execution. Enable manually, then run "make qecmatest" for a * reasonably good coverage of different functions and programs.
2831. try { FS.unmount("/"); } catch (e) { console.log("Failed to unmount default '/' MEMFS mount: " + e); }
2832. original, including partial last component
2833. Ensure socket is closed even when detach is initiated by Duktape * rather than debug client.

2834. * Minimal Linenoise completion support
2835. caller coerces
2836. [... bytecode_filename src_data src_len function]
2837. Optional bytecode dump.
2838. Print error objects with a stack trace specially. * Note that getting the stack trace may throw an error * so this also needs to be safe call wrapped.
2839. * Duktape heap lifecycle
2840. Manual test for duk_debugger_cooperate()
2841. 'idx_obj' points to the object matching the last * full component, use [p_curr,p] as a filter for * that object.
2842. Running stdin like a full file (reading all lines before * compiling) is useful with emduk: * cat test.js | ./emduk --run-stdin
2843. should never happen, but just in case
2844. Catches e.g. 'foo..bar' -> we want 'bar' only.
2845. Print error to stderr and pop error.
2846. * Signal handling setup
2847. ~2 GB
2848. **comment:** * Cleanup and exit
 label: code-design
2849. Partial ends in a period, e.g. 'Math.' -> complete all Math properties.
2850. fatal_handler
2851. **comment:** Very simplified "is identifier part" check.
 label: code-design
2852. Read until EOF, avoid fseek/stat because it won't work with stdin.
2853. global
2854. Defined in duk_cmdline_ajduk.c or alljoyn.js headers.
2855. **comment:** This is not necessary but should be harmless.
 label: code-design
2856. 'p' will either be p_start - 1 (ran out of buffer) or point to * the first offending character.
2857. nrets
2858. **comment:** XXX: Here it'd be nice to get some stats for the compilation result * when a suitable command line is given (e.g. code size, constant * count, function count. These are available internally but not through * the public API.
 label: code-design
2859. In non-interactive mode, success results are not written at all. * It is important that the result value is not string coerced, * as the string coercion may cause an error in some cases.
2860. * In interactive mode, write to stdout so output won't * interleave as easily. * * NOTE: the ToString() coercion may fail in some cases; * for instance, if you evaluate: * * ({valueOf: function() {return {}}, * toString: function() {return {}}}); * * The error is: * * TypeError: failed to coerce with [[DefaultValue]] * duk_api.c:1420 * * These are handled now by the caller which also has stack * trace printing support. User code can print out errors * safely using duk_safe_to_string().
2861. DUK_CMDLINE_FILEIO
2862. **comment:** +25% and some extra
 label: code-design
2863. * Command line execution tool. Useful for test cases and manual testing. * * To enable linenoise and other fancy stuff, compile with -DDUK_CMDLINE_FANCY.
 * It is not the default to maximize portability. You can also compile in * support for example allocators, grep for DUK_CMDLINE_*.
2864. get_value
2865. nret
2866. for properties of plain strings etc
2867. Not an Error instance, don't read "stack".
2868. 'this' binding
2869. Source code.
2870. **comment:** XXX: handle partial writes
 label: code-design
2871. DUK_CMDLINE_SIGNAL
2872. **comment:** Emscripten specific: stdin EOF doesn't work as expected. * Instead, when 'emduk' is executed using Node.js, a file * piped to stdin repeats (!). Detect that repeat and cut off * the stdin read. Ensure the loop repeats enough times to * avoid detecting spurious loops. * * This only seems to work for inputs up to 256 bytes long.
 label: code-design
2873. skip code
2874. * Simple file read/write bindings
2875. * Usage
2876. **comment:** XXX: There's no key quoting now, it would require replacing the * last component with a ['foo\nbar'] style lookup when appropriate.
 label: code-design
2877. 'p' now points to a string of the form 'foo.bar.quux'. Look up * all the components except the last; treat the last component as * a partial name which is used as a filter for the previous full * component. All lookups are from the global object now.
2878. p_curr == p_end allowed on purpose, to handle 'Math.' for example.
2879. For automatic reattach testing.
2880. Not found.
2881. Use duk_compile_lstring_filename() variant which avoids interning * the source code. This only really matters for low memory environments.
2882. heap_udata: ignored by AjsHeap, use as marker
2883. * Memory limit
2884. fall thru
2885. original, e.g. 'Math.'
2886. DUK_CMDLINE_LINENOISE
2887. * Execute from file handle etc
2888. * Create heap
2889. completion to last component
2890. 128 MB
2891. signal(SIGPIPE, SIG_IGN);
2892. EMSCRIPTEN
2893. Bytecode.
2894. in interactive mode, write to stdout
2895. Try to use NODEFS to provide access to local files. Mount the * CWD as /working, and then prepend "/working/" to relative native * paths in file calls to get something that works reasonably for * relative paths. Emscripten doesn't support replacing virtual * "/" with host "/" (the default MEMFS at "/" can't be unmounted) * but we can mount "/tmp" as host "/tmp" to allow testcase runs. * * https://kripken.github.io/emscripten-site/docs/api_reference/Filesystem-API.html#filesystem-api-nodefs * https://github.com/kripken/emscripten/blob/master/tests/fs/test_nodefs_rw.c
2896. **comment:** Suppress warnings about plain fopen() etc.
 label: code-design
2897. suppress warning
2898. try { FS.mkdir("/tmp"); } catch (e) { console.log("Failed to create virtual /tmp: " + e); }
2899. Should never happen, just in case.
2900. * Execute any argument file(s)

2901. **comment:** duk_safe_call() cleans up
label: code-design

2902. * Parse options

2903. an error 'taints' the execution

2904. At the moment it's not possible to replace the default MEMFS mounted at '/': * <https://github.com/kripken/emscripten/issues/2040> *
https://github.com/kripken/emscripten/blob/incoming/src/library_fs.js#L1341-L1358

2905. **comment:** This is useful at the global level; libraries should avoid SIGPIPE though
label: code-design

2906. nargs

2907. Helper define to enable a feature set; can also use separate defines.

2908. Last component is partial, complete.

2909. **comment:** Workaround for snprintf() missing in older MSVC versions. * Note that _snprintf() may not NUL terminate the string, but * this difference does not matter here as a NUL terminator is * always explicitly added.
label: code-design

2910. Ignore array index keys: usually not desirable, and would * also require ['0'] quoting.

2911. Scan backwards for a maximal string which looks like a property * chain (e.g. foo.bar.quux).

2912. skip filename

2913. **comment:** * Heap initialization when using AllJoyn.js pool allocator (without any * other AllJoyn.js integration). This serves as an example of how to * integrate Duktape with a pool allocator and is useful for low memory * testing. * * The pool sizes are not optimized here. The sizes are chosen so that * you can look at the high water mark (hwm) and use counts (use) and see * how much allocations are needed for each pool size. To optimize pool * sizes more accurately, you can use -alloc-logging and inspect the memory * allocation log which provides exact byte counts etc. * * <https://git.allseenalliance.org/cgit/core/alljoyn-js.git> *
<https://git.allseenalliance.org/cgit/core/alljoyn-js.git/tree/ajs.c>
label: code-design

2914. This extern declaration is provided by duktape.h, array provided by duktape.c.

2915. heapConfig

2916. Enable heap dumps

2917. duk_heap, with heap ptr compression, ROM strings+objects

2918. ...

2919. Scan ROM pointer range for faster detection of "is 'p' a ROM pointer" * later on.

2920. heapSz

2921. **comment:** not needed
label: requirement

2922. numPools

2923. **comment:** * Simplified example of an external strings strategy where incoming strings * are written sequentially into a fixed, memory mapped flash area. * * The example first scans if the string is already in the flash (which may * happen if the same string is interned multiple times), then adds it to * flash if there is space. * * This example is too slow to be used in a real world application: there * should be e.g. a hash table to quickly check for strings that are already * present in the string data (similarly to how string interning works in * Duktape itself).
label: code-design

2924. * Not present yet, check if we have space. Again, be careful to * ensure there is space for a NUL following the input data.

2925. Linear scan. An actual implementation would need some acceleration * structure, e.g. select a sublist based on first character. * * NOTE: input string (behind 'ptr' with 'len' bytes) DOES NOT have a * trailing NUL character. Any strings returned from this function * MUST have a trailing NUL character.

2926. * 'ajduk' specific functionality, examples for low memory techniques

2927. Pointer compression with ROM strings/objects: * * For now, use DUK_USE_ROM_OBJECTS to signal the need for compressed ROM * pointers. DUK_USE_ROM_PTRCOMP_FIRST is provided for the ROM pointer * compression range minimum to avoid duplication in user code.

2928. to avoid empty source file

2929. Ensure that we always get the heap_udata given in heap creation. * (Useful for Duktape development, not needed for user programs.)

2930. AjsHeap.dump(), allows Ecmascript code to dump heap status at suitable * points.

2931. heapNum

2932. protected call not yet running

2933. **comment:** The if-condition should be the fastest possible check * for "is 'p' in ROM?". If pointer is in ROM, we'd like * to compress it quickly. Here we just scan a ~1K array * which is very bad for performance and for illustration * only.
label: code-design

2934. Ensure that we always get the heap_udata given in heap creation.

2935. heap

2936. * These strings are from util/duk_meta_to_strarray.py

2937. **comment:** * Check if we already have the string. Be careful to compare for * NUL terminator too, it is NOT present in 'ptr'. This algorithm * is too simplistic and way too slow for actual use.
label: code-design

2938. duk_heap, with heap ptr compression, RAM strings+objects

2939. This is a blind lookup, could check index validity. * Duktape should never decompress a pointer which would * be out-of-bounds here.

2940. **comment:** * External strings strategy intended for valgrind testing: external strings * are allocated using malloc()/free() so that valgrind can be used to ensure * that strings are e.g. freed exactly once.
label: code-design

2941. * These strings are manually added, and would be gathered in some * application specific manner.

2942. **comment:** It's not worth it to make very small strings external, as * they would take the same space anyway. Also avoids zero * length degenerate case.
label: code-design

2943. **comment:** Somewhat portable way of losing a const without warnings. * Another approach is to cast through intptr_t, but that * type is not always available.
label: code-design

2944. * Example pointer compression functions. * * 'base' is chosen so that no non-NULL pointer results in a zero result * which is reserved for NULL pointers.

2945. * There is space, add the string to our collection, being careful * to append the NUL.

2946. DUK_CMDLINE_AJSHEAP

2947. **comment:** * Wrapped ajs_heap.c alloc functions * * Used to write an alloc log.
label: code-design

2948. until_nul

2949. We should really never be here: Duktape should only be * compressing pointers which are in the ROM compressed * pointers list, which are known at 'make dist' time. * We go on, causing a pointer compression error.

2950. * Simplified example of an external strings strategy where a set of strings * is gathered during application compile time and baked into the application * binary. * * Duktape built-in strings are available from duk_build_meta.json, see * util/duk_meta_to_strarray.py. There may also be a lot of application * specific strings, e.g. those used by application specific APIs. These * must be gathered through some other means, see e.g. util/scan_strings.py.

2951. * Execution timeout example

2952. See userdata discussion in ajsheap_enc16().

2953. duk_hthread, with heap ptr compression, ROM strings+objects

2954. **comment:** potentially unused
label: code-design

2955. **comment:** Userdata is not needed in this case but would be useful if heap * pointer compression were used for multiple heaps. The userdata * allows the callback to distinguish between heaps and their base * pointers. * * If not needed, the userdata can be left out during compilation * by simply ignoring the userdata argument of the pointer encode * and decode macros. It is kept here so that any bugs in actually * providing the value inside Duktape are revealed during compilation.

label: code-design

2956. numHeaps
2957. * Helpers
2958. seconds
2959. * Convert an 8-bit input string (e.g. ISO-8859-1) into CESU-8. * Calling code supplies the "code page" as a 256-entry array of * codepoints for the conversion. * * This is useful when input data is in non-UTF-8 format and must * be converted at runtime, e.g. when compiling non-UTF-8 source * code. Another alternative is to use e.g. iconv.
2960. Temporary buffer length wraps.
2961. max expansion is 1 input byte -> 3 output bytes
2962. Decode an 8-bit string using 'codepage' into Unicode codepoints and * re-encode into CESU-8. Codepage argument must point to a 256-entry * table. Only supports BMP (codepoints U+0000 to U+FFFF).
2963. [... tmp res]
2964. In CESU-8 all codepoints in [0x0000,0xFFFF] are * allowed, including surrogates.
2965. DUK_CODEPAGE_CONV_H_INCLUDED
2966. <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP1252.TXT>
2967. Example: compile and run test source encoded in Windows codepage 1252.
2968. undefined
2969. Exercise all 3 byte lengths: any ASCII character is 1 byte, 0xFC maps to * U+00FC which is 2 bytes, and 0x80 maps to U+20AC which is 3 bytes.
2970. * Same as above, but if the exception inherits from std::exception, it's * "what()" will be included in the error message.
2971. * If you let your own C++ exceptions propagate out of a Duktape/C function * it will be caught by Duktape and considered a programming error. Duktape * will catch the exception and convert it to a Duktape error. * * This may be allowed in a later version once all the implications have been * worked out.
2972. ERROR: exception propagated to Duktape
2973. * Example class with a destructor
2974. suppress warning
2975. * Example of how to use DUK_USE_CPP_EXCEPTIONS to support automatic * variables (e.g. destructor calls) in Duktape/C functions. * * Compile with -DDUK_OPT_CPP_EXCEPTIONS: * * \$ g++ -fPIC -o test -DDUK_OPT_CPP_EXCEPTIONS -I<duktape_dist>/src/ \ * <duktape_dist>/src/duktape.cpp_exceptions.cpp -lstdc++ * or ensure duk_config.h has DUK_USE_CPP_EXCEPTIONS enabled using * genconfig. When executed you should see something like: * * \$./test * my_class instance created * my_class instance destroyed <-- destructor gets called * --> rc=1 (SyntaxError: parse error (line 1)) * [...] * * Duktape uses a custom exception class (duk_internal_exception) which * doesn't inherit from any base class, so that catching any base classes * in user code won't accidentally catch exceptions thrown by Duktape.
2976. * You can use C++ exceptions inside Duktape/C functions for your own * purposes but you should catch them before they propagate to Duktape.
2977. * SyntaxError caused by eval exits Duktape/C function but destructors * are executed.
2978. * Same as above, but if the exception inherits from std::exception with * a NULL what(). Duktape will describe the error as 'unknown' if so.
2979. Handshake line is available for caller for the * duration of the callback, and must not be freed * by the caller.
2980. Too small, resize.
2981. just in case, if dvalues changed something
2982. mixed endian (arm)
2983. Define to enable error prints to stderr.
2984. 0x80 ... 0xbf
2985. **comment:** IEEE double byte order, detect at run time (could also use * preprocessor defines but that's verbose to make portable). * * >>> struct.unpack('>d', '1122334455667788'.decode('hex')) * (3.841412024471731e-226,) * >>> struct.unpack('>d', '8877665544332211'.decode('hex')) * (-7.086876636573014e-268,) * >>> struct.unpack('>d', '4433221188776655'.decode('hex')) * (3.5294303071877444e+20,)
label: code-design
2986. Handshake line is returned as a dvalue for convenience; it's * not actually a part of the dvalue phase of the protocol.
2987. Too big, resize so that we reclaim memory if we have just * received a large string/buffer value.
2988. little endian
2989. 0x00 ... 0x1f
2990. **comment:** * Example debug transport with a local debug message encoder/decoder. * * Provides a "received dvalue" callback for a fully parsed dvalue (user * code frees dvalue) and a "cooperate" callback for e.g. UI integration. * There are a few other callbacks. See test.c for usage examples. * * This transport implementation is not multithreaded which means that: * * - Callbacks to "received dvalue" callback come from the Duktape thread, * either during normal execution or from duk_debugger_cooperate(). * * - Calls into duk_trans_dvalue_send() must be made from the callbacks * provided (e.g. "received dvalue" or "cooperate") which use the active * Duktape thread. * * - The only exception to this is when Duktape is idle: you can then call * duk_trans_dvalue_send() from any thread (only one thread at a time). * When you next call into Duktape or call duk_debugger_cooperate(), the * queued data will be read and processed by Duktape. * * There are functions for creating and freeing values; internally they use * malloc() and free() for memory management. Duktape heap alloc functions * are not used to minimize disturbances to the Duktape heap under debugging. * * Doesn't depend on C99 types; assumes "int" is at least 32 bits, and makes * a few assumptions about format specifiers.
label: code-design
2991. never here
2992. 0x60 ... 0x7f
2993. Caller must provide a buffer at least DUK_DVALUE_TOSTRING_BUflen in size.
2994. Harmless warning on some platforms (re: range)
2995. Alloc size is len + 1 so that a NUL terminator is always * guaranteed which is convenient, e.g. you can printf() the * value safely.
2996. big endian
2997. Integers are network endian, read back into host format.
2998. Portable IEEE double byteswap. Relies on runtime detection of * host endianness.
2999. Integers are written in network endian format.
3000. IEEE doubles are network endian, read back into host format.
3001. 32 hex encoded or \xFF escaped bytes, possible "...", NUL
3002. * Dvalue transport handling
3003. Integers are network endian, read back into host format in * a portable manner.
3004. 0xc0 ... 0xff
3005. * Dvalue handling
3006. **comment:** Define to enable debug prints to stderr.
label: code-design
3007. end switch
3008. * Duktape callbacks
3009. Portable sign handling, doesn't assume 'int' is exactly 32 bits * like a direct cast would.
3010. Trial parse handshake line or dvalue(s).
3011. IEEE doubles are written in network endian format.
3012. Must cooperate until user callback provides data. From * Duktape's perspective we MUST block until data is received.
3013. mixed endian
3014. 0x20 ... 0x5f
3015. **comment:** some extra to reduce resizes
label: code-design
3016. Append data.
3017. When read_offset is large enough, "rebase" buffer by deleting already * read data and updating offsets.
3018. big endian: ok as is

3019. Convert argument dvalue into Duktape debug protocol format. * Literal constants are used here for the debug protocol, * e.g. initial byte 0x02 is REP, see doc/debugger.rst.
3020. tolerates NULL
3021. d: ieee double
3022. sending towards Duktape (duktape read callback)
3023. Buffer size needed by duk_dvalue_to_string().
3024. buf: buffer data, len: buffer length
3025. DUK_TRANS_DVALUE_H_INCLUDED
3026. struct duk_dvalue 'tag' values, note that these have nothing to do with * Duktape debug protocol initial byte. Struct fields used with the type * are noted next to the define.
3027. i: class number, buf: pointer data, len: pointer length
3028. i: 32-bit signed integer
3029. buf: pointer data, len: pointer length
3030. Sending dvalues towards Duktape.
3031. Duktape debug callbacks provided by the transport.
3032. receiving from Duktape (duktape write callback)
3033. buf: string data, len: string length
3034. Could use a union for the value but the gain would be relatively small.
3035. no fields
3036. **comment:** 0=little endian, 1=big endian, 2=mixed endian
 label: code-design
3037. Initializing and freeing the transport context.
3038. Dvalue handling.
3039. i: lightfunc flags, buf: pointer data, len: pointer length
3040. Attach debugger; this will fail with a fatal error here unless * debugger support is compiled in. To fail more gracefully, call * this under a duk_safe_call() to catch the error.
3041. DumpHeap
3042. fake ptr len
3043. The Duktape handshake line is given in 'line' (without LF). * The 'line' argument can be accessed for the duration of the * callback (read only). Don't free 'line' here, the transport * handles that.
3044. medium, >= 32 chars
3045. Evaluate simple test code, callbacks will "step over" until end. * * The test code here is just for exercising the debug transport. * The 'evalMe' variable is evaluated (using debugger command Eval) * before every step to force different dvalues to be carried over * the transport.
3046. duk_trans_dvalue_send_req_cmd() sends a REQ dvalue followed by * an integer dvalue (command) for convenience.
3047. classnum
3048. Detached call forwarded as is.
3049. 0x14 = StepOver
3050. * Example program using the dvalue debug transport.
3051. long buffer, >= 65536 chars
3052. TriggerStatus
3053. suppress warning
3054. Here a normal debug client would wait for dvalues until an EOM * dvalue was received (which completes a debug message). The * debug message would then be handled, possibly causing UI changes * and/or causing debug commands to be sent to Duktape. * * The callback is responsible for eventually freeing the dvalue. * Here we free it immediately, but an actual client would probably * gather dvalues into an array or linked list to handle when the * debug message was complete.
3055. Duktape is not blocked; you can cooperate with e.g. a user * interface here and send dvalues to Duktape, but don't block.
3056. Also send a dummy TriggerStatus request with trailing dvalues * ignored by Duktape; Duktape will parse the dvalues to be able to * skip them, so that the dvalue encoding is exercised.
3057. 0x1e = Eval
3058. **comment:** First time Duktape becomes blocked, send DumpHeap which * exercises a lot of parsing code. * * NOTE: Valgrind may complain about reading uninitialized * bytes. This is caused by the DumpHeap command writing out * verbatim duk_tval values which are intentionally not * always fully initialized for performance reasons.
 label: code-design
3059. short, <= 31 chars
3060. Duktape is blocked on a read and won't continue until debug * command(s) are sent. * * Normally you'd enter your own event loop here, and process * events until something needs to be sent to Duktape. For * example, the user might press a "Step over" button in the * UI which would cause dvalues to be sent. You can then * return from this callback. * * The code below sends some example messages for testing the * dvalue handling of the transport. * * If you create dvalues manually and send them using * duk_trans_dvalue_send(), you must free the dvalues after * the send call returns using duk_dvalue_free().
3061. lf_flags
3062. DUK_TRANS_SOCKET_H_INCLUDED
3063. **comment:** XXX: For now, close the listen socket because we won't accept new * connections anyway. A better implementation would allow multiple * debug attaches.
 label: code-design
3064. Duktape debug transport callback: (possibly partial) read.
3065. This shouldn't happen.
3066. should never happen
3067. This TCP transport requires no write flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.
3068. * Example debug transport using a Linux/Unix TCP socket * * Provides a TCP server socket which a debug client can connect to. * After that data is just passed through.
3069. This TCP transport requires no read flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.
3070. backlog
3071. In a production quality implementation there would be a sanity * timeout here to recover from "black hole" disconnects.
3072. Duktape debug transport callback: (possibly partial) write.
3073. Write flush. If the transport combines multiple writes * before actually sending, a write flush is an indication * to write out any pending bytes: Duktape may not be doing * any more writes on this occasion.
3074. * Transport init and finish
3075. * Duktape callbacks
3076. nothing to read
3077. also returns 0, which is correct
3078. Read flush: Duktape may not be making any more read calls at this * time. If the transport maintains a receive window, it can use a * read flush as a signal to update the window status to the remote * peer. A read flush is guaranteed to occur before Duktape stops * reading for a while; it may occur in other situations as well so * it's not a 100% reliable indication.
3079. **comment:** not needed by the example
 label: requirement
3080. something to read
3081. **comment:** XXX: For now, close the listen socket because we won't accept new * connections anyway. A better implementation would allow multiple * debug attaches.
 label: code-design
3082. Duktape debug transport callback: (possibly partial) read.

3083. **comment:** MinGW workaround for missing getaddrinfo() etc: * http://programmingrants.blogspot.fi/2009/09/tips-on-undefined-reference-to.html
label: code-design

3084. This shouldn't happen.

3085. This TCP transport requires no write flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.

3086. This TCP transport requires no read flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.

3087. In a production quality implementation there would be a sanity * timeout here to recover from "black hole" disconnects.

3088. Duktape debug transport callback: (possibly partial) write.

3089. Write flush. If the transport combines multiple writes * before actually sending, a write flush is an indication * to write out any pending bytes: Duktape may not be doing * any more writes on this occasion.

3090. * Transport init and finish

3091. * Example debug transport using a Windows TCP socket * * Provides a TCP server socket which a debug client can connect to. * After that data is just passed through. * * https://msdn.microsoft.com/en-us/library/windows/desktop/ms737593(v=vs.85).aspx * * Compiling 'duk' with debugger support using MSVC (Visual Studio): * * > cl /W3 /O2 /Feduk.exe * /DDUK_OPT_DEBUGGER_SUPPORT /DDUK_OPT_INTERRUPT_COUNTER * /DDUK_CMDLINE_DEBUGGER_SUPPORT * /lexamples\debug-trans-socket /Isrc * examples\cmdline\duk_cmdline.c * examples\debug-trans-socket\duk_trans_socket_windows.c * src\duktape.c * * With MinGW: * * \$ gcc -oduks.exe -Wall -O2 \ * -DDUK_OPT_DEBUGGER_SUPPORT -DDUK_OPT_INTERRUPT_COUNTER \ * -DDUK_CMDLINE_DEBUGGER_SUPPORT \ * -lexamples\debug-trans-socket -Isrc \ * examples\cmdline\duk_cmdline.c \ * examples\debug-trans-socket\duk_trans_socket_windows.c \ * src\duktape.c -lm -lws2_32

3092. * Duktape callbacks

3093. also returns 0, which is correct

3094. nothing to read

3095. never here

3096. Read flush: Duktape may not be making any more read calls at this * time. If the transport maintains a receive window, it can use a * read flush as a signal to update the window status to the remote * peer. A read flush is guaranteed to occur before Duktape stops * reading for a while; it may occur in other situations as well so * it's not a 100% reliable indication.

3097. **comment:** not needed by the example
label: requirement

3098. something to read

3099. Return a fixed time here as a dummy example.

3100. * Dummy Date provider * * There are two minimally required macros which you must provide in * duk_config.h: * * extern duk_double_t dummy_get_now(void); * * #define DUK_USE_DATE_GET_NOW(ctx) dummy_get_now() * #define DUK_USE_DATE_GET_LOCAL_TZOFFSET(d) 0 * * Note that since the providers are macros, you don't need to use * all arguments. Similarly, you can "return" fixed values as * constants. Above, local timezone offset is always zero i.e. * we're always in UTC. * * You can also provide optional macros to parse and format timestamps * in a platform specific format. If not provided, Duktape will use * ISO 8601 only (which is often good enough).

3101. * Very simple example program for evaluating expressions from * command line

3102. nargs

3103. nrets

3104. * A few basic tests

3105. XXX: or t->target + t->delay?

3106. **comment:** Active timers. Dense list, terminates to end of list or first unused timer. * The list is sorted by 'target', with lowest 'target' (earliest expiry) last * in the list. When a timer's callback is being called, the timer is moved * to 'timer_expiring' as it needs special handling should the user callback * delete that particular timer.
label: code-design

3107. ignore errors for now -> [... stash eventTimers]

3108. not found, append to list

3109. Initialize global stash 'eventTimers'.

3110. Update timer_count.

3111. Misc

3112. -> [...]

3113. -> [... stash eventTimers retval]

3114. One-shot timer (always removed) or removed by user callback.

3115. Finally, register the callback to the global stash 'eventTimers' object.

3116. * Determine poll() timeout (as close to poll() as possible as * the wait is relative).

3117. Interval timer, not removed by user callback. Queue back to * timer list and bubble to its final sorted position.

3118. deleted, perhaps by previous callback

3119. Timer is now at the last position; use swaps to "bubble" it to its * correct sorted position.

3120. * If exit has been requested, exit without running further * callbacks.

3121. **comment:** mark to-be-deleted, cleaned up by next poll
label: code-design

3122. last timer at timer_count - 1

3123. Zero last element for clarity.

3124. Socket poll state.

3125. [... stash eventTimers]

3126. error

3127. * Compact poll list by removing pollfds with fd == 0.

3128. Set global 'EventLoop'.

3129. eventTimers[timer_id] = callback

3130. **comment:** zeroize unused entries for sanity
label: code-design

3131. Last timer expires first (list is always kept sorted).

3132. -> [... stash eventTimers func]

3133. delay/interval

3134. oneshot=1 (setTimeout), repeated=0 (setInterval)

3135. The callback associated with the timer is held in the "global stash", * in <stash>.eventTimers[String(id)]. The references must be deleted * when a timer struct is deleted.

3136. Get Javascript compatible 'now' timestamp (millisecs since 1970).

3137. events == 0 means stop listening to the FD

3138. -> [timer_id stash eventTimers]

3139. * Expire timers.

3140. Shift elements downwards to keep the timer list dense * (no need if last element).

3141. * If exit requested, bail out as fast as possible.

3142. 't' expires later than 't-1', so swap them and repeat.

3143. Timer to bubble is at index i, timer to compare to is * at i-1 (both guaranteed to exist).

3144. Return timer id.

3145. Bubble last timer on timer list backwards until it has been moved to * its proper sorted position (based on 'target' time).

3146. next target time

3147. i = input index * j = output index (initially same as i)

3148. The C state is now up-to-date, but we still need to delete * the timer callback state from the global 'stash'.

3149. 'rc' fds active

3150. clamping ensures that fits
3151. * Expired timer(s) still exist?
3152. timeout
3153. * Move the timer to 'expiring' for the duration of the callback. * Mark a one-shot timer deleted, compute a new target for an interval.
3154. * Call timer callback. The callback can operate on the timer list: * add new timers, remove timers. The callback can even remove the * expired timer whose callback we're calling. However, because the * timer being expired has been moved to 'timer_expiring', we don't * need to worry about the timer's offset changing on the timer list.
3155. **comment:** this is quite excessive for embedded use, but good for testing
label: code-design
3156. if timeout, no need to check pollfd
3157. The Ecmascript poll handler is passed through EventLoop.fdPollHandler * which c_eventloop.js sets before we come here.
3158. indexes: * 0 = function (callback) * 1 = delay * 2 = boolean: oneshot
3159. -> [func delay oneshot stash eventTimers]
3160. **comment:** * Unlike insertion, deletion needs a full scan of the timer list * and an expensive remove. If no match is found, nothing is deleted. * Caller gets a boolean return code indicating match. * * When a timer is being expired and its user callback is running, * the timer has been moved to 'timer_expiring' and its deletion * needs special handling: just mark it to-be-deleted and let the * expiry code remove it.
label: code-design
3161. nargs
3162. * Poll for activity or timeout.
3163. copy only if indices have diverged
3164. timer has been requested for removal
3165. -> [global EventLoop fdPollHandler]
3166. Because a user callback can mutate the timer list (by adding or deleting * a timer), we expire one timer and then rescan from the end again. There * is a sanity limit on how many times we do this per expiry round.
3167. keep output index the same
3168. **comment:** * Check socket activity, handle all sockets. Handling is offloaded to * Ecmascript code (fd + revents). * * If FDs are removed from the poll list while we're processing callbacks, * the entries are simply marked unused (fd set to 0) without actually * removing them from the poll list. This ensures indices are not * disturbed. The poll list is compacted before next poll().
label: code-design
3169. numeric ID (returned from e.g. setTimeout); zero if unused
3170. Should never happen, so return whatever.
3171. delete eventTimers[timer_id]
3172. 't' expires earlier than (or same time as) 't-1', so we're done.
3173. indexes: * 0 = timer id
3174. **comment:** * C eventloop example. * * Timer management is similar to eventloop.js but implemented in C. * In particular, timer insertion is an O(n) operation; in a real world * eventloop based on a heap insertion would be O(log N).
label: code-design
3175. print('unexpected revents, close fd');
3176. no size control now print('READ', Duktape.enc('jx', data));
3177. * C eventloop example (c_eventloop.c). * * Ecmascript code to initialize the exposed API (setTimeout() etc) when * using the C eventloop. * * <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Timers>
3178. This simple example doesn't have support for write blocking / draining
3179. Connected
3180. print('ACCEPT:', Duktape.enc('jx', acc_res));
3181. Special case: callback arguments are provided. [arg1, arg2, ...] [global(this), arg1, arg2, ...]
3182. print('activity on fd', fd, 'revents', revents);
3183. * Timer API
3184. **comment:** * Socket handling * * Ideally this would be implemented more in C than here for more speed * and smaller footprint: C code would directly maintain the callback state * and such. * * Also for more optimal I/O, the buffer churn caused by allocating and * freeing a lot of buffer values could be eliminated by reusing buffers. * Socket reads would then go into a pre-allocated buffer, for instance.
label: code-design
3185. retval ignored
3186. print('UNKNOWN');
3187. Legacy case: callback is a string.
3188. oneshot
3189. Normal case: callback given as a function without arguments.
3190. Math.exp(0)...Math.exp(8) is an uneven distribution between 1...~2980.
3191. Here the inserts take a lot of time because the underlying timer manager * data structure has O(n) insertion performance.
3192. * Test using timers and intervals with curses.
3193. * Pure Ecmascript eventloop example. * * Timer state handling is inefficient in this trivial example. Timers are * kept in an array sorted by their expiry time which works well for expiring * timers, but has O(n) insertion performance. A better implementation would * use a heap or some other efficient structure for managing timers so that * all operations (insert, remove, get nearest timer) have good performance. * * <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Timers>
3194. **comment:** If exit requested, don't call any more callbacks. This allows a callback to do cleanups and request exit, and can be sure that no more callbacks are processed.
label: code-design
3195. Remove the timer from the active list and process it. The user callback may add new timers which is not a problem. The callback may also delete timers which is not a problem unless the timer being deleted is the timer whose callback we're running; this is why the timer is recorded in this.expiring so that clearTimeout() and clearInterval() can detect this situation.
3196. sockets fd -> callback fd -> callback fd -> callback
3197. If the timer was one-shot, it's marked 'removed'. If the user callback requested deletion for the timer, it's also marked 'removed'. If the timer is an interval (and is not marked removed), insert it back into the timer list.
3198. this.dumpState();
3199. Timer has expired and we're processing its callback. User callback has requested timer deletion. Mark removed, so that the timer is not reinserted back into the active list. This is actually a common case because an interval may very well cancel itself.
3200. Special case: callback arguments are provided. [arg1, arg2, ...] [global(this), arg1, arg2, ...]
3201. This simple example doesn't have support for write blocking / draining
3202. no such ID, ignore
3203. print('zero read for fd ' + fd + ', closing forcibly'); ignore result
3204. print('UNKNOWN');
3205. Legacy case: callback is a string.
3206. timers active timers, sorted (nearest expiry last) set to timer being expired (needs special handling in clearTimeout/clearInterval)
3207. no size control now print('READ', Duktape.enc('jx', data));
3208. print('revents ' + t.revents + ' for fd ' + fd + ', closing forcibly'); ignore result
3209. misc
3210. print('ACCEPT:', Duktape.enc('jx', acc_res));
3211. * Create poll socket list. This is a very naive approach. * On Linux, one could use e.g. epoll() and manage socket lists * incrementally.
3212. start
3213. deleteCount

3214. * Find 'i' such that we want to insert *after* timers[i] at index i+1. * If no such timer, for-loop terminates with i-1, and we insert at -1+1=0.
3215. print('exit requested, exit');
3216. Remove timer/interval with a timer ID. The timer/interval can reside either on the active list or it may be an expired timer (this.expiring) whose user callback we're running when this function gets called.
3217. Get timer with lowest expiry time. Since the active timers list is sorted, it's always the last timer.
3218. * Wait timeout for timer closest to expiry. Since the poll * timeout is relative, get this as close to poll() as possible.
3219. print(new Date(), 'poll_set OUT:', Duktape.enc('jx', poll_set));
3220. * Do the actual poll.
3221. * Timer API * * These interface with the singleton EventLoop.
3222. print('UNKNOWN POLLOUT');
3223. Timers to expire?
3224. * Exit check (may be requested by a user callback)
3225. **comment:** Timer on active list: mark removed (not really necessary, but nice for dumping), and remove from active list.
label: code-design
3226. insert after 't', to index i+1
3227. * Event loop * * Timers are sorted by 'target' property which indicates expiry time of * the timer. The timer expiring next is last in the array, so that * removals happen at the end, and inserts for timers expiring in the * near future displace as few elements in the array as possible.
3228. print(new Date(), 'poll_set IN:', Duktape.enc('jx', poll_set));
3229. * Here we must be careful with mutations: user callback may add and * delete an arbitrary number of timers. * * Current solution is simple: check whether the timer at the end of * the list has expired. If not, we're done. If it has expired, * remove it from the active list, record it in this.expiring, and call * the user callback. If user code deletes the this.expiring timer, * there is special handling which just marks the timer deleted so * it won't get inserted back into the active list. * * This process is repeated at most maxExpirys times to ensure we don't * get stuck forever; user code could in principle add more and more * already expired timers.
3230. custom call
3231. * Process expired timers.
3232. * Process all sockets so that nothing is left unhandled for the * next round.
3233. flag for removal
3234. Timer has not expired, and no other timer could have expired either because the list is sorted.
3235. Reinsert interval timer to correct sorted position. The timer must be an interval timer because one-shot timers are marked 'removed' above.
3236. Eat errors silently. When resizing curses window an EINTR happens now.
3237. Normal case: callback given as a function without arguments.
3238. Set global 'FileIo'.
3239. * File I/O binding example.
3240. **comment:** XXX: pcall the string coercion
label: code-design
3241. fall thru
3242. **comment:** This is useful at the global level; libraries should avoid SIGPIPE though
label: code-design
3243. nargs
3244. nrets
3245. [... global func]
3246. Duktape/C function, safe called
3247. nret
3248. Start a zero timer which will call _USERCODE from within * the event loop.
3249. Finally, launch eventloop. This call only returns after the * eventloop terminates.
3250. signal(SIGPIPE, SIG_IGN);
3251. Compile input and place it into global _USERCODE
3252. Print error to stderr and pop error.
3253. NO_SIGNAL
3254. **comment:** XXX: print error objects specially
label: code-design
3255. * Main for evloop command line tool. * * Runs a given script from file or stdin inside an eventloop. The * script can then access setTimeout() etc.
3256. Set global 'Ncurses'.
3257. XXX: no screen management now
3258. **comment:** * Ncurses bindings example. * * VALGRIND NOTE: when you use ncurses, there seems to be no way to get a * clean valgrind run. Even if ncurses state is properly shut down, there * will still be some residual leaks. * * Debian: install libncurses5-dev
label: code-design
3259. [retarr]
3260. -> [... retarr key]
3261. Linux 2.6.17 and upwards, requires _GNU_SOURCE etc, not added * now because we don't use it.
3262. -> [... retarr fd]
3263. enum_flags
3264. leave enum on stack
3265. * C wrapper for poll().
3266. update revents
3267. Set global 'Poll' with functions and constants.
3268. -> [... enum key val events]
3269. -> [... enum key val]
3270. -> [... enum]
3271. -> [... retarr val "revents" fds[i].revents]
3272. [... enum key]
3273. rc = ppoll(fds, n, &ts, NULL);
3274. -> [... enum key key]
3275. -> [... retarr val]
3276. uppercase
3277. ensure we get a plain buffer
3278. Handle socket data carefully: if you convert it to a string, it may not be valid UTF-8 etc. Here we operate on the data directly in the buffer.
3279. backlog
3280. Set global 'Socket'.
3281. MSG_NOSIGNAL: avoid SIGPIPE
3282. * TCP sockets binding example.
3283. fib.js
3284. Pure Ecmascript version of low level helper
3285. Check 'val' for primality
3286. prime.js
3287. Select available helper at load time
3288. Find primes below one million ending in '9999'.
3289. primecheck.c
3290. nargs

3291. ignore result
3292. process.js
3293. nargs
3294. index
3295. processlines.c
3296. pop result/error
3297. ignore result
3298. uppercase.c
3299. We're going to need 'sz' additional entries on the stack.
3300. one return value
3301. * Very simple example program
3302. pop global
3303. #args
3304. suppress warning
3305. pop eval result
3306. nargs
3307. **comment:** * Pretty print JSON from stdin into indented JX.
 label: code-design
3308. **comment:** suppress warnings
 label: code-design
3309. nrets
3310. -1 for filename
3311. **comment:** The double value in the union is there to ensure alignment is * good for IEEE doubles too. In many 32-bit environments 4 bytes * would be sufficiently aligned and the double value is unnecessary.
 label: code-design
3312. -> [... source filename]
3313. nrets
3314. * Main
3315. * Setup sandbox
3316. Handle the ptr-NULL vs. size-zero cases explicitly to minimize * platform assumptions. You can get away with much less in specific * well-behaving environments.
3317. must not return
3318. 256kB sandbox
3319. * Sandboxing example * * Uses custom memory allocation functions which keep track of total amount * of memory allocated, imposing a maximum total allocation size.
3320. -> [... filename source]
3321. * Sandbox setup and test
3322. Should be zero.
3323. Suppress warning.
3324. nargs
3325. flags
3326. Compile as program
3327. * Memory allocator which backs to standard library memory functions but * keeps a small header to track current allocation size. * * Many other sandbox allocation models are useful, e.g. preallocated pools.
3328. * Execute code from specified file
3329. * Mandelbrot example: * * \$./duk mandel.js * [...]
3330. z -> z^2 + c * -> (xx+i*yy)^2 + (x0+i*y0) * -> xx*xx+i*2*xx*yy-yy*yy + x0 + i*y0 * -> (xx*xx - yy*yy + x0) + i*(2*xx*yy + y0)
3331. xx^2 + yy^2 >= 4.0
3332. capture in closure in case changed later
3333. * Minimal console.log() polyfill
3334. nop
3335. * Ensure Error .fileName, .lineNumber, and .stack are not directly writable, * but can be written using Object.defineProperty(). This matches Duktape * 1.3.0 and prior. * * See: <https://github.com/svaarala/duktape/pull/390>.
3336. already non-writable
3337. already writable
3338. * Ensure Error .fileName, .lineNumber, and .stack are directly writable * without having to use Object.defineProperty(). This matches Duktape * 1.4.0 behavior. * * See: <https://github.com/svaarala/duktape/pull/390>.
3339. tag for unpacked duk_tval
3340. tag for packed duk_tval
3341. internal tag is fastint
3342. public type is DUK_TYPE_NUMBER
3343. * Helper to check if a number is internally represented as a fastint: * * if (Duktape.isFastint(x)) { * print('fastint'); * } else { * print('not a fastint (or not a number)'); * } * * NOTE: This helper depends on the internal tag numbering (defined in * duk_tval.h) which is both version specific and depends on whether * duk_tval is packed or not.
3344. Tag number depends on duk_tval packing.
3345. null or undefined
3346. enumerable own keys
3347. * Object.assign(), described in E6 Section 19.1.2.1 * * <http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.assign>
3348. * Object.prototype.__defineGetter__ polyfill
3349. * Object.prototype.__defineSetter__ polyfill
3350. **comment:** * Performance.now() polyfill * * <http://www.w3.org/TR/hr-time/#sec-high-resolution-time> * * Dummy implementation which uses the Date built-in and has no higher * resolution. If/when Duktape has a built-in high resolution timer * interface, reimplement this.
 label: code-design
3351. This should not really happen, but would indicate x64.
3352. **comment:** Minimize warnings for unused internal functions with GCC >= 3.1.1 and * Clang. Based on documentation it should suffice to have the attribute * in the declaration only, but in practice some warnings are generated unless * the attribute is also applied to the definition.
 label: code-design
3353. Emscripten (provided explicitly by user), improve if possible
3354. Both MinGW and MSVC have a 64-bit type.
3355. * Platform autodetection
3356. --- TinyC ---
3357. mint lib is missing these
3358. SuperH
3359. --- x64 ---
3360. MSVC does not have sys/param.h
3361. --- Motorola 68k ---
3362. since gcc-2.5
3363. --- MIPS 64-bit ---

3364. **comment:** XXX: DUK_UNREACHABLE for msvc?
 label: code-design

3365. OpenBSD
3366. NetBSD
3367. Apple OSX, iOS
3368. Cannot determine byte order; `__ORDER_PDP_ENDIAN__` is related to 32-bit * integer ordering and is not relevant.

3369. Type for public API calls.

3370. **comment:** Rely as little as possible on compiler behavior for NaN comparison, * signed zero handling, etc. Currently never activated but may be needed * for broken compilers.
 label: code-design

3371. --- Generic BSD ---
3372. uclibc may be missing these
3373. FreeBSD
3374. not defined by default

3375. **comment:** Windows, both 32-bit and 64-bit
 label: code-design

3376. --- OpenBSD ---
3377. **comment:** * Alternative customization header * * If you want to modify the final DUK_USE_xxx flags directly (without * using the available DUK_OPT_xxx flags), define DUK_OPT_HAVE_CUSTOM_H * and tweak the final flags there.
 label: code-design

3378. Cygwin
3379. integer endianness is little on purpose
3380. e.g. `getdate_r`
3381. * Autogenerated defaults
3382. Atari Mint
3383. **comment:** GCC older than 4.6: avoid overflow warnings related to using INFINITY
 label: code-design

3384. Most portable, wastes space
3385. GCC/clang inaccurate math would break compliance and probably duk_tval, * so refuse to compile. Relax this if -ffast-math is tested to work.

3386. --- Generic ---
3387. VS2012+ has stdint.h, < VS2012 does not (but it's available for download).

3388. **comment:** * Alignment requirement and support for unaligned accesses * * Assume unaligned accesses are not supported unless specifically allowed * in the target platform. Some platforms may support unaligned accesses * but alignment to 4 or 8 may still be desirable.
 label: requirement

3389. Clang
3390. AmigaOS. Neither AMIGA nor `__amigaos__` is defined on VBCC, so user must * define 'AMIGA' manually when using VBCC.

3391. --- GCC ---
3392. Duktape/C function return value, platform int is enough for now to * represent 0, 1, or negative error code. Must be compatible with * assigning truth values (e.g. `duk_ret_t rc = (foo == bar);`).
3393. Low memory algorithm: separate chaining using arrays, fixed size hash
3394. `__ OVERRIDE_DEFINES __`
3395. <http://stackoverflow.com/questions/5919996/how-to-detect-reliably-mac-os-x-ios-linux-windows-in-c-preprocessor>
3396. vbcc + AmigaOS has C99 but no inttypes.h
3397. MSVC
3398. * Check whether or not a packed duk_tval representation is possible. * What's basically required is that pointers are 32-bit values * (`sizeof(void *) == 4`). Best effort check, not always accurate. * If guess goes wrong, crashes may result; self tests also verify * the guess.
3399. --- Linux ---
3400. --- AmigaOS ---
3401. BCC, assume we're on x86.
3402. Pointer size determination based on `__WORDSIZE` or architecture when * that's not available.
3403. Unsigned index variant.
3404. External provider already defined.
3405. Byte order varies, so rely on autodetect.
3406. DUK_USE_UNION_INITIALIZERS: required from compilers, so no fill-in.
3407. **comment:** Special naming to avoid conflict with e.g. DUK_FREE() in duk_heap.h * (which is unfortunately named). May sometimes need replacement, e.g. * some compilers don't handle zero length or NULL correctly in realloc().
 label: code-design

3408. empty
3409. --- MSVC ---
3410. We're generally assuming that we're working on a platform with a 32-bit * address space. If DUK_SIZE_MAX is a typecast value (which is necessary * if SIZE_MAX is missing), the check must be avoided because the * preprocessor can't do a comparison.
3411. 64-bit constants. Since LL / ULL constants are not always available, * use computed values. These values can't be used in preprocessor * comparisons; flag them as such.
3412. --- x86 ---
3413. --- MIPS 32-bit ---
3414. XXX: add feature options to force basic types from outside?
3415. Array index values, could be exact 32 bits. * Currently no need for signed duk_arridx_t.
3416. SIZE_MAX may be missing so use an approximate value for it.
3417. DUK_F_BCC
3418. Complex condition broken into separate parts.
3419. POSIX
3420. If not provided, use safe default for alignment.
3421. * Date provider selection * * User may define DUK_USE_DATE_GET_NOW() etc directly, in which case we'll * rely on an external provider. If this is not done, revert to previous * behavior and use Unix/Windows built-in provider.
3422. Cannot determine byte order.
3423. --- Cygwin ---
3424. !defined(DUK_USE_BYTEORDER) && defined(`__BYTE_ORDER__`)
3425. More or less standard endianness predefines provided by header files. * The ARM hybrid case is detected by assuming that `__FLOAT_WORD_ORDER__` will be big endian, see: <http://lists.mysql.com/internals/443>. * On some platforms some defines may be present with an empty value which * causes comparisons to fail: <https://github.com/svaarala/duktape/issues/453>.
3426. !defined(DUK_USE_BYTEORDER)
3427. **comment:** For custom platforms allow user to define byteorder explicitly. * Since endianness headers are not standardized, this is a useful * workaround for custom platforms for which endianness detection * is not directly supported. Perhaps custom hardware is used and * user cannot submit upstream patches.
 label: code-design

3428. **comment:** Don't know how to declare unreachable point, so don't do it; this * may cause some spurious compilation warnings (e.g. "variable used * uninitialized").
 label: code-design

3429. Good default is a bit arbitrary because alignment requirements * depend on target. See <https://github.com/svaarala/duktape/issues/102>.

3430. --- Windows ---

3431. User provided InitJS.
3432. These have been tested from VS2008 onwards; may work in older VS versions * too but not enabled by default.
3433. Error codes are represented with platform int. High bits are used * for flags and such, so 32 bits are needed.
3434. Intermediate define for 'have inttypes.h'
3435. build is not C99 or C++11, play it safe
3436. GCC: test not very accurate; enable only in relatively recent builds * because of bugs in gcc-4.4 (<http://lists.debian.org/debian-gcc/2010/04/msg00000.html>)
3437. Intel x86 (32-bit), x64 (64-bit) or x32 (64-bit but 32-bit pointers), * define only one of DUK_F_X86, DUK_F_X64, DUK_F_X32. *
- <https://sites.google.com/site/x32abi/>
3438. Enabled with debug/assertions just so that any issues can be caught.
3439. C99 or compatible
3440. --- Mac OSX, iPhone, Darwin ---
3441. C99 or above
3442. MinGW. Also GCC flags (DUK_F_GCC) are enabled now.
3443. This detection is not very reliable.
3444. Support for 48-bit signed integer duk_tval with transparent semantics.
3445. GCC and Clang provide endianness defines as built-in predefines, with * leading and trailing double underscores (e.g. __BYTE_ORDER__). See * output of "make gcpredefs" and "make clangpredefs". Clang doesn't * seem to provide __FLOAT_WORD_ORDER__; assume not mixed endian for clang. *
- <http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html>
3446. 64-bit type detection is a bit tricky. ** ULLONG_MAX is a standard define. __LONG_LONG_MAX__ and __ULONGLONG_MAX__ * are used by at least GCC (even if system headers don't provide ULLONG_MAX). * Some GCC variants may provide __LONG_LONG_MAX__ but not __ULONGLONG_MAX__. * * ULL / LL constants are rejected / warned about by some compilers, even if * the compiler has a 64-bit type and the compiler/system headers provide an * unsupported constant (ULL/LL)! Try to avoid using ULL / LL constants. * As a side effect we can only check that e.g. ULONG_MAX is larger than 32 * bits but can't be sure it is exactly 64 bits. Self tests will catch such * cases.
3447. Convert any input pointer into a "void **", losing a const qualifier. * This is not fully portable because casting through duk_uintptr_t may * not work on all architectures (e.g. those with long, segmented pointers).
3448. Small integers (16 bits or more) can fall back to the 'int' type, but * have a typedef so they are marked "small" explicitly.
3449. **comment:** Most portable, potentially wastes space
- label:** code-design
3450. PowerPC
3451. --- Solaris ---
3452. **comment:** Technically C99 (C++11) but found in many systems. On some systems * __STDC_LIMIT_MACROS and __STDC_CONSTANT_MACROS must be defined before * including stdint.h (see above).
- label:** code-design
3453. * duk_config.h configuration header generated by genconfig.py. ** Git commit: cad34ae155acb0846545ca6bf2d29f9463b22bbb * Git describe: v1.5.2 * Git branch: HEAD ** Supported platforms: * - Mac OSX, iPhone, Darwin * - OpenBSD * - Generic BSD * - Atari ST TOS * - AmigaOS * - Windows * - Flashplayer (Crossbridge) * - QNX * - TI-Nspire * - Emscripten * - Linux * - Solaris * - Generic POSIX * - Cygwin * - Generic UNIX * - Generic fallback ** Supported architectures: * - x86 * - x64 * - x32 * - ARM 32-bit * - ARM 64-bit * - MIPS 32-bit * - MIPS 64-bit * - PowerPC 32-bit * - PowerPC 64-bit * - SPARC 32-bit * - SPARC 64-bit * - SuperH * - Motorola 68k * - Emscripten * - Generic ** Supported compilers: * - Clang * - GCC * - MSVC * - Emscripten * - TinyC * - VBCC * - Bruce's C compiler * - Generic *
3454. Missing some obvious constants.
3455. No provider for DUK_USE_DATE_PARSE_STRING(), fall back to ISO 8601 only.
3456. no endian.h or stdint.h
3457. --- Atari ST TOS ---
3458. GCC: assume we have __va_copy() in non-C99 mode.
3459. User forced alignment to 4 or 8.
3460. C99 / C++11 and above: rely on va_copy() which is required.
3461. <http://bellard.org/tcc/tcc-doc.html#SEC9>
3462. Check that architecture is two's complement, standard C allows e.g. * INT_MIN to be -2**31+1 (instead of -2**31).
3463. Based on 'make checkalign' there are no alignment requirements on * Linux MIPS except for doubles, which need align by 4. Alignment * requirements vary based on target though.
3464. AmigaOS on M68k
3465. not configured for DLL build
3466. * You may add overriding #define/#undef directives below for * customization. You of course cannot un-#include or un-typedef * anything; these require direct changes above.
3467. **comment:** cannot detect 64-bit type, not always needed so don't error
- label:** code-design
3468. --- Generic UNIX ---
3469. std::exception
3470. A few types are assumed to always exist.
3471. Float word order not known, assume not a hybrid.
3472. --- Generic fallback ---
3473. **comment:** --- Generic POSIX ---
- label:** code-design
3474. no parsing (not an error)
3475. **comment:** ANSI C (various versions) and some implementations require that the * pointer arguments to memset(), memcpy(), and memmove() be valid values * even when byte size is 0 (even a NULL pointer is considered invalid in * this context). Zero-size operations as such are allowed, as long as their * pointer arguments point to a valid memory area. The DUK_MEMSET(), * DUK_MEMCPY(), and DUK_MEMMOVE() macros require this same behavior, i.e.: * (1) pointers must be valid and non-NULL, (2) zero size must otherwise be * allowed. If these are not fulfilled, a macro wrapper is needed. * *
- <http://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0> * <http://lists.cs.uiuc.edu/pipermail/llvmdev/2007-October/011065.html> ** Not sure what's the required behavior when a pointer points just past the * end of a buffer, which often happens in practice (e.g. zero size memmoves). * For example, if allocation size is 3, the following pointer would not * technically point to a valid memory byte: * * <-- alloc --> * | 0 | 1 | 2 | * ^ - p=3, points after last valid byte (2)
- label:** code-design
3476. C++11 or above
3477. SPARC byte order varies so rely on autodetection.
3478. DUK_F_PACKED_TVAL_PROVIDED
3479. <http://www.monkey.org/openbsd/archive/ports/0401/msg00089.html>
3480. VBCC
3481. sigsetjmp() alternative
3482. GCC. Clang also defines __GNUC__ so don't detect GCC if using Clang.
3483. --- SuperH ---
3484. Pre-C99: va_list type is implementation dependent. This replacement * assumes it is a plain value so that a simple assignment will work. * This is not the case on all platforms (it may be a single-array element, * for instance).
3485. **comment:** The best type for an "all around int" in Duktape internals is "at least * 32 bit signed integer" which is most convenient. Same for unsigned type. * Prefer 'int' when large enough, as it is almost always a convenient type.
- label:** code-design
3486. uclibc
3487. Explicit marker needed; may be 'defined', 'undefined', 'or 'not provided'.

3488. No provider for DUK_USE_DATE_FORMAT_STRING(), fall back to ISO 8601 only.

3489. **comment:** Many platforms are missing fpclassify() and friends, so use replacements * if necessary. The replacement constants (FP_NAN etc) can be anything but * match Linux constants now.
label: code-design

3490. * Checks for config option consistency (DUK_USE_xxx)

3491. autodetect compiler

3492. Rely on C89 headers only; time.h must be here.

3493. These functions don't currently need replacement but are wrapped for * completeness. Because these are used as function pointers, they need * to be defined as concrete C functions (not macros).

3494. --- TI-Nspire ---

3495. **comment:** We need va_copy() which is defined in C99 / C++11, so an awkward * replacement is needed for pre-C99 / pre-C++11 environments. This * will quite likely need portability hacks for some non-C99 * environments.
label: code-design

3496. AmigaOS on M68K or PPC is always big endian.

3497. Feature option forcing.

3498. --- ARM 64-bit ---

3499. * Feature option handling

3500. **comment:** * Wrapper typedefs and constants for integer types, also sanity check types. ** C99 typedefs are quite good but not always available, and we want to avoid * forcibly redefining the C99 typedefs. So, there are Duktape wrappers for * all C99 typedefs and Duktape code should only use these typedefs. Type * detection when C99 is not supported is best effort and may end up detecting * some types incorrectly. ** Pointer sizes are a portability problem: pointers to different types may * have a different size and function pointers are very difficult to manage * portably. ** http://en.wikipedia.org/wiki/C_data_types#Fixed-width_integer_types ** Note: there's an interesting corner case when trying to define minimum * signed integer value constants which leads to the current workaround of * defining e.g. -0x80000000 as (-0x7fffffffL - 1L). See doc/code-issues.txt * for a longer discussion. ** Note: avoid typecasts and computations in macro integer constants as they * can then no longer be used in macro relational expressions (such as * #if DUK_SIZE_MAX < 0xffffffffUL). There is internal code which relies on * being able to compare DUK_SIZE_MAX against a limit.
label: code-design

3501. Not standard but common enough

3502. --- QNX ---

3503. **comment:** Object property allocation layout has implications for memory and code * footprint and generated code size/speed. The best layout also depends * on whether the platform has alignment requirements or benefits from * having mostly aligned accesses.
label: code-design

3504. (v)snprintf() is missing before MSVC 2015. Note that _(v)snprintf() does * NOT NUL terminate on truncation, but Duktape code never assumes that. * http://stackoverflow.com/questions/2915672/snprintf-and-visual-studio-2010

3505. Motorola 68K. Not defined by VBCC, so user must define one of these * manually when using VBCC.

3506. http://msdn.microsoft.com/en-us/library/aa235362(VS.60).aspx

3507. Non-C99 case, still relying on DUK_UINTPTR_MAX, as long as it is not a computed value

3508. http://bellard.org/tcc/tcc-doc.html#SEC7

3509. * Intermediate helper defines

3510. **comment:** Macro hackery to convert e.g. __LINE__ to a string without formatting, * see: http://stackoverflow.com/questions/240353/convert-a-preprocessor-token-to-a-string
label: code-design

3511. Flash player (e.g. Crossbridge)

3512. no endian.h

3513. Shared includes: C89

3514. integer byte order

3515. float word order

3516. * Convert DUK_USE_BYTEORDER, from whatever source, into currently used * internal defines. If detection failed, #error out.

3517. e.g. ptrdiff_t

3518. **comment:** XXX: DUK_NOINLINE, DUK_INLINE, DUK_ALWAYS_INLINE for msvc?
label: code-design

3519. SPARC

3520. VBCC supports C99 so check only for C99 for union initializer support. * Designated union initializers would possibly work even without a C99 check.

3521. C++11 apparently ratified stdint.h

3522. QNX gcc cross compiler seems to define e.g. __LITTLEENDIAN__ or __BIGENDIAN__: * \$ /opt/qnx650/host/linux/x86/usr/bin/i486-pc-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 67:#define __LITTLEENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/mips-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 81:#define __BIGENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/arm-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 70:#define __LITTLEENDIAN__ 1

3523. At least some uclibc versions have broken floating point math. For * example, fpclassify() can incorrectly classify certain NaN formats. * To be safe, use replacements.

3524. Placeholder fix for (detection is wider than necessary): * http://llvm.org/bugs/show_bug.cgi?id=17788

3525. snprintf() is technically not part of C89 but usually available.

3526. --- x32 ---

3527. Basic integer typedefs and limits, preferably from inttypes.h, otherwise * through automatic detection.

3528. **comment:** Only include when compiling Duktape to avoid polluting application build * with a lot of unnecessary defines.
label: code-design

3529. --- VBCC ---

3530. On some systems SIZE_MAX can be smaller than max unsigned 32-bit value * which seems incorrect if size_t is (at least) an unsigned 32-bit type. * However, it doesn't seem useful to error out compilation if this is the * case.

3531. **comment:** On other platforms use layout 2, which requires some padding but * is a bit more natural than layout 3 in ordering the entries. Layout * 3 is currently not used.
label: code-design

3532. VS2005+ should have variadic macros even when they're not C99.

3533. illumos / Solaris

3534. DUK_COMPILING_DUKTAPE

3535. DUK_OPT_FORCE_BYTORDER

3536. QNX

3537. since gcc-4.5

3538. **comment:** Windows 32-bit and 64-bit are currently the same.
label: code-design

3539. This is optionally used by panic handling to cause the program to segfault * (instead of e.g. abort()) on panic. Valgrind will then indicate the C * call stack leading to the panic.

3540. IEEE float/double typedef.

3541. --- ARM 32-bit ---

3542. **comment:** XXX: This is technically not guaranteed because it's possible to configure * an x86 to require aligned accesses with Alignment Check (AC) flag.
label: code-design

3543. TinyC

3544. **comment:** Note: the funny looking computations for signed minimum 16-bit, 32-bit, and * 64-bit values are intentional as the obvious forms (e.g. -0x80000000L) are * -not- portable. See code-issues.txt for a detailed discussion.

label: code-design

3545. These are necessary wild guesses.

3546. Already provided above

3547. Atari ST TOS. `__TOS__` defined by PureC. No platform define in VBCC * apparently, so to use with VBCC user must define `__TOS__` manually.

3548. Byte order is big endian but cannot determine IEEE double word order.

3549. --- Flashplayer (Crossbridge) ---

3550. MIPS byte order varies so rely on autodetection.

3551. stdint.h not available

3552. **comment:** The most portable way to figure out local time offset is `gmtime()`, * but it's not thread safe so use with caution.

label: requirement

3553. Codepoint type. Must be 32 bits or more because it is used also for * internal codepoints. The type is signed because negative codepoints * are used as internal markers (e.g. to mark EOF or missing argument). * (X)UTF-8/CESU-8 encode/decode take and return an unsigned variant to * ensure `duk_uint32_t` casts back and forth nicely. Almost everything * else uses the signed one.

3554. DUK_USE_VARIADIC_MACROS: required from compilers, so no fill-in.

3555. TOS on M68K is always big endian.

3556. don't use `strftime()` for now

3557. **comment:** not sure, not needed with C99 anyway

label: requirement

3558. C++ doesn't have standard designated union initializers ({ .foo = 1 }).

3559. --- Emscripten ---

3560. nop

3561. DLL build detection

3562. **comment:** Avoid custom date parsing and formatting for portability.

label: code-design

3563. Workaround for older C++ compilers before including <inttypes.h>, * see e.g.: https://sourceware.org/bugzilla/show_bug.cgi?id=15366

3564. * Compiler autodetection

3565. --- SPARC 32-bit ---

3566. --- Clang ---

3567. On some platforms int is 16-bit but long is 32-bit (e.g. PureC)

3568. **comment:** * Byte order and double memory layout detection * * Endianness detection is a major portability hassle because the macros * and headers are not standardized. There's even variance across UNIX * platforms. Even with "standard" headers, details like underscore count * varies between platforms, e.g. both `_BYTE_ORDER` and `_BYT_E_ORDER` are used * (Crossbridge has a single underscore, for instance). * * The checks below are structured with this in mind: several approaches are * used, and at the end we check if any of them worked. This allows generic * approaches to be tried first, and platform/compiler specific hacks tried * last. As a last resort, the user can force a specific endianness, as it's * not likely that automatic detection will work on the most exotic platforms. * * Duktape supports little and big endian machines. There's also support * for a hybrid used by some ARM machines where integers are little endian * but IEEE double values use a mixed order (12345678 -> 43218765). This * byte order for doubles is referred to as "mixed endian".

label: code-design

3569. **comment:** Check whether we should use 64-bit integers or not. * * Quite incomplete now. Use 64-bit types if detected (C99 or other detection) * unless they are known to be unreliable. For instance, 64-bit types are * available on VBCC but seem to misbehave.

label: code-design

3570. VS2013+ supports union initializers but there's a bug involving union-inside-struct: * <https://connect.microsoft.com/VisualStudio/feedback/details/805981> * The bug was fixed (at least) in VS2015 so check for VS2015 for now: * <https://blogs.msdn.microsoft.com/vcblog/2015/07/01/c-compiler-front-end-fixes-in-vs2015/> * Manually tested using VS2013, CL reports 18.00.31101, so enable for VS2013 too.

3571. no user declarations

3572. DUK_CONFIG_H_INCLUDED

3573. BCC (Bruce's C compiler): this is a "torture target" for compilation

3574. Generic Unix (includes Cygwin)

3575. There was a curious bug where `test-bi-date-canceling.js` would fail e.g. * on 64-bit Ubuntu, gcc-4.8.1, -m32, and no -std=c99. Some date computations * using doubles would be optimized which then broke some corner case tests. * The problem goes away by adding 'volatile' to the datetime computations. * Not sure what the actual triggering conditions are, but using this on * non-C99 systems solves the known issues and has relatively little cost * on other platforms.

3576. C99 / C++11 and above: rely on `va_copy()` which is required. * Omit parenthesis on macro right side on purpose to minimize differences * to direct use.

3577. C99 types

3578. DUK_SIZE_MAX (= SIZE_MAX) is often reliable

3579. --- PowerPC 64-bit ---

3580. See: /opt/qnx650/target/qnx6/usr/include/sys/platform.h

3581. --- SPARC 64-bit ---

3582. On platforms without any alignment issues, layout 1 is preferable * because it compiles to slightly less code and provides direct access * to property keys.

3583. Strict C99 case: DUK_UINTPTR_MAX (= UINTPTR_MAX) should be very reliable

3584. autodetect platform

3585. **comment:** NetBSD 6.0 x86 (at least) has a few problems with `pow()` semantics, * see `test-bug-netbsd-math-pow.js`. Use NetBSD specific workaround. * (This might be a wider problem; if so, generalize the define name.)

label: code-design

3586. autodetect architecture

3587. In VBCC (0.0 / 0.0) results in a warning and 0.0 instead of NaN. * In MSVC (VS2010 Express) (0.0 / 0.0) results in a compile error. * Use a computed NaN (initialized when a heap is created at the * latest).

3588. Clang: assume we have `__va_copy()` in non-C99 mode.

3589. --- Bruce's C compiler ---

3590. Same as 'duk_int_t' but guaranteed to be a 'fast' variant if this * distinction matters for the CPU. These types are used mainly in the * executor where it might really matter.

3591. vbcc + AmigaOS may be missing these

3592. C99 or C++11, no known issues

3593. **comment:** XXX: DUK_LIKELY, DUK_UNLIKELY for msvc?

label: code-design

3594. Initial fix: disable secure CRT related warnings when compiling Duktape * itself (must be defined before including Windows headers). Don't define * for user code including `duktape.h`.

3595. varargs

3596. * Fill-ins for platform, architecture, and compiler

3597. e.g. `strftime`

3598. **comment:** Note: PRS and FMT are intentionally left undefined for now. This means * there is no platform specific date parsing/formatting but there is still * the ISO 8601 standard format.

label: code-design

3599. `vsnprintf()` is technically not part of C89 but usually available.

3600. Rely on autodetection for byte order, alignment, and packed tval.

3601. AmigaOS + M68K seems to have math issues even when using GCC cross * compilation. Use replacements for all AmigaOS versions on M68K * regardless of compiler.

3602. Byte order is little endian but cannot determine IEEE double word order.

3603. C++

3604. MIPS. Related defines: `__MIPSEB__`, `__MIPSEL__`, `__mips_isa_rev`, `__LP64__`

3605. VBCC is missing the built-ins even in C99 mode (perhaps a header issue).
3606. _MSC_VER
3607. **comment:** Macro for suppressing warnings for potentially unreferenced variables. * The variables can be actually unreferenced or unreferenced in some * specific cases only; for instance, if a variable is only debug printed, * it is unreferenced when debug printing is disabled.
label: code-design
3608. Byte order varies, rely on autodetection.
3609. Use _setjmp() on Apple by default, see GH-55.
3610. TI-Nspire (using Ndless)
3611. --- PowerPC 32-bit ---
3612. Some math functions are C99 only. This is also an issue with some * embedded environments using uclibc where uclibc has been configured * not to provide some functions. For now, use replacements whenever * using uclibc.
3613. Most portable
3614. Boolean values are represented with the platform 'int'.
3615. Shared includes: stdint.h is C99
3616. BSD variant
3617. Old uclibcs have a broken memcpy so use memmove instead (this is overly wide * now on purpose): <http://lists.uclibc.org/pipermail/uclibc-cvs/2008-October/025511.html>
3618. date.h is omitted, and included per platform
3619. * Architecture autodetection
3620. The most portable current time provider is time(), but it only has a * one second resolution.
3621. Convenience, e.g. gcc 4.5.1 == 40501; <http://stackoverflow.com/questions/6031819/emulating-gccs-built-in-unreachable>
3622. defined(DUK_USE_BYTEORDER)
3623. In VBCC (1.0 / 0.0) results in a warning and 0.0 instead of infinity. * Use a computed infinity (initialized when a heap is created at the * latest).
3624. For now, hash part is dropped if and only if 16-bit object fields are used.
3625. MSVC dllexport/dllimport: appropriate __declspec depends on whether we're * compiling Duktape or the application.
3626. **comment:** On Windows, assume we're little endian. Even Itanium which has a * configurable endianness runs little endian in Windows.
label: code-design
3627. **comment:** Compiler specific hackery needed to force struct size to match alignment, * see e.g. duk_hbuffer.h. * * <http://stackoverflow.com/questions/11130109/struct-size-alignment>
label: code-design
3628. Index values must have at least 32-bit signed range.
3629. **comment:** Workaround for GH-323: avoid inlining control when compiling from * multiple sources, as it causes compiler portability trouble.
label: code-design
3630. byte order
3631. Linux
3632. Based on 'make checkalign' there are no alignment requirements on * Linux SH4, but align by 4 is probably a good basic default.
3633. MSVC preprocessor defines: <http://msdn.microsoft.com/en-us/library/b0084kay.aspx> * _MSC_FULL_VER includes the build number, but it has at least two formats, see e.g. * BOOST_MSVC_FULL_VER in http://www.boost.org/doc/libs/1_52_0/boost/config/compiler/visualc.hpp
3634. ARM
3635. Fast variants of small integers, again for really fast paths like the * executor.
3636. **comment:** When C99 types are not available, we use heuristic detection to get * the basic 8, 16, 32, and (possibly) 64 bit types. The fast/least * types are then assumed to be exactly the same for now: these could * be improved per platform but C99 types are very often now available. * 64-bit types are not available on all platforms; this is OK at least * on 32-bit platforms. * * This detection code is necessarily a bit hacky and can provide typedefs * and defines that won't work correctly on some exotic platform.
label: code-design
3637. **comment:** * Cleanups (all statement parsing flows through here). * * Pop label site and reset labels. Reset 'next temp' to value at * entry to reuse temps.
label: code-design
3638. **comment:** * Reference counting helper macros. The macros take a thread argument * and must thus always be executed in a specific thread context. The * thread argument is needed for features like finalization. Currently * it is not required for INCREF, but it is included just in case. * * Note that 'raw' macros such as DUK_HEAPHDR_GET_REFCOUNT() are not * defined without DUK_USE_REFERENCE_COUNTING, so caller must #ifdef * around them.
label: code-design
3639. alloc external with size zero
3640. 'i' is the first entry we'll keep
3641. If blen <= 0xffffUL, clen is also guaranteed to be <= 0xffffUL.
3642. Set catcher regs: idx_base+0 = value, idx_base+1 = lj_type.
3643. resend state next time executor is about to run
3644. **comment:** XXX: avoid this check somehow
label: code-design
3645. Should match Function.prototype.toString()
3646. * Main struct
3647. '/'
3648. * Unwind debugger state. If we unwind while stepping * (either step over or step into), pause execution.
3649. Duktape.modLoaded[] module cache
3650. [obj handler trap]
3651. Not safe to use 'reg_varbind' as assignment expression * value, so go through a temp.
3652. **comment:** size for formatting buffers
label: code-design
3653. default case exists: go there if no case matches
3654. registers are mutable, non-deletable
3655. '{'
3656. If the debugger is active we need to force an interrupt so that * debugger breakpoints are rechecked. This is important for function * calls caused by side effects (e.g. when doing a DUK_OP_GETPROP), see * GH-303. Only needed for success path, error path always causes a * breakpoint recheck in the executor. It would be enough to set this * only when returning to an Ecmascript activation, but setting the flag * on every return should have no ill effect.
3657. get or set a range of flags; m=first bit number, n=number of bits
3658. [level obj func pc line]
3659. * Ecmascript compliant [[GetProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * May cause arbitrary side effects and invalidate (most) duk_tval * pointers.
3660. [arg1 ... argN this]
3661. * Matching order: * * Punctuator first chars, also covers comments, regexps * LineTerminator * Identifier or reserved word, also covers null/true/false literals * NumericLiteral * StringLiteral * EOF ** The order does not matter as long as the longest match is * always correctly identified. There are order dependencies * in the clauses, so it's not trivial to convert to a switch.
3662. Allocator functions.
3663. next does not exist or next is not a letter
3664. * Create and push an error object onto the top of stack. * If a "double error" occurs, use a fixed error instance * to avoid further trouble.
3665. 'enum'
3666. SameValue(NaN, NaN) = true, regardless of NaN sign or extra bits

3667. just making sure
3668. needed for inf: causes mantissa to become zero, * and rounding to be skipped.
3669. **comment:** XXX: string not shared because it is conditional
 label: code-design
3670. carry
3671. Initial bytes for markers.
3672. queue back to heap_allocated
3673. XXX: We can't resize the value stack to a size smaller than the * current top, so the order of the resize and adjusting the stack * top depends on the current vs. final size of the value stack. * The operations could be combined to avoid this, but the proper * fix is to only grow the value stack on a function call, and only * shrink it (without throwing if the shrink fails) on function * return.
3674. print provider
3675. form: { DecimalDigits , }, val1 = min count
3676. * Figure out the final, non-bound constructor, to get "prototype" * property.
3677. **comment:** * Note: * * - duk_match_regexp() is required not to longjmp() in ordinary "non-match" * conditions; a longjmp() will terminate the entire matching process. * * - Clearing saved[] is not necessary because backtracking does it * * - Backtracking also rewinds ctx.recursion back to zero, unless an * internal/limit error occurs (which causes a longjmp()) * * - If we supported anchored matches, we would break out here * unconditionally; however, Ecmascript regexps don't have anchored * matches. It might make sense to implement a fast bail-out if * the regexp begins with '^' and sp is not 0: currently we'll just * run through the entire input string, trivially failing the match * at every non-zero offset.
 label: code-design
3678. DUK_TOK_LPAREN
3679. caller ensures; rom objects are never bufferobjects now
3680. Would be nice to bulk clear the allocation, but the context * is 1-2 kilobytes and nothing should rely on it being zeroed.
3681. 1 -> needs a PUTVAR
3682. * Post-increment/decrement will return the original value as its * result value. However, even that value will be coerced using * ToNumber() which is quite awkward. Specific bytecode opcodes * are used to handle these semantics. * * Note that post increment/decrement has a "no LineTerminator here" * restriction. This is handled by duk_expr_lbp(), which forcibly terminates * the previous expression if a LineTerminator occurs before '++'/'--'.
3683. leave out trailing 'unused' elements
3684. **comment:** TODO: implement Proxy object support here
 label: requirement
3685. * Multiply + add + carry for 32-bit components using only 16x16->32 * multiplies and carry detection based on unsigned overflow. * * 1st mult, 32-bit: (A*2^16 + B) * 2nd mult, 32-bit: (C*2^16 + D) * 3rd add, 32-bit: E * 4th add, 32-bit: F * * (AC*2^16 + B) * (C*2^16 + D) + E + F * = AC*2^32 + AD*2^16 + BC*2^16 + BD + E + F * = AC*2^32 + (AD + BC)*2^16 + (BD + E + F) * = AC*2^32 + AD*2^16 + BC*2^16 + (BD + E + F)
3686. args start at index 2
3687. * Various Unicode help functions for character classification predicates, * case conversion, decoding, etc.
3688. eat colon
3689. always set; points to a reserved valstack slot
3690. Track number of escapes: necessary for proper keyword * detection.
3691. * Node.js Buffer.prototype.write(string, [offset], [length], [encoding])
3692. IdentifierPart production with IdentifierStart, ASCII, and non-BMP excluded
3693. should never happen but default here
3694. 'env'
3695. Preliminaries, required by setjmp() handler. Must be careful not * to throw an unintended error here.
3696. -> [val]
3697. emitted
3698. resume delete to target
3699. -> [... obj]
3700. success
3701. +0.0
3702. replacer is a mutation risk
3703. parse key and value
3704. **comment:** * Arithmetic operations other than '+' have number-only semantics * and are implemented here. The separate switch-case here means a * "double dispatch" of the arithmetic opcode, but saves code space. * * E5 Sections 11.5, 11.5.1, 11.5.2, 11.5.3, 11.6, 11.6.1, 11.6.2, 11.6.3.
 label: code-design
3705. No duk_bw_remove_ensure_slice(), functionality would be identical.
3706. if boolean matches A, skip next inst
3707. ($\geq e 0$) AND ($= f (\text{expt } b (- p 1))$) * * be <- ($\text{expt } b e$) == b^{e-1} * be1 <- (* be b) == ($\text{expt } b (+ e 1)$) == $b^{(e+1)}$ * r <- (* f be1 2) == $2 * f * b^{(e+1)}$ [if $b==2$ -> f * $b^{(e+2)}$] * s <- (* b 2) [if $b==2$ -> 4] * m+ <- be1 == $b^{(e+1)}$ * m- <- be == b^{e-1} * k <- 0 * B <- B * low_ok <- round * high_ok <- round
3708. Perform fixed-format rounding.
3709. If not within Ecmascript range, some integer time calculations * won't work correctly (and some asserts will fail), so bail out * if so. This fixes test-bug-date-insane-setyear.js. There is * a +/- 24h leeway in this range check to avoid a test262 corner * case documented in test-bug-date-timeval-edges.js.
3710. Failed to match the quantifier, restore lexer and parse * opening brace as a literal.
3711. [... val]
3712. 'true'
3713. **comment:** If we don't have a jmpbuf_ptr, there is little we can do * except panic. The caller's expectation is that we never * return. * * With C++ exceptions we now just propagate an uncaught error * instead of invoking the fatal error handler. Because there's * a dummy jmpbuf for C++ exceptions now, this could be changed.
 label: code-design
3714. For now shared handler is fine.
3715. may be less, since DELETED entries are NULLED by rehash
3716. Handle single character fills as memset() even when * the fill data comes from a one-char argument.
3717. OK
3718. **comment:** * Bytecode dump/load * * The bytecode load primitive is more important performance-wise than the * dump primitive. * * Unlike most Duktape API calls, bytecode dump/load is not guaranteed to be * memory safe for invalid arguments - caller beware! There's little point * in trying to achieve memory safety unless bytecode instructions are also * validated which is not easy to do with indirect register references etc.
 label: code-design
3719. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
3720. maximum loopcount for peephole optimization
3721. cleared before entering finally
3722. flags for duk_hobject_get_own_propdesc() and variants
3723. **comment:** Execute finalizers before freeing the heap, even for reachable * objects, and regardless of whether or not mark-and-sweep is * enabled. This gives finalizers the chance to free any native * resources like file handles, allocations made outside Duktape, * etc. This is quite tricky to get right, so that all finalizer * guarantees are honored. * * XXX: this perhaps requires an execution time limit.
 label: code-design
3724. useful for patching jumps later
3725. avoid attempts to add/remove object keys
3726. The get/set pointers could be 16-bit pointer compressed but it * would make no difference on 32-bit platforms because duk_tval is * 8 bytes or more anyway.
3727. * Local defines
3728. **comment:** XXX: compression

label: code-design

3729. already NULLed (by unwind)

3730. Don't allow actual chars after equal sign.

3731. -> [Object defineProperty undefined obj key desc]

3732. whether to use macros or helper function depends on call count

3733. DUK_USE_FASTINT

3734. 7: toISOString

3735. **comment:** XXX: "read only object"?

label: code-design

3736. no action

3737. **comment:** XXX: There is currently no support for writing buffer object * indexed elements here. Attempt to do so will succeed and * write a concrete property into the buffer object. This should * be fixed at some point but because buffers are a custom feature * anyway, this is relatively unimportant.

label: code-design

3738. must hold DUK_VARARGS

3739. capture is 'undefined', always matches!

3740. **comment:** * XXX: could make this a lot faster if we create the double memory * representation directly. Feasible easily (must be uniform random).

label: code-design

3741. 'raw'

3742. DUK_TOK_RPAREN

3743. request to create catch binding

3744. for object-bound identifiers

3745. **comment:** XXX: compression (as an option)

label: code-design

3746. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** This handling is now identical for C and Ecmascript functions. * C functions always have the 'NEWENV' flag set, so their * environment record initialization is delayed (which is good). ** Delayed creation (on demand) is handled in duk_js_var.c.

3747. **comment:** * Declaration binding instantiation conceptually happens when calling a * function; for us it essentially means that function prologue. The * conceptual process is described in E5 Section 10.5. ** We need to keep track of all encountered identifiers to (1) create an * identifier-to-register map ("varmap"); and (2) detect duplicate * declarations. Identifiers which are not bound to registers still need * to be tracked for detecting duplicates. Currently such identifiers * are put into the varmap with a 'null' value, which is later cleaned up. ** To support functions with a large number of variable and function * declarations, registers are not allocated beyond a certain limit; * after that limit, variables and functions need slow path access. * Arguments are currently always register bound, which imposes a hard * (and relatively small) argument count limit. ** Some bindings in E5 are not configurable (= deletable) and almost all * are mutable (writable). Exceptions are: ** - The 'arguments' binding, established only if no shadowing argument * or function declaration exists. We handle 'arguments' creation * and binding through an explicit slow path environment record. ** - The "name" binding for a named function expression. This is also * handled through an explicit slow path environment record.

label: code-design

3748. first coerce to a plain value

3749. -> [... func varmap enum]

3750. * Replace lexical environment for global scope ** Create a new object environment for the global lexical scope. * We can't just reset the _Target property of the current one, * because the lexical scope is shared by other threads with the * same (initial) built-ins.

3751. force_global

3752. NB: use 's' as temp on purpose

3753. Not odd, or y == -Infinity

3754. * Global object built-ins

3755. A bunch of helpers (for size optimization) that combine duk_expr()/duk_exptrtop() * and result conversions. ** Each helper needs at least 2-3 calls to make it worth while to wrap.

3756. Value stack: these are expressed as pointers for faster stack manipulation. * [valstack, valstack_top[is GC-reachable, [valstack_top, valstack_end[is * not GC-reachable but kept initialized as 'undefined'.

3757. Output a specified number of digits instead of using the shortest * form. Used for toPrecision() and toFixed().

3758. [body formals], formals is comma separated list that needs to be parsed

3759. bytecode execution

3760. thisArg

3761. * Not found

3762. XXX: duk_uarridx_t?

3763. E5 Section 9.4, ToInteger()

3764. force_new

3765. duk_js_call.c is required to restore the stack reserve * so we only need to reset the top.

3766. * __FILE__ / __LINE__ entry, here 'pc' is line number directly. * Sometimes __FILE__ / __LINE__ is reported as the source for * the error (fileName, lineNumber), sometimes not.

3767. avoid dereference after potential callstack realloc

3768. * Pass 2

3769. compensate for eval() call

3770. set values into ret array

3771. * Pick an object from the head (don't remove yet).

3772. fast path

3773. **comment:** Fast exit if indices are identical. This is valid for a non-existent property, * for an undefined value, and almost always for ToString() coerced comparison of * arbitrary values (corner cases where this is not the case include e.g. a an * object with varying ToString() coercion). ** The specification does not prohibit "caching" of values read from the array, so * assuming equality for comparing an index with itself falls into the category of * "caching". ** Also, compareFn may be inconsistent, so skipping a call to compareFn here may * have an effect on the final result. The specification does not require any * specific behavior for inconsistent compare functions, so again, this fast path * is OK.

label: code-design

3774. **comment:** XXX: this could be a DUK_CONSTP instead

label: code-design

3775. [... this tracedata sep this]

3776. * After arguments, allocate special registers (like shuffling temps)

3777. [ToObject(this) item1 ... itemN arr item(i)]

3778. The implementation for computing of start_pos and end_pos differs * from the standard algorithm, but is intended to result in the exactly * same behavior. This is not always obvious.

3779. This behavior mostly mimics Node.js now.

3780. **comment:** XXX: typing (duk_hcompiledfunction has duk_uint32_t)

label: code-design

3781. **comment:** XXX: because we're dealing with 'own' properties of a fresh array, * the array initializer should just ensure that the array has a large * enough array part and write the values directly into array part, * and finally set 'length' manually in the end (as already happens now).

label: code-design

3782. This may happen even after the fast path check, if exponent is * not balanced (e.g. "0e1"). Remember to respect zero sign.

3783. Encode a fastint from duk_tval ptr, no value stack effects.

3784. 'has'

3785. DUK_TOK_COMMA

3786. 'void'
3787. bufwriter for code
3788. * Abandon array failed, need to decref keys already inserted * into the beginning of new_e_k before unwinding valstack.
3789. * UTF-8 / XUTF-8 / CESU-8 constants
3790. when going backwards, we decrement cpos 'early'; * 'p' may point to a continuation byte of the char * at offset 'cpos', but that's OK because we'll * backtrack all the way to the initial byte.
3791. DUK_TOK_IDENTIFIER
3792. * Regexp recursive matching function. * * Returns 'sp' on successful match (points to character after last matched one), * NULL otherwise. * * The C recursion depth limit check is only performed in this function, this * suffices because the function is present in all true recursion required by * regexp execution.
3793. **comment:** Use recursion_limit to ensure we don't overwrite js_ctx->visiting[] * array so we don't need two counter checks in the fast path. The * slow path has a much larger recursion limit which we'll use if * necessary.
label: code-design
3794. [...] func varmap enum key value this]
3795. 1: toDateString
3796. **comment:** Fill offset handling is more lenient than in Node.js.
label: code-design
3797. Positive if local time ahead of UTC.
3798. With this check in place fast paths won't need read-only * object checks. This is technically incorrect if there are * setters that cause no writes to ROM objects, but current * built-ins don't have such setters.
3799. save stack top
3800. Insert an empty jump in the middle of code emitted earlier. This is * currently needed for compiling for-in.
3801. Treat like a debugger statement: ignore when not attached.
3802. 'import'
3803. Duktape/C API guaranteed entries (on top of args)
3804. avoid warning (unsigned)
3805. Helper which can be called both directly and with duk_safe_call().
3806. [...] template]
3807. -> [buffer]
3808. **comment:** XXX: Here a "slice copy" would be useful.
label: code-design
3809. Must ensure result is 64-bit (no overflow); a * simple and sufficient fast path is to allow only * 32-bit inputs. Avoid zero inputs to avoid * negative zero issues (-1 * 0 = -0, for instance).
3810. duk_unicode_ids_m_let_noa[]
3811. Don't intercept a DoubleError, we may have caused the initial double * fault and attempting to intercept it will cause us to be called * recursively and exhaust the C stack.
3812. **comment:** * Node.js Buffer.prototype.equals() * Node.js Buffer.prototype.compare() * Node.js Buffer.compare()
label: requirement
3813. indirect allocs
3814. Lightfuncs are always considered strict.
3815. implies DUK_HOBJECT_IS_BUFFEROBJECT
3816. safe, because matched (NUL causes a break)
3817. * Packed tval sanity
3818. longjmp type
3819. should be first on 64-bit platforms
3820. **comment:** * XXX: duk_uint_fast32_t should probably be used in many places here.
label: code-design
3821. [...] | (crud) errobj]
3822. expensive flag
3823. validation performed by duk_hexval
3824. non-strict equality from here on
3825. allocate catcher and populate it (should be atomic)
3826. since duk_abandon_array_checked() causes a resize, there should be no gaps in keys
3827. **comment:** DUK_TVAL_SET_TVAL_UPDREF() is used a lot in executor, property lookups, * etc, so it's very important for performance. Measure when changing. * * NOTE: the source and destination duk_tval pointers may be the same, and * the macros MUST deal with that correctly.
label: code-design
3828. if 0, 'str16' used, if > 0, 'strlist16' used
3829. probe sequence (open addressing)
3830. token was preceded by a lineterm
3831. -> [...]]
3832. * Store entry state.
3833. Perform an intermediate join when this many elements have been pushed * on the value stack.
3834. duk_unicode_ids_m_let_noabmp[]
3835. * Parse variant 3 or 4. * * For variant 3 (e.g. "for (A in C) D;") the code for A (except the * final property/variable write) has already been emitted. The first * instruction of that code is at pc_v34_lhs; a JUMP needs to be inserted * there to satisfy control flow needs. * * For variant 4, if the variable declaration had an initializer * (e.g. "for (var A = B in C) D;") the code for the assignment * (B) has already been emitted. * * Variables set before entering here: * * pc_v34_lhs: insert a "JUMP L2" here (see doc/compiler.rst example). * reg_temps + 0: iteration target value (written to LHS) * reg_temps + 1: enumerator object
3836. 'return'
3837. **comment:** * String table algorithm: fixed size string table with array chaining * * The top level string table has a fixed size, with each slot holding * either NULL, string pointer, or pointer to a separately allocated * string pointer list. * * This is good for low memory environments using a pool allocator: the * top level allocation has a fixed size and the pointer lists have quite * small allocation size, which further matches the typical pool sizes * needed by objects, strings, property tables, etc.
label: code-design
3838. **comment:** XXX: remove this native function and map 'stack' accessor * to the toString() implementation directly.
label: code-design
3839. * Create a RegExp instance (E5 Section 15.10.7). * * Note: the output stack left by duk_regexp_compile() is directly compatible * with the input here. * * Input stack: [escaped_source bytecode] (both as strings) * Output stack: [RegExp]
3840. **comment:** log level could be popped but that's not necessary
label: code-design
3841. thread currently running (only one at a time)
3842. numargs
3843. already defined, good
3844. * Currently only allowed only if yielding thread has only * EcmaScript activations (except for the Duktape.Thread.yield() * call at the callstack top) and none of them constructor * calls. * * This excludes the 'entry' thread which will always have * a preventCount > 0.
3845. Copy term name until end or '/'.
3846. -> [...] trap handler]
3847. Non-BMP characters within valid UTF-8 range: encode as is. * They'll decode back into surrogate pairs if the escaped * output is decoded.
3848. [...] v1 v2 name filename str] -> [...] str v2 name filename]
3849. DUK_TOK_DEBUGGER

3850. Init newly allocated slots (only).
3851. a complex (new) atom taints the result
3852. [this value]
3853. See: test-bug-tailcall-preventyield-assert.c
3854. check_object_coercible
3855. **comment:** XXX: source code property
 label: code-design
3856. **comment:** XXX: Would be nice to share the fast path loop from duk_hex_decode() * and avoid creating a temporary buffer. However, there are some * differences which prevent trivial sharing: * * - Pipe char detection * - EOF detection * - Unknown length of input and output * * The best approach here would be a bufwriter and a reasonably sized * safe inner loop (e.g. 64 output bytes at a time).
 label: code-design
3857. **comment:** XXX: unnecessary, handle in adjust
 label: code-design
3858. [... key trap handler]
3859. **comment:** * XXX: As noted above, a protected API call won't be counted as a * catcher. This is usually convenient, e.g. in the case of a top- * level duk_pcall(), but may not always be desirable. Perhaps add an * argument to treat them as catchers?
 label: code-design
3860. const
3861. even for zero-length string
3862. eat 'if'
3863. 14: getDay
3864. Custom behavior: plain buffer is used as internal buffer * without making a copy (matches Duktape.Buffer).
3865. function: create an arguments object on function call
3866. -> [... target]
3867. * Nice debug log.
3868. **comment:** XXX: copy flags using a mask
 label: code-design
3869. [... func/retval] -> [...]
3870. **comment:** XXX: optimize loops
 label: code-design
3871. up to 36 bit codepoints
3872. no fractions in internal time
3873. ditto
3874. [key] -> []
3875. Track number of escapes; count not really needed but directive * prologues need to detect whether there were any escapes or line * continuations or not.
3876. [buf? res]
3877. **comment:** Duktape 0.11.0 and prior tried to optimize the resize by not * counting the number of actually used keys prior to the resize. * This worked mostly well but also caused weird leak-like behavior * as in: test-bug-object-prop-alloc-unbounded.js. So, now we count * the keys explicitly to compute the new entry part size.
 label: code-design
3878. undefined -> skip (replaced with empty)
3879. eat opening quote on first loop
3880. **comment:** * Array built-ins * * Note that most Array built-ins are intentionally generic and work even * when the 'this' binding is not an Array instance. To ensure this, * Array algorithms do not assume "magical" Array behavior for the "length" * property, for instance. * * XXX: the "Throw" flag should be set for (almost?) all [[Put]] and * [[Delete]] operations, but it's currently false throughout. Go through * all put/delete cases and check throw flag use. Need a new API primitive * which allows throws flag to be specified. * * XXX: array lengths above 2G won't work reliably. There are many places * where one needs a full signed 32-bit range ([-0xffffffff, 0xffffffff], * i.e. -33- bits). Although array 'length' cannot be written to be outside * the unsigned 32-bit range (E5.1 Section 15.4.5.1 throws a RangeError if so) * some intermediate values may be above 0xffffffff and this may not be always * correctly handled now (duk_uint32_t is not enough for all algorithms). * * For instance, push() can legitimately write entries beyond length 0xffffffff * and cause a RangeError only at the end. To do this properly, the current * push() implementation tracks the array index using a 'double' instead of a * duk_uint32_t (which is somewhat awkward). See test-bi-array-push-maxlen.js. * * On using "put" vs. "def" prop * ===== * * Code below must be careful to use the appropriate primitive as it matters * for compliance. When using "put" there may be inherited properties in * Array.prototype which cause side effects when values are written. When * using "define" there are no such side effects, and many test262 test cases * check for this (for real world code, such side effects are very rare). * Both "put" and "define" are used in the E5.1 specification; as a rule, * "put" is used when modifying an existing array (or a non-array 'this' * binding) and "define" for setting values into a fresh result array. * * Also note that Array instance 'length' should be writable, but not * enumerable and definitely not configurable: even Duktape code internally * assumes that an Array instance will always have a 'length' property. * Preventing deletion of the property is critical.
 label: code-design
3881. no need to decref
3882. 18: getMinutes
3883. 'exports'
3884. * Unicode letter check.
3885. DUK_USE_MATH_BUILTIN
3886. No field needed when strings are in ROM.
3887. value was undefined/unsupported
3888. eat 'function'
3889. keep in valstack
3890. * Must guarantee all actually used array entries will fit into * new entry part. Add one growth step to ensure we don't run out * of space right away.
3891. step 11
3892. IdentifierPart production with IdentifierStart and ASCII excluded
3893. Internal timevalue is already NaN, so don't touch it.
3894. flag for later write
3895. GH-303
3896. Popping one element is called so often that when footprint is not an issue, * compile a specialized function for it.
3897. DUK_TOK_SEMICOLON
3898. return value from Duktape.Thread.yield()
3899. Causes a ToNumber() coercion, but doesn't break coercion order since * year is coerced first anyway.
3900. Parts are in local time, convert when setting.
3901. * E5 Section 7.4. If the multi-line comment contains a newline, * it is treated like a single line terminator for automatic * semicolon insertion.
3902. Note: if target_pc1 == i, we'll optimize a jump to itself. * This does not need to be checked for explicitly; the case * is rare and max iter breaks us out.
3903. Backup 'caller' property and update its value.
3904. assume exactly 1 arg, which is why * is forbidden; arg size still * depends on type though.
3905. Add function object.
3906. finalization
3907. [ToObject(this) item1 ... itemN arr]
3908. 0x50...0x5f
3909. Rule table: first matching rule is used to determine what to do next.
3910. CATCH flag may be enabled or disabled here; it may be enabled if * the statement has a catch block but the try block does not throw * an error.
3911. * Init the heap object

3912. Inherit from ROM-based global object: less RAM usage, less transparent.
3913. Note: 0xff != DUK_BC_C_MAX
3914. COMMA
3915. Node.js return value for noAssert out-of-bounds reads is * usually (but not always) NaN. Return NaN consistently.
3916. **comment:** function must not be tail called
 label: code-design
3917. stage 3: update length (done by caller), decide return code
3918. idx_space
3919. statement does not terminate directive prologue
3920. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined fresh_require exports module mod_func exports fresh_require exports module]
3921. **comment:** XXX: optimize for buffer inputs: no need to coerce to a string * which causes an unnecessary interning.
 label: code-design
3922. * Table for hex decoding ASCII hex digits
3923. function: create new environment when called (see duk_hcompiledfunction)
3924. pop regexp res_obj or match string
3925. P
3926. required if NAMEBINDING set
3927. slow path decode
3928. don't care value, year is mandatory
3929. really 'not applicable' anymore, should not be referenced after this
3930. skip jump conditionally
3931. * Flags parsing (see E5 Section 15.10.4.1).
3932. for updating (all are set to < 0 for virtual properties)
3933. * Run (object) finalizers in the "to be finalized" work list.
3934. 'warn'
3935. DUK_USE_FILE_IO
3936. [... constructor arg1 ... argN final_cons fallback]
3937. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur.
3938. 'Array' object, array length and index exotic behavior
3939. a statement following a label cannot be a source element * (a function declaration).
3940. 24: setMilliseconds
3941. \xffVarenv'
3942. **comment:** DUK_USE_DATE_GET_LOCAL_TZOFFSET() needs to be called with a * time value computed from UTC parts. At this point we only * have 'd' which is a time value computed from local parts, so * it is off by the UTC-to-local time offset which we don't know * yet. The current solution for computing the UTC-to-local * time offset is to iterate a few times and detect a fixed * point or a two-cycle loop (or a sanity iteration limit), * see test-bi-date-local-parts.js and test-bi-date-tzoffset-basic-fi.js. * * E5.1 Section 15.9.1.9: * UTC(t) = t - LocalTZA - DaylightSavingTA(t - LocalTZA) * * For NaN/inf, DUK_USE_DATE_GET_LOCAL_TZOFFSET() returns 0.
 label: code-design
3943. DUK_TOK_BREAK
3944. We're conceptually between two opcodes; act->pc indicates the next * instruction to be executed. This is usually the correct pc/line to * indicate in Status. (For the 'debugger' statement this now reports * the pc/line after the debugger statement because the debugger opcode * has already been executed.)
3945. DUK_TOK_MOD_EQ
3946. **comment:** * Write to 'length' of an array is a very complex case * handled in a helper which updates both the array elements * and writes the new 'length'. The write may result in an * unconditional RangeError or a partial write (indicated * by a return code). * * Note: the helper has an unnecessary writability check * for 'length', we already know it is writable.
 label: code-design
3947. **comment:** * Allocation size for 'curr_alloc' is alloc_size. There is no * automatic NUL terminator for buffers (see above for rationale). * * 'curr_alloc' is explicitly allocated with heap allocation * primitives and will thus always have alignment suitable for * e.g. duk_tval and an IEEE double.
 label: code-design
3948. * ctx->prev_token token to process with duk__expr_led() * ctx->curr_token updated by caller
3949. Note: target registers a and a+1 may overlap with DUK_REGP(b). * Careful here.
3950. * Mark unreachable, finalizable objects. * * Such objects will be moved aside and their finalizers run later. They have * to be treated as reachability roots for their properties etc to remain * allocated. This marking is only done for unreachable values which would * be swept later (refzero_list is thus excluded). * * Objects are first marked FINALIZABLE and only then marked as reachability * roots; otherwise circular references might be handled inconsistently.
3951. **comment:** XXX: accept any duk_hbufferobject type as an input also?
 label: code-design
3952. Note: DST adjustment is determined using UTC time.
3953. caller must check
3954. 2
3955. **comment:** Fresh require: require.id is left configurable (but not writable) * so that is not easy to accidentally tweak it, but it can still be * done with Object.defineProperty(). * * XXX: require.id could also be just made non-configurable, as there * is no practical reason to touch it.
 label: code-design
3956. log this with a normal debug level because this should be relatively rare
3957. * Ecmascript execution, support primitives.
3958. * replace()
3959. out_clamped=NULL, RangeError if outside range
3960. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC
3961. idx_args = idx_func + 2
3962. handle comma and closing bracket
3963. * Avoid a GC if GC is already running. See duk_heap_mem_alloc().
3964. Note: this allows creation of internal strings.
3965. DUK_FLD_DOUBLE
3966. **comment:** * Logging * * Current logging primitive is a sprintf-style log which is convenient * for most C code. Another useful primitive would be to log N arguments * from value stack (like the Ecmascript binding does).
 label: code-design
3967. caller and arguments must use the same thrower, [[ThrowTypeError]]
3968. bound function 'length' property is interesting
3969. DUK_REPLACEMENTS_H_INCLUDED
3970. [... error func]
3971. expt 0x000 is zero/subnormal
3972. There is a caller; it MUST be an Ecmascript caller (otherwise it would * match entry level check)
3973. restore PC
3974. allow negative PCs, behave as a no-op
3975. \xffValue'
3976. * Activation defines
3977. Step 15: insert itemCount elements into the hole made above

3978. No need to replace the 'enum_target' value in stack, only the * enum_target reference. This also ensures that the original * enum target is reachable, which keeps the proxy and the proxy * target reachable. We do need to replace the internal _Target.

3979. XXX: ARRAY_PART for Array prototype?

3980. Coerce value to a number before computing check_length, so that * the field type specific coercion below can't have side effects * that would invalidate check_length.

3981. string is a valid array index

3982. * Fast path: assume no mutation, iterate object property tables * directly; bail out if that assumption doesn't hold.

3983. 'Number'

3984. buffer is behind a pointer, dynamic or external

3985. * in

3986. allow caller to give a const number with the DUK__CONST_MARKER

3987. DUK_USE_JSON_DECNUMBER_FASTPATH

3988. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack (e.g. if * lval is already a string).

3989. XXX: use DUK_HSTRING_FLAG_INTERNAL?

3990. func support for [[HasInstance]] checked in the beginning of the loop

3991. %p

3992. weak refs should be handled here, but no weak refs for * any non-string objects exist right now.

3993. pop key

3994. implementation specific

3995. * Log level check

3996. **comment:** XXX: unify handling with native call.
label: code-design

3997. internal extra elements assumed on function entry, * always added to user-defined 'extra' for e.g. the * duk_check_stack() call.

3998. * Complex atom * * The original code is used as a template, and removed at the end * (this differs from the handling of simple quantifiers). * * NOTE: there is no current solution for empty atoms in complex * quantifiers. This would need some sort of a 'progress' instruction. * * XXX: impose limit on maximum result size, i.e. atom_code_len * atom_copies?

3999. Leading zero is not counted towards precision digits; not * in the integer part, nor in the fraction part.

4000. 39: setYear

4001. **comment:** * ToString() (E5 Section 9.8) * * ==> implemented in the API.
label: requirement

4002. E5.1 Section 15.9.1.6

4003. DUK_USE_JSON_QUOTESTRING_FASTPATH

4004. always in executor

4005. because of valstack init policy

4006. embed: func ptr

4007. local

4008. Sanity check

4009. * Object environment record. * * Binding (target) object is an external, uncontrolled object. * Identifier may be bound in an ancestor property, and may be * an accessor. Target can also be a Proxy which we must support * here.

4010. * Manipulate callstack for the call.

4011. Currently the bytecode executor and executor interrupt * instruction counts are off because we don't execute the * interrupt handler when we're about to exit from the initial * user call into Duktape. * * If we were to execute the interrupt handler here, the counts * would match. You can enable this block manually to check * that this is the case.

4012. **comment:** Note: combining comparison ops must be done carefully because * of uncomparable values (NaN): it's not necessarily true that * (x >= y) === !(x < y). Also, evaluation order matters, and * although it would only seem to affect the compiler this is * actually not the case, because there are also run-time coercions * of the arguments (with potential side effects). * * XXX: can be combined; check code size.
label: code-design

4013. typeof

4014. Execute bytecode until returned or longjmp().

4015. overflow not possible, buffer limits

4016. 'interface'

4017. zero everything unless requested not to do so

4018. E5 Section 15.12.3, main algorithm, step 4.b.ii steps 1-4.

4019. * Node.js Buffer.isBuffer()

4020. DUK_TOK_COLON

4021. B -> target register for next key * C -> enum register

4022. -> [... flags escaped_source bytecode]

4023. backref n -> saved indices [n*2, n*2+1]

4024. value1 -> error object

4025. remove key

4026. **comment:** * String hash computation (interning). * * String hashing is performance critical because a string hash is computed * for all new strings which are candidates to be added to the string table. * However, strings actually added to the string table go through a codepoint * length calculation which dominates performance because it goes through * every byte of the input string (but only for strings added). * * The string hash algorithm should be fast, but on the other hand provide * good enough hashes to ensure both string table and object property table * hash tables work reasonably well (i.e., there aren't too many collisions * with real world inputs). Unless the hash is cryptographic, it's always * possible to craft inputs with maximal hash collisions. * * NOTE: The hash algorithms must match src/dukutil.py:duk_heap_hashstring() * for ROM string support!
label: code-design

4027. **comment:** XXX: should the API call handle this directly, i.e. attempt * to duk_push_hobject(ctx, null) would push a null instead? * (On the other hand 'undefined' would be just as logical, but * not wanted here.)
label: code-design

4028. **comment:** For short strings, use a fixed temp buffer.
label: code-design

4029. label limits

4030. marker values for hash part

4031. DUK_TOK_GE

4032. -> [... closure]

4033. NULL pointer is required to encode to zero, so memset is enough.

4034. else an array initializer element

4035. * E5 Section 7.3 * * A LineTerminatorSequence essentially merges <CR> <LF> sequences * into a single line terminator. This must be handled by the caller.

4036. 35: setUTCMonth

4037. [... retval]

4038. Input shuffling happens before the actual operation, while output * shuffling happens afterwards. Output shuffling decisions are still * made at the same time to reduce branch clutter; output shuffle decisions * are recorded into X_out variables.

4039. 'new' MemberExpression Arguments

4040. assume plain values

4041. duk_reconst_t is unsigned, so use 0 as dummy value (ignored by caller)

4042. [... val obj]

4043. -> [... re_obj input bc saved_buf lastIndex]

4044. **comment:** (advance << 8) + token_type, updated at function end, * init is unnecessary but suppresses "may be used uninitialized" warnings.
label: code-design

4045. first, parse regexp body roughly

4046. Duktape/C (nativefunction) object, exotic 'length'

4047. DUK_HOBJECT_FLAG_EXOTIC_ARRAY varies

4048. 'this' binding

4049. [... val key] -> [... key val]

4050. isAccessor

4051. Cannot be compressed as a heap pointer because may point to * an arbitrary address.

4052. skip opening slash on first loop

4053. transfer flags

4054. DUK_TOK_INTERFACE

4055. may be undefined

4056. accept anything, expect first value (EOF will be * caught by duk__dec_value() below.)

4057. iteration statements allow continue

4058. Clamp so that values at 'clamp_top' and above are wiped and won't * retain reachable garbage. Then extend to 'nregs' because we're * returning to an Ecmascript function.

4059. 'while'

4060. **comment:** Checking this here rather than in memory alloc primitives * reduces checking code there but means a failed allocation * will go through a few retries before giving up. That's * fine because this only happens during debugging.
label: code-design

4061. **comment:** * Array abandon check; abandon if: * * new_used / new_size < limit * new_used < limit * new_size || limit is 3 bits fixed point * new_used < limit' / 8 * new_size || *8 * 8*new_used < limit' * new_size || :8 * new_used < limit' * (new_size / 8) * * Here, new_used = a_used, new_size = a_size. * * Note: some callers use approximate values for a_used and/or a_size * (e.g. dropping a '+1' term). This doesn't affect the usefulness * of the check, but may confuse debugging.
label: code-design

4062. Target is before source. Source offset is expressed as * a "before change" offset. Account for the memmove.

4063. object is constructable

4064. regexp support disabled

4065. This should be equivalent to match() algorithm step 8.f.iii.2: * detect an empty match and allow it, but don't allow it twice.

4066. assume keys are compacted

4067. currently implicitly also DUK_USE_DOUBLE_LINKED_HEAP

4068. advance, whatever the current token is; parse next token in regexp context

4069. *ToInt32(), ToUint32(), ToUint16() (E5 Sections 9.5, 9.6, 9.7)

4070. rego to allocate

4071. preallocated temporaries (2) for variants 3 and 4

4072. [regexp input]

4073. function: function object is strict

4074. DUK_TOK_ALSHIFT

4075. **comment:** XXX: what to do if _Formals is not empty but compiler has * optimized it away -- read length from an explicit property * then?
label: code-design

4076. work list for objects whose refcounts are zero but which have not been * "finalized"; avoids recursive C calls when refcounts go to zero in a * chain of objects.

4077. **comment:** context and locale specific rules which cannot currently be represented * in the caseconv bitstream: hardcoded rules in C
label: code-design

4078. * Heap string representation. * * Strings are byte sequences ordinarily stored in extended UTF-8 format, * allowing values larger than the official UTF-8 range (used internally) * and also allowing UTF-8 encoding of surrogate pairs (CESU-8 format). * Strings may also be invalid UTF-8 altogether which is the case e.g. with * strings used as internal property names and raw buffers converted to * strings. In such cases the 'clen' field contains an inaccurate value. * * Ecmascript requires support for 32-bit long strings. However, since each * 16-bit codepoint can take 3 bytes in CESU-8, this representation can only * support about 1.4G codepoint long strings in extreme cases. This is not * really a practical issue.

4079. call is a direct eval call

4080. varname is still reachable

4081. Leave 'setter' on stack

4082. seconds

4083. **comment:** * Ecmascript compiler. * * Parses an input string and generates a function template result. * Compilation may happen in multiple contexts (global code, eval * code, function code). * * The parser uses a traditional top-down recursive parsing for the * statement level, and an operator precedence based top-down approach * for the expression level. The attempt is to minimize the C stack * depth. Bytecode is generated directly without an intermediate * representation (tree), at the cost of needing two passes over each * function. * * The top-down recursive parser functions are named "duk__parse_XXX". * * Recursion limits are in key functions to prevent arbitrary C recursion: * function body parsing, statement parsing, and expression parsing. * * See doc/compiler.rst for discussion on the design. * * A few typing notes: * * - duk_reconst_t: unsigned, no marker value for "none" * - duk_reg_t: signed, < 0 = none * - PC values: duk_int_t, negative values used as markers
label: code-design

4084. Append a "(line NNN)" to the "message" property of any error * thrown during compilation. Usually compilation errors are * SyntaxErrors but they can also be out-of-memory errors and * the like.

4085. **comment:** XXX: any way to detect faster whether something needs to be closed? * We now look up _Callee and then skip the rest.
label: requirement

4086. DUK_USE_VARIADIC_MACROS

4087. * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * ieee value = 1.ffff... * 2^(e - 1023) (normal) * = 0.ffff... * 2^(-1022) (denormal) * * algorithm v = f * b^e

4088. key is now reachable in the valstack

4089. MakeDate

4090. Request callback should push values for reply to client onto valstack

4091. inserted jump

4092. [enum_target res]

4093. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize or a forced relocating realloc?
label: code-design

4094. For Ecmascript strings, this check can only match for * initial UTF-8 bytes (not continuation bytes). For other * strings all bets are off.

4095. Then reuse the unwound activation; callstack was not shrunk so there is always space

4096. DUK_USE_DEBUGGER_DUMPHEAP || DUK_USE_DEBUGGER_INSPECT

4097. for storing/restoring the varmap binding for catch variable

4098. r <- (2 * f) * b^(e+1)

4099. [...]

4100. **comment:** Careful here and with other duk_put_prop_xxx() helpers: the * target object and the property value may be in the same value * stack slot (unusual, but still conceptually clear).
label: code-design

4101. function is strict

4102. must work for nargs <= 0

4103. Ignore digits beyond a radix-specific limit, but note them * in expt_adj.

4104. don't run finalizers; leave finalizable objects in finalize_list for next round

4105. [this value] -> [value this]

4106. Update 'buffer_length' to be the effective, safe limit which * takes into account the underlying buffer. This value will be * potentially invalidated by any side effect.
4107. 2: toTimeString
4108. **comment:** No need for constants pointer (= same as data). * * When using 16-bit packing alignment to 4 is nice. 'funcs' will be * 4-byte aligned because 'constants' are duk_tvals. For now the * inner function pointers are not compressed, so that 'bytecode' will * also be 4-byte aligned.
label: code-design
4109. * Initialize function state for a zero-argument function
4110. Guaranteed by recursion_limit setup so we don't have to * check twice.
4111. line continuation
4112. IsGenericDescriptor(desc) == true; this means in practice that 'desc' * only has [[Enumerable]] or [[Configurable]] flag updates, which are * allowed at this point.
4113. '-Infinity'
4114. * Resume a thread. * * The thread must be in resumable state, either (a) new thread which hasn't * yet started, or (b) a thread which has previously yielded. This method * must be called from an EcmaScript function. * * Args: * - thread * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.
4115. type tag (public)
4116. **comment:** It'd be nice to do something like this - but it doesn't * work for closures created inside the catch clause.
label: code-design
4117. don't re-enter e.g. during Eval
4118. mp <- b^e
4119. special handling of fmt==NULL
4120. Ensure space for final configuration (idx_rebase + num_stack_rets) * and intermediate configurations.
4121. parse value
4122. **comment:** Note: not honoring round-to-even should work but now generates incorrect * results. For instance, 1e23 serializes to "a000...", i.e. the first digit * equals the radix (10). Scaling stops one step too early in this case. * Don't know why this is the case, but since this code path is unused, it * doesn't matter.
label: code-design
4123. **comment:** The 'strict' flag is copied to get the special [[Get]] of E5.1 * Section 15.3.5.4 to apply when a 'caller' value is a strict bound * function. Not sure if this is correct, because the specification * is a bit ambiguous on this point but it would make sense.
label: code-design
4124. function may call direct eval
4125. **comment:** _Varmap: omitted if function is guaranteed not to do slow path identifier * accesses or if it would turn out to be empty of actual register mappings * after a cleanup. When debugging is enabled, we always need the varmap to * be able to lookup variables at any point.
label: code-design
4126. [array totalLength]
4127. Opcode interval for a Date-based status/peak rate limit check. Only * relevant when debugger is attached. Requesting a timestamp may be a * slow operation on some platforms so this shouldn't be too low. On the * other hand a high value makes Duktape react to a pause request slowly.
4128. -> [... key val val']
4129. [... re_obj input] -> [... re_obj input bc]
4130. valstack slot for 1st token value
4131. side effects, possibly errors
4132. NOTE: Multiple evaluation of 'ptr' in this macro.
4133. Overflow of the execution counter is fine and doesn't break * anything here.
4134. mark-and-sweep: finalized (on previous pass)
4135. curr is accessor, desc is data
4136. [func thisArg arg1 ... argN]
4137. XXX: the insert helpers should ensure that the bytecode result is not * larger than expected (or at least assert for it). Many things in the * bytecode, like skip offsets, won't work correctly if the bytecode is * larger than say 2G.
4138. this function
4139. **comment:** 'tv' becomes invalid
label: code-design
4140. is an actual function (not global/eval code)
4141. Catch attempts to use out-of-range reg/const. Without this * check Duktape 0.12.0 could generate invalid code which caused * an assert failure on execution. This error is triggered e.g. * for functions with a lot of constants and a try-catch statement. * Shuffling or opcode semantics change is needed to fix the issue. * See: test-bug-trycatch-many-constants.js.
4142. 'callee'
4143. Initial stack size satisfies the stack spare constraints so there * is no need to require stack here.
4144. 'byteOffset'
4145. **comment:** duk_handle_call_xxx: constructor call (i.e. called as 'new Foo()')
label: code-design
4146. no operators
4147. Strict: never allow function declarations outside top level.
4148. [... arg1 ... argN envobj]
4149. * Field access
4150. 0x30...0x3f
4151. value1 -> label number, pseudo-type to indicate a continue continuation (for ENDFIN)
4152. true
4153. **comment:** XXX: common reg allocation need is to reuse a sub-expression's temp reg, * but only if it really is a temp. Nothing fancy here now.
label: code-design
4154. only use the highest bit
4155. Function values are handled completely above (including * coercion results):
4156. distinct bignums, easy mistake to make
4157. **comment:** XXX: this is now a very unoptimal implementation -- this can be * made very simple by direct manipulation of the object internals, * given the guarantees above.
label: code-design
4158. Split time value into parts. The time value is assumed to be an internal * one, i.e. finite, no fractions. Possible local time adjustment has already * been applied when reading the time value.
4159. If flags != 0 (strict or SameValue), thr can be NULL. For loose * equals comparison it must be != NULL.
4160. DUK_TOK_INSTANCEOF
4161. time
4162. update entry, allocating if necessary
4163. A structure for 'snapshotting' a point for rewinding
4164. * EcmaScript call
4165. cannot be arguments exotic
4166. **comment:** this is not strictly necessary, but helps debugging
label: code-design
4167. * Prototypes for built-in functions not automatically covered by the * header declarations emitted by genbuiltins.py.
4168. push initial function call to new thread stack; this is * picked up by resume().
4169. 31: setUTCHours
4170. Decode a one-bit flag, and if set, decode a value of 'bits', otherwise return * default value. Return value is signed so that negative marker value can be * used by caller as a "not present" value.

4171. format plus something to avoid just missing
4172. **comment:** XXX: post-checks (such as no duplicate keys)
 label: code-design
4173. if property already exists, overwrites silently
4174. **comment:** 'val' is never NaN, so no need to normalize
 label: code-design
4175. * Lowercase digits for radix values 2 to 36. Also doubles as lowercase * hex nybble table.
4176. may be reg or const
4177. decl name
4178. rnd_state for duk_util_tinyrandom.c
4179. Note: heap->heap_thread, heap->curr_thread, and heap->heap_object * are on the heap allocated list.
4180. jump is inserted here
4181. misc
4182. 'length' and other virtual properties are not * enumerable, but are included if non-enumerable * properties are requested.
4183. only flag now
4184. [... func this arg1 ... argN]
4185. record previous atom info in case next token is a quantifier
4186. -"-
4187. * Ecmascript compliant [[GetOwnProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * Notes: * * - Getting a property descriptor may cause an allocation (and hence * GC) to take place, hence reachability and refcount of all related * values matter. Reallocation of value stack, properties, etc may * invalidate many duk_tval pointers (concretely, those which reside * in memory areas subject to reallocation). However, heap object * pointers are never affected (heap objects have stable pointers). * * - The value of a plain property is always reachable and has a non-zero * reference count. * * - The value of a virtual property is not necessarily reachable from * elsewhere and may have a refcount of zero. Hence we push it onto * the valstack for the caller, which ensures it remains reachable * while it is needed. * * - There are no virtual accessor properties. Hence, all getters and * setters are always related to concretely stored properties, which * ensures that the get/set functions in the resulting descriptor are * reachable and have non-zero refcounts. Should there be virtual * accessor properties later, this would need to change.
4188. not enumerable
4189. %f and %lf both consume a 'long'
4190. * Catcher defines
4191. never executed
4192. **comment:** At least 'magic' has a significant impact on function * identity.
 label: code-design
4193. 'c'
4194. important to do this *after* pushing, to make the thread reachable for gc
4195. [offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
4196. Should not happen.
4197. init pointer fields to null
4198. * toString(), valueOf()
4199. success, fixup pointers
4200. **comment:** XXX: several pointer comparison issues here
 label: code-design
4201. insert (DUK_REOP_WIPERANGE, start, count) in reverse order so the order ends up right
4202. Check that value is a duk_hbufferobject and return a pointer to it.
4203. Note: DUK_VALSTACK_INITIAL_SIZE must be >= DUK_VALSTACK_API_ENTRY_MINIMUM * + DUK_VALSTACK_INTERNAL_EXTRA so that the initial stack conforms to spare * requirements.
4204. **comment:** XXX: may need a 'length' filter for forEach()
 label: code-design
4205. Note: assumes that duk_util_probe_steps size is 32
4206. guaranteed when building arguments
4207. updates thread state, minimizes its allocations
4208. '{"_inf":true}'
4209. **comment:** Without variadic macros resort to comma expression trickery to handle debug * prints. This generates a lot of harmless warnings. These hacks are not * needed normally because DUK_D() and friends will hide the entire debug log * statement from the compiler.
 label: code-design
4210. **comment:** TimeClip() should never be necessary
 label: code-design
4211. 0x10-0x1f
4212. E5.1 Section B.2.2, step 7.
4213. Parser part masks.
4214. end switch (tok)
4215. negative -> complex atom
4216. -> [... val root ""]
4217. result array
4218. XXX: fast path for arrays?
4219. -> [... ToObject(this) ToUint32(length)]
4220. Continue checked execution if there are breakpoints or we're stepping. * Also use checked execution if paused flag is active - it shouldn't be * because the debug message loop shouldn't terminate if it was. Step out * is handled by callstack unwind and doesn't need checked execution. * Note that debugger may have detached due to error or explicit request * above, so we must recheck attach status.
4221. 0x00...0x0f
4222. coercion order matters
4223. 'jc'
4224. 1 if property found, 0 otherwise
4225. fixed-format
4226. not strictly necessary because of lookahead '}' above
4227. **comment:** XXX: must be able to represent -len
 label: code-design
4228. DUK_USE_DPRINT_COLORS
4229. expect_eof
4230. borrowed reference to catch variable name (or NULL if none)
4231. **comment:** XXX: these could be implemented as macros calling an internal function * directly. * XXX: same issue as with Duktape.fin: there's no way to delete the property * now (just set it to undefined).
 label: code-design
4232. **comment:** XXX: set magic directly here? (it could share the c_nargs arg)
 label: code-design
4233. lookup should prevent this

4234. * Preliminary activation record and valstack manipulation. * The concrete actions depend on whether we're dealing * with a tail call (reuse an existing activation), a resume, * or a normal call. * * The basic actions, in varying order, are: * * - Check stack size for call handling * - Grow call stack if necessary (non-tail-calls) * - Update current activation (idx_retval) if necessary * (non-tail, non-resume calls) * - Move start of args (idx_args) to valstack bottom * (tail calls) * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.

4235. result index for filter()

4236. either nonzero value is ok

4237. bc

4238. * DELVAR * * See E5 Sections: * 11.4.1 The delete operator * 10.2.1.1.5 DeleteBinding (N) [declarative environment record] * 10.2.1.2.5 DeleteBinding (N) [object environment record] * * Variable bindings established inside eval() are deletable (configurable), * other bindings are not, including variables declared in global level. * Registers are always non-deletable, and the deletion of other bindings * is controlled by the configurable flag. * * For strict mode code, the 'delete' operator should fail with a compile * time SyntaxError if applied to identifiers. Hence, no strict mode * run-time deletion of identifiers should ever happen. This function * should never be called from strict mode code!

4239. * Mark refzero_list objects. * * Objects on the refzero_list have no inbound references. They might have * outbound references to objects that we might free, which would invalidate * any references held by the refzero objects. A refzero object might also * be rescued by refcount finalization. Refzero objects are treated as * reachability roots to ensure they (or anything they point to) are not * freed in mark-and-sweep.

4240. Negative zero needs special handling in JX/JC because * it would otherwise serialize to '0', not '-0'.

4241. can't resize below 'top'

4242. round up roughly to next 'grow step'

4243. **comment:** * E5 Section 7.6: * * IdentifierPart: * IdentifierStart * UnicodeCombiningMark * UnicodeDigit * UnicodeConnectorPunctuation * <ZWNJ> [U+200C] * <ZWJ> [U+200D] * * IdentifierPart production has one multi-character production * as part of its IdentifierStart alternative. The '\v' character * of an escape sequence is not matched here, see discussion in * duk_unicode_is_identifier_start(). * * To match non-ASCII characters (codepoints >= 0x80), a very slow * linear range-by-range scan is used. The codepoint is first compared * to the IdentifierStart ranges, and if it doesn't match, then to a * set consisting of code points in IdentifierPart but not in * IdentifierStart. This is done to keep the unicode range data small, * at the expense of speed. * * The ASCII fast path consists of: * * 0x0030 ... 0x0039 ['0' ... '9', UnicodeDigit] * 0x0041 ... 0x005a ['A' ... 'Z', IdentifierStart] * 0x0061 ... 0x007a ['a' ... 'z', IdentifierStart] * 0x0024 ['\$', IdentifierStart] * 0x005f['_', IdentifierStart and * UnicodeConnectorPunctuation] * * UnicodeCombiningMark has no code points <= 0x7f. * * The matching code reuses the "identifier start" tables, and then * consults a separate range set for characters in "identifier part" * but not in "identifier start". These can be extracted with the * "src/extract_chars.py" script. * * UnicodeCombiningMark -> categories Mn, Mc * UnicodeDigit -> categories Nd * UnicodeConnectorPunctuation -> categories Pc
label: code-design

4244. last array index explicitly initialized, +1

4245. Value would yield 'undefined', so skip key altogether. * Side effects have already happened.

4246. need side effects, not value

4247. * Check function name validity now that we know strictness. * This only applies to function declarations and expressions, * not setter/getter name. * * See: test-dev-strict-mode-boundary.js

4248. Target may be a Proxy or property may be an accessor, so we must * use an actual, Proxy-aware hasprop check here. * * out->holder is NOT set to the actual duk_hobject where the * property is found, but rather the object binding target object.

4249. Note: for srclen=0, src may be NULL

4250. never an empty match, so step 13.c.iii can't be triggered

4251. Outer executor with setjmp/longjmp handling.

4252. if B/C is => this value, refers to a const

4253. if debugging disabled

4254. if true, the stack already contains the final result

4255. * E5 Section 7.2 specifies six characters specifically as * white space: * * 0009;<control>;Cc;0;S;;;;N;CHARACTER TABULATION;;;; * 000B;
<control>;Cc;0;S;;;;N;LINE TABULATION;;;; * 000C;<control>;Cc;0;WS;;;;N;FORM FEED (FF);; * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK
SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; * FEFF;ZERO WIDTH NO-BREAK SPACE;Cf;0;BN;;;;N;BYTE ORDER MARK;;;; * * It
also specifies any Unicode category 'Zs' characters as white * space. These can be extracted with the "src/extract_chars.py" script. * Current result: * * RAW
OUTPUT: * ===== * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; *
1680;OGHAM SPACE MARK;Zs;0;WS;;;;N;;;; * 180E;MONGOLIAN VOWEL SEPARATOR;Zs;0;WS;;;;N;;;; * 2000;EN QUAD;Zs;0;WS;2002;;;;N;;;; *
2001;EM QUAD;Zs;0;WS;2003;;;;N;;;; * 2002;EN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2003;EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; *
2004;THREE-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2005;FOUR-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2006;SIX-PER-EM
SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2007;FIGURE SPACE;Zs;0;WS;<noBreak> 0020;;;;N;;;; * 2008;PUNCTUATION SPACE;Zs;0;WS;<compat>
0020;;;;N;;;; * 2009;THIN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 200A;HAIR SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 202F;NARROW NO-BREAK
SPACE;Zs;0;CS;<noBreak> 0020;;;;N;;;; * 205F;MEDIUM MATHEMATICAL SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 3000;IDEOGRAPHIC
SPACE;Zs;0;WS;<wide> 0020;;;;N;;;; * * RANGES: * ===== * 0x0020 * 0x0a0 * 0x1680 * 0x180e * 0x2000 ... 0x200a * 0x202f * 0x205f * 0x3000 * * A
manual decoder (below) is probably most compact for this.

4256. ToNumber(bool) is +1.0 or 0.0. Tagged boolean value is always 0 or 1.

4257. when nothing is running, API calls are in non-strict mode

4258. **comment:** String length is computed here to avoid multiple evaluation * of a macro argument in the calling side.
label: code-design

4259. see duk_js_executor.c

4260. Convert heap string index to a token (reserved words)

4261. **comment:** XXX: refactor and share with other code
label: code-design

4262. order must match constants in genbuiltins.py

4263. Buffer is kept as is, with the fixed/dynamic nature of the * buffer only changed if requested. An external buffer * is converted into a non-external dynamic buffer in a * duk_to_dynamic_buffer() call.

4264. * Ecmascript compliant [[Delete]](P, Throw).

4265. -> [... str]

4266. args go here in parens

4267. 0xb0...0xbf

4268. Write in big endian

4269. only need to guarantee 1 more slot, but allocation growth is in chunks

4270. **comment:** alloc function typedefs in duktape.h
label: code-design

4271. i >= 0 would always be true

4272. **comment:** XXX: inline into a prototype walking loop?
label: code-design

4273. Lightfunc handling by ToObject() coercion.

4274. DUK_BUFOBJ_UINT16ARRAY

4275. 'else'

4276. 'String' object, array index exotic behavior

4277. A -> target reg * B -> object reg/const (may be const e.g. in "foo[1]") * C -> key reg/const

4278. There is no eval() special handling here: eval() is never * automatically converted to a lightfunc.

4279. **comment:** XXX: identify enumeration target with an object index (not top of stack)
label: code-design

4280. Write unsigned 32-bit integer.

4281. The underlying types for offset/length in duk_hbufferobject is * duk_uint_t; make sure argument values fit and that offset + length * does not wrap.

4282. [body formals source template closure]

4283. no refcount changes
4284. [arg1 ... argN-1 body] -> [body arg1 ... argN-1]
4285. First evaluate LHS fully to ensure all side effects are out.
4286. properties object
4287. Call handling and success path. Success path exit cleans * up almost all state.
4288. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer * stack[4] = regexp match OR match string
4289. object is extensible
4290. * The length comparisons are present to handle * strings like "use strict\u0000foo" as required.
4291. lookup name from an open declarative record's registers
4292. relookup (side effects)
4293. **comment:** XXX: make a common DUK_USE_ option, and allow custom fixed seed?
label: code-design
4294. x <- y * z
4295. * Note: assuming new_a_size == 0, and that entry part contains * no conflicting keys, refcounts do not need to be adjusted for * the values, as they remain exactly the same. * * The keys, however, need to be interned, incref'd, and be * reachable for GC. Any intern attempt may trigger a GC and * claim any non-reachable strings, so every key must be reachable * at all times. * * A longjmp must not occur here, as the new_p allocation would * be freed without these keys being decref'd, hence the messy * decref handling if intern fails.
4296. **comment:** * 'props' contains {key,value,flags} entries, optional array entries, and * an optional hash lookup table for non-array entries in a single 'sliced' * allocation. There are several layout options, which differ slightly in * generated code size/speed and alignment/padding; duk_features.h selects * the layout used. * * Layout 1 (DUK_USE_HOBJECT_LAYOUT_1): * * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * * Layout 2 (DUK_USE_HOBJECT_LAYOUT_2): * * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_uint8_t) + pad bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * * Layout 3 (DUK_USE_HOBJECT_LAYOUT_3): * * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * * In layout 1, the 'e_next' count is rounded to 4 or 8 on platforms * requiring 4 or 8 byte alignment. This ensures proper alignment * for the entries, at the cost of memory footprint. However, it's * probably preferable to use another layout on such platforms instead. * * In layout 2, the key and value parts are swapped to avoid padding * the key array on platforms requiring alignment by 8. The flags part * is padded to get alignment for array entries. The 'e_next' count does * not need to be rounded as in layout 1. * * In layout 3, entry values and array values are always aligned properly, * and assuming pointers are at most 8 bytes, so are the entry keys. Hash * indices will be properly aligned (assuming pointers are at least 4 bytes). * Finally, flags don't need additional alignment. This layout provides * compact allocations without padding (even on platforms with alignment * requirements) at the cost of a bit slower lookups. * * Objects with few keys don't have a hash index; keys are looked up linearly, * which is cache efficient because the keys are consecutive. Larger objects * have a hash index part which contains integer indexes to the entries part. * * A single allocation reduces memory allocation overhead but requires more * work when any part needs to be resized. A sliced allocation for entries * makes linear key matching faster on most platforms (more locality) and * skimps on flags size (which would be followed by 3 bytes of padding in * most architectures if entries were placed in a struct). * * 'props' also contains internal properties distinguished with a non-BMP * prefix. Often used properties should be placed early in 'props' whenever * possible to make accessing them as fast as possible.
label: code-design
4297. Catches >0x100000000 and negative values.
4298. * Entry points
4299. * Error built-ins
4300. Packed or unpacked tval
4301. * PC-to-line constants
4302. top-down expression parser
4303. E5 Section 15.4.5.1, steps 4.e.i - 4.e.ii
4304. fall through
4305. x
4306. **comment:** XXX: write protect after flag? -> any chance of handling it here?
label: code-design
4307. Term was '!' and is eaten entirely (including dup slashes).
4308. fast path for ASCII
4309. num_stack_res
4310. %lx
4311. * Not found as concrete or virtual
4312. 'Buffer'
4313. [... env target target]
4314. DUK_TOK_LOR
4315. DUK_OP_DECLVAR flags in A; bottom bits are reserved for propdesc flags (DUK_PROPDESC_FLAG_XXX)
4316. Day-of-month is one-based in the API, but zero-based * internally, so fix here. Note that month is zero-based * both in the API and internally.
4317. The coercion order must match the ToPropertyDescriptor() algorithm * so that side effects in coercion happen in the correct order. * (This order also happens to be compatible with duk_def_prop(), * although it doesn't matter in practice.)
4318. triggers garbage digit check below
4319. relevant array index is non-configurable, blocks write
4320. **comment:** the NaN variant we use
label: code-design
4321. * Compact the function template.
4322. PRIMARY EXPRESSIONS
4323. * indexOf(), lastIndexOf()
4324. Sometimes this assert is not true right now; it will be true after * rounding. See: test-bug-numconv-mantissa-assert.js.
4325. Automatic error throwing, retval check.
4326. Fast path for the case where the register * is a number (e.g. loop counter).
4327. always push some string
4328. spare
4329. Note: new_e_next matches pushed temp key count, and nothing can * fail above between the push and this point.
4330. * Heap flags
4331. Object extra properties. * * There are some difference between function templates and functions. * For example, function templates don't have .length and nargs is * normally used to instantiate the functions.
4332. We intentionally ignore the duk_safe_call() return value and only * check the output type. This way we don't also need to check that * the returned value is indeed a string in the success case.
4333. * Arguments exotic behavior not possible for new properties: all * magically bound properties are initially present in the arguments * object, and if they are deleted, the binding is also removed from * parameter map.
4334. get_value
4335. clear array part flag only after switching
4336. !'
4337. tv1 points to value storage
4338. Map DUK_HBUFFEROBJECT_ELEM_xxx to prototype object built-in index. * Sync with duk_hbufferobject.h.

4339. * Create "fallback" object to be used as the object instance, * unless the constructor returns a replacement value. * Its internal prototype needs to be set based on "prototype" * property of the constructor.

4340. Slightly modified "Bernstein hash" from: ** http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx ** Modifications: string skipping and reverse direction similar to * Lua 5.1.5, and different hash initializer. ** The reverse direction ensures last byte it always included in the * hash which is a good default as changing parts of the string are * more often in the suffix than in the prefix.

4341. ascii fast path: avoid decoding utf-8

4342. just in case

4343. entry_top + 4

4344. m- < (* m- B)

4345. UTC

4346. no valstack space check

4347. End of input (NUL) goes through slow path and causes SyntaxError.

4348. [... enum_target res trap_result val]

4349. **comment:** XXX: unnecessary '%s' formatting here, but cannot use * 'msg' as a format string directly.
label: code-design

4350. ptr_curr_pc != NULL only when bytecode executor is active.

4351. Restore entry thread executor curr_pc stack frame pointer.

4352. Storing the entry top is cheaper here to ensure stack is correct at exit, * as there are several paths out.

4353. duk_hobject_set_length_zero(thr, func->h_funcs);

4354. fall thru

4355. see above

4356. * Struct defines

4357. [... name reg/null] -> [...]

4358. mandatory if du.d is a NaN

4359. duk_tval ptr for 'func' on stack (borrowed reference)

4360. For tv1 == tv2, this is a no-op (no explicit check needed).

4361. Use b[] to access the size of the union, which is strictly not * correct. Can't use fixed size unless there's feature detection * for pointer byte size.

4362. local copy to avoid relookups

4363. **comment:** XXX: this has some overlap with object inspection; remove this and make * DumpHeap return lists of heapptrs instead?
label: code-design

4364. Return to the bytecode executor caller which will unwind stacks. * Return value is already on the stack top: [... retval].

4365. * Hobject Ecmascript [[Class]].

4366. Handle one slow path unit (or finish if we're done).

4367. **comment:** Copy the .buffer property, needed for TypedArray.prototype.subarray(). ** XXX: limit copy only for TypedArray classes specifically?
label: code-design

4368. enough to cover the whole mantissa

4369. DUK_TOK_BXOR_EQ

4370. **comment:** * Hobject enumeration support. ** Creates an internal enumeration state object to be used e.g. with for-in * enumeration. The state object contains a snapshot of target object keys * and internal control state for enumeration. Enumerator flags allow caller * to e.g. request internal/non-enumerable properties, and to enumerate only * "own" properties. ** Also creates the result value for e.g. Object.keys() based on the same * internal structure. ** This snapshot-based enumeration approach is used to simplify enumeration: * non-snapshot-based approaches are difficult to reconcile with mutating * the enumeration target, running multiple long-lived enumerators at the * same time, garbage collection details, etc. The downside is that the * enumerator object is memory inefficient especially for iterating arrays.
label: code-design

4371. repl string scan

4372. [key result] -> [result]

4373. '\xffTracedata'

4374. ... and its 'message' from an instance property

4375. AUTHORS.rst

4376. Potential direct eval call detected, flag the CALL * so that a run-time "direct eval" check is made and * special behavior may be triggered. Note that this * does not prevent 'eval' from being register bound.

4377. prefer jump

4378. **comment:** * Arithmetic, binary, and logical helpers. ** Note: there is no opcode for logical AND or logical OR; this is on * purpose, because the evalution order semantics for them make such * opcodes pretty pointless: short circuiting means they are most * comfortably implemented as jumps. However, a logical NOT opcode * is useful. ** Note: careful with duk_tval pointers here: they are potentially * invalidated by any DECREF and almost any API call. It's still * preferable to work without making a copy but that's not always * possible.
label: code-design

4379. automatic length update

4380. args

4381. 'String'

4382. With ROM-based strings, heap->strs[] and thr->strs[] are omitted * so nothing to initialize for strs[].

4383. * Note: prototype chain is followed BEFORE first comparison. This * means that the instanceof lval is never itself compared to the * rval.prototype property. This is apparently intentional, see E5 * Section 15.3.5.3, step 4.a. ** Also note: * * js> (function() {}) instanceof Function * true * js> Function instanceof Function * true * * For the latter, h_proto will be Function.prototype, which is the * built-in Function prototype. Because Function.[[Prototype]] is * also the built-in Function prototype, the result is true.

4384. not a vararg function

4385. Shouldn't happen but check anyway.

4386. E5 Sections 11.8.3, 11.8.5; x <= y --> not (x > y) --> not (y < x)

4387. Used by duk_emit*() calls so that we don't shuffle the loadints that * are needed to handle indirect opcodes.

4388. No strs[] pointer.

4389. **comment:** XXX: assertion that entries >= old_len are already unused
label: code-design

4390. e.g. DUK_OP_PREINCR

4391. eat 'while'

4392. nuke values at idx_rebase to get the first retval (initially * at idx_rcbase) to idx_rebase

4393. input radix

4394. char: use int cast

4395. result

4396. exponent digit

4397. **comment:** This is a cleaner approach and also produces smaller code than * the other alternative. Use duk_require_string() for format * safety (although the source property should always exist).
label: code-design

4398. defprop_flags

4399. s <- 2

4400. spaces (nargs - 1) + newline

4401. * Parse inner function

4402. if accessor without getter, return value is undefined

4403. **comment:** * For non-ASCII strings, we need to scan forwards or backwards * from some starting point. The starting point may be the start * or end of the string, or some cached midpoint in the string * cache. ** For "short" strings we simply scan without checking or updating * the cache. For longer strings we check and update the cache as * necessary, inserting a new cache entry if none exists.
label: code-design

4404. XXX: len >= 0x80000000 won't work below because we need to be * able to represent -len.

4405. who resumed us (if any)

4406. out_clamped==NULL -> RangeError if outside range

4407. relookup after possible realloc

4408. [... regexp_object escaped_source bytecode]

4409. m+ <- (* m+ B)

4410. bytecode opcode (or extraop) for binary ops

4411. forget temp

4412. * Shared assignment expression handling * * args = (opcode << 8) + rbp * * If 'opcode' is DUK_OP_NONE, plain assignment without arithmetic. * Syntactically valid left-hand-side forms which are not accepted as * left-hand-side values (e.g. as in "f() = 1") must NOT cause a * SyntaxError, but rather a run-time ReferenceError. ** When evaluating X <op>= Y, the LHS (X) is conceptually evaluated * to a temporary first. The RHS is then evaluated. Finally, the * <op> is applied to the initial value of RHS (not the value after * RHS evaluation), and written to X. Doing so concretely generates * inefficient code so we'd like to avoid the temporary when possible. * See: <https://github.com/svaarala/duktape/pull/992>. ** The expression value (final LHS value, written to RHS) is * conceptually copied into a fresh temporary so that it won't * change even if the LHS/RHS values change in outer expressions. * For example, it'd be generally incorrect for the expression value * to be the RHS register binding, unless there's a guarantee that it * won't change during further expression evaluation. Using the * temporary concretely produces inefficient bytecode, so we try to * avoid the extra temporary for some known-to-be-safe cases. * Currently the only safe case we detect is a "top level assignment", * for example "x = y + z;", where the assignment expression value is * ignored. * See: test-dev-assign Expr.js and test-bug-assign-mutate-gh381.js.

4413. TRYCATCH, cannot emit now (not enough info)

4414. Value stack slot limits: these are quite approximate right now, and * because they overlap in control flow, some could be eliminated.

4415. absolute position for digit considered for rounding

4416. -> [... this timeval_new]

4417. TypeError if wrong; class check, see E5 Section 15.10.6

4418. DUK_HOBJECT_FLAG_EXOTIC_DUKFUNC: omitted here intentionally

4419. * Byte order. Important to self check, because on some exotic platforms * there is no actual detection but rather assumption based on platform * defines.

4420. Both inputs are zero; cases where only one is zero can go * through main algorithm.

4421. same thread

4422. 'Boolean'

4423. * Inside one or more 'with' statements fall back to slow path always. * (See e.g. test-stmt-with.js.)

4424. * Init the heap thread

4425. XXX: need a duk_require_func_or_lfunc_coerce()

4426. avoid referencing, invalidated

4427. Don't support "straddled" source now.

4428. * Object compaction. ** Compaction is assumed to never throw an error.

4429. [arg1 ... argN this loggerLevel loggerName]

4430. duk_tval intentionally skipped

4431. * Since built-ins are not often extended, compact them.

4432. * Compilation

4433. Push a new ArrayBuffer (becomes view .buffer)

4434. idx_start

4435. Note: we rely on the _Varmap having a bunch of nice properties, like: * - being compacted and unmodified during this process * - not containing an array part * - having correct value types

4436. **comment:** XXX: return indication of "terminalness" (e.g. a 'throw' is terminal)
label: code-design

4437. may be NULL if no constants or inner funcs

4438. Equivalent year for DST calculations outside [1970,2038[range, see * E5 Section 15.9.1.8. Equivalent year has the same leap-year-ness and * starts with the same weekday on Jan 1. * https://bugzilla.mozilla.org/show_bug.cgi?id=351066

4439. empty expressions can be detected conveniently with nud/led counts

4440. There are at most 7 args, but we use 8 here so that also * DUK_DATE_IDX_WEEKDAY gets initialized (to zero) to avoid the potential * for any Valgrind gripes later.

4441. [... val callback thisArg val i obj]

4442. [... this tracedata sep this str1 ... strN]

4443. important chosen base types

4444. Digit count indicates number of fractions (i.e. an absolute * digit index instead of a relative one). Used together with * DUK_N2S_FLAG_FIXED_FORMAT for toFixed().

4445. must also check refzero_list

4446. MEMBER/NEW/CALL EXPRESSIONS

4447. **comment:** XXX: add flag to indicate whether caller cares about return value; this * affects e.g. handling of assignment expressions. This change needs API * changes elsewhere too.
label: code-design

4448. Fast jumps (which avoid longjmp) jump directly to the jump sites * which are always known even while the iteration/switch statement * is still being parsed. A final peephole pass "straightens out" * the jumps.

4449. [... func]

4450. * Function gets no new environment when called. This is the * case for global code, indirect eval code, and non-strict * direct eval code. There is no direct correspondence to the * E5 specification, as global/eval code is not exposed as a * function.

4451. Wrappers for calling standard math library methods. These may be required * on platforms where one or more of the math built-ins are defined as macros * or inline functions and are thus not suitable to be used as function pointers.

4452. **comment:** Throwing an error this deep makes the error rather vague, but * saves hundreds of bytes of code.
label: code-design

4453. This is not strictly necessary, but avoids compiler warnings; e.g. * gcc won't reliably detect that no uninitialized data is read below.

4454. Trigger at zero or below

4455. Custom coercion for API

4456. **comment:** Note: cannot read more than 24 bits without possibly shifting top bits out. * Fixable, but adds complexity.
label: code-design

4457. Almost all global level Function objects are constructable * but not all: Function.prototype is a non-constructable, * callable Function.

4458. source ends before dest starts

4459. * Statement terminator check, including automatic semicolon * handling. After this step, 'curr_tok' should be the first * token after a possible statement terminator.

4460. * Exposed string-to-number API * * Input: [string] * Output: [number] * * If number parsing fails, a NaN is pushed as the result. If number parsing * fails due to an internal error, an InternalError is thrown.

4461. 64-bit OK because always >= 0

4462. reg/const for switch value

4463. **comment:** When formatting an argument to a string, errors may happen from multiple * causes. In general we want to catch obvious errors like a toLogString() * throwing an error, but we don't currently try to catch every possible * error. In particular, internal errors (like out of memory or stack) are * not caught. Also, we expect Error.toString() to not throw an error.

label: code-design

4464. Each round of finalizer execution may spawn new finalizable objects * which is normal behavior for some applications. Allow multiple * rounds of finalization, but use a shrinking limit based on the * first round to detect the case where a runaway finalizer creates * an unbounded amount of new finalizable objects. Finalizer rescue * is not supported: the semantics are unclear because most of the * objects being finalized here are already reachable. The finalizer * is given a boolean to indicate that rescue is not possible. ** See discussion in: <https://github.com/svaarala/duktape/pull/473>

4465. no prototype, updated below

4466. unwind to 'resume' caller

4467. 0xf0-0xff

4468. **comment:** XXX: missing trap result validation for non-configurable target keys * (must be present), for non-extensible target all target keys must be * present and no extra keys can be present. * <http://www.ecma-international.org/ecma-262/6.0/#sec-proxy-object-internal-methods-and-internal-slots-ownpropertykeys>

label: code-design

4469. **comment:** NOTE: lightfuncs are coerced to full functions because * lightfuncs don't fit into a property value slot. This * has some side effects, see test-dev-lightfunc-accessor.js.

label: code-design

4470. IdentityEscape, with dollar added as a valid additional * non-standard escape (see test-regexp-identity-escape-dollar.js). * Careful not to match end-of-buffer (<0) here.

4471. 1st related value (type specific)

4472. The caller is responsible for being sure that bytecode being loaded * is valid and trusted. Invalid bytecode can cause memory unsafe * behavior directly during loading or later during bytecode execution * (instruction validation would be quite complex to implement). ** This signature check is the only sanity check for detecting * accidental invalid inputs. The initial 0xFF byte ensures no * ordinary string will be accepted by accident.

4473. Shared object part

4474. always throw ReferenceError for unresolved

4475. * Mark roots, hoping that recursion limit is not normally hit. * If recursion limit is hit, run additional reachability rounds * starting from "temproots" until marking is complete. ** Marking happens in two phases: first we mark actual reachability * roots (and run "temproots" to complete the process). Then we * check which objects are unreachable and are finalizable; such * objects are marked as FINALIZABLE and marked as reachability * (and "temproots" is run again to complete the process). ** The heap finalize_list must also be marked as a reachability root. * There may be objects on the list from a previous round if the * previous run had finalizer skip flag.

4476. Shift to sign extend.

4477. '^'

4478. required_desc_flags

4479. **comment:** Stack indices for better readability

label: code-design

4480. * Try registers

4481. no prototype or class yet

4482. shadowed; update value

4483. E5 Section 8.6.2 + custom classes

4484. **comment:** Lookup current thread; use the stable 'entry_thread' for this to * avoid clobber warnings. Any valid, reachable 'thr' value would be * fine for this, so using 'entry_thread' is just to silence warnings.

label: code-design

4485. RELATIONAL EXPRESSION

4486. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property existence check right now.

4487. These are just convenience "wiping" of state. Side effects should * not be an issue here: thr->heap and thr->heap->jl have a stable * pointer. Finalizer runs etc capture even out-of-memory errors so * nothing should throw here.

4488. Zero escape (also allowed in non-strict mode)

4489. roughly 64 bytes

4490. XXX: this doesn't actually work properly for tail calls, so * tail calls are disabled when DUK_USE_NONSTD_FUNC_CALLER_PROPERTY * is in use.

4491. * Dragon4 slow path (binary) digit generation. * An extra digit is generated for rounding.

4492. this is a bit approximate (errors out before max is reached); this is OK

4493. **comment:** * Parse a function-body-like expression (FunctionBody or Program * in E5 grammar) using a two-pass parse. The productions appear * in the following contexts: * - function expression * - function statement * - function declaration * - getter in object literal * - setter in object literal * - global code * - eval code * - Function constructor body * * This function only parses the statement list of the body; the argument * list and possible function name must be initialized by the caller. * For instance, for Function constructor, the argument names are originally * on the value stack. The parsing of statements ends either at an EOF or * a closing brace; this is controlled by an input flag. ** Note that there are many differences affecting parsing and even code * generation: * - Global and eval code have an implicit return value generated * by the last statement; function code does not * - Global code, eval code, and Function constructor body end in * an EOF, other bodies in a closing brace ('}') * * Upon entry, 'curr_tok' is ignored and the function will pull in the * first token on its own. Upon exit, 'curr_tok' is the terminating * token (EOF or closing brace).

label: code-design

4494. When torture not enabled, can just use the same helper because * 'reg' won't get spilled.

4495. Note: actual update happens once write has been completed * without error below. The write should always succeed * from a specification viewpoint, but we may e.g. run out * of memory. It's safer in this order.

4496. Convert buffer to result string.

4497. default prototype (Note: 'thr' must be reachable)

4498. we know these because enum objects are internally created

4499. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).

label: code-design

4500. **comment:** Direct eval requires that there's a current * activation and it is an Ecmascript function. * When Eval is executed from e.g. cooperate API * call we'll need to do an indirect eval instead.

label: code-design

4501. 1 = num actual 'return values'

4502. should never happen

4503. '-' verbatim

4504. Regexp tokens

4505. avoid attempt to compact the current object

4506. DUK_TOK_TRUE

4507. idx_bottom and idx_retval are only used for book-keeping of * Ecmascript-initiated calls, to allow returning to an Ecmascript * function properly. They are duk_size_t to match the convention * that value stack sizes are duk_size_t and local frame indices * are duk_idx_t.

4508. **comment:** This loop is optimized for size. For speed, there should be * two separate loops, and we should ensure that memcmp() can be * used without an extra "will searchstring fit" check. Doing * the preconditioning for 'p' and 'p_end' is easy but cpos * must be updated if 'p' is wound back (backward scanning).

label: code-design

4509. min(incl)

4510. may be equal

4511. traceback depth ensures fits into 16 bits

4512. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack. ** The right hand side could be a light function (as they generally * behave like objects). Light functions never have a 'prototype' * property so E5.1 Section 15.3.5.3 step 3 always throws a TypeError. * Using duk_require_hobject() is thus correct (except for error msg).

4513. DUK_TOK_GET

4514. * Various

4515. Module has now evaluated to a wrapped module function. Force its `* .name` to match `module.name` (defaults to last component of resolved `* ID`) so that it is shown in stack traces too. Note that we must not `*` introduce an actual name binding into the function scope (which is `*` usually the case with a named function) because it would affect the `*` scope seen by the module and shadow accesses to globals of the same name. `*` This is now done by compiling the function as anonymous and then forcing `* its .name` without setting a "has name binding" flag.

4516. Characters outside BMP cannot be escape()'d. We could `*` encode them as surrogate pairs (for codepoints inside `*` valid UTF-8 range, but not extended UTF-8). Because `* escape()` and `unescape()` are legacy functions, we don't.

4517. trailer

4518. `*` Check whether we need to abandon an array part (if it exists)

4519. [... env callee]

4520. **comment:** These is not 100% because format would need to be non-portable "long long". `*` Also print out as doubles to catch cases where the "long" type is not wide `*` enough; the limits will then not be printed accurately but the magnitude `*` will be correct.
label: code-design

4521. an Ecmascript function

4522. **comment:** avoid pressure to add/remove strings, invalidation of call data argument, etc.
label: code-design

4523. end of bytecode

4524. key is 'length', cannot match argument exotic behavior

4525. `*` First check whether property exists; if not, simple case. This covers `*` steps 1-4.

4526. `*` Do-while statement is mostly trivial, but there is special `*` handling for automatic semicolon handling (triggered by the `*` `DUK__ALLOW_AUTO_SEMI_ALWAYS` flag related to a bug filed at: `** https://bugs.ecmascript.org/show_bug.cgi?id=8 **` See doc/compiler.rst for details.

4527. must be found: was found earlier, and cannot be inherited

4528. Input value should be on stack top and will be coerced and `*` popped. Refuse to update an Array's 'length' to a value `*` outside the 32-bit range. Negative zero is accepted as zero.

4529. shift in zeroes

4530. free object and all auxiliary (non-heap) allocs

4531. Useful for internal call sites where we either expect an object (function) `*` or a lightfunc. Returns NULL for a lightfunc.

4532. bytes in dest

4533. `stack[0] = regexp * stack[1] = string`

4534. flags

4535. Check for maximum buffer length.

4536. 'Thread'

4537. **comment:** Pointer to bytecode executor's 'curr_pc' variable. Used to copy `*` the current PC back into the topmost activation when activation `*` state is about to change (or "syncing" is otherwise needed). This `*` is rather awkward but important for performance, see execution.rst.
label: code-design

4538. The `Quote(value)` operation: quote a string. `** Stack policy: [] -> [].`

4539. Output #2: last component name

4540. `tc1 = false, tc2 = false`

4541. 0

4542. Exponent without a sign or with a +/- sign is accepted `*` by all call sites (even `JSON.parse()`).

4543. `0x90-0x9f`

4544. **comment:** XXX: Shuffling support could be implemented here so that `LDINT+LDINTX` `*` would only shuffle once (instead of twice). The current code works `*` though, and has a smaller compiler footprint.
label: code-design

4545. `*` Write to entry part

4546. To use the shared helper need the virtual index.

4547. `*` Slow delete, but we don't care as we're already in a very slow path. `*` The delete always succeeds: key has no exotic behavior, property `*` is configurable, and no resize occurs.

4548. LAYOUT 1

4549. For `len == 0`, `i` is initialized to `len - 1` which underflows. `*` The condition (`i < len`) will then exit the for-loop on the `*` first round which is correct. Similarly, loop termination `*` happens by `i` underflowing.

4550. Raw internal valstack access macros: access is unsafe so call site `*` must have a guarantee that the index is valid. When that is the case, `*` using these macro results in faster and smaller code than `duk_get_tval()`. `*` Both 'ctx' and 'idx' are evaluated multiple times, but only for asserts.

4551. `*` Node.js Buffer.isEncoding()

4552. array entries are all plain values

4553. **comment:** Buffer values are encoded in (lowercase) hex to make the `*` binary data readable. Base64 or similar would be more `*` compact but less readable, and the point of JX/JC `*` variants is to be as useful to a programmer as possible.
label: code-design

4554. **comment:** XXX: the helper currently assumes stack top contains new `*` 'length' value and the whole calling convention is not very `*` compatible with what we need.
label: code-design

4555. Coerce to number before validating pointers etc so that the `*` number coercions in `duk_hbufferobject_validated_write()` are `*` guaranteed to be side effect free and not invalidate the `*` pointer checks we do here.

4556. Mark-and-sweep interval is relative to combined count of objects and `*` strings kept in the heap during the latest mark-and-sweep pass. `*` Fixed point .8 multiplier and .0 adder. Trigger count (interval) is `*` decreased by each (re)allocation attempt (regardless of size), and each `*` refzero processed object. `*` 'SKIP' indicates how many (re)allocations to wait until a retry if `*` GC is skipped because there is no thread do it with yet (happens `*` only during init phases).

4557. XXX: assert 'c' is an enumerator

4558. two special escapes: '\n' and "", other printables as is

4559. for `indexOf`, `ToInteger(undefined)` would be 0, i.e. correct, but `*` handle both `indexOf` and `lastIndexOf` specially here.

4560. `DUK_USE_AUGMENT_ERROR_THROW`

4561. `*` Exponent notation for non-base-10 numbers isn't specified in Ecmascript `*` specification, as it never explicitly turns up: non-decimal numbers can `*` only be formatted with `Number.prototype.toString([radix])` and for that, `*` behavior is not explicitly specified. `**` Logical choices include formatting the exponent as decimal (e.g. binary `* 100000` as `1e+5`) or in current radix (e.g. binary `100000` as `1e+101`). `*` The Dragon4 algorithm (in the original paper) prints the exponent value `*` in the target radix B. However, for radix values 15 and above, the `*` exponent separator 'e' is no longer easily parseable. Consider, for `*` instance, the number `"1.faecee+1c"`.

4562. Template functions are not strictly constructable (they don't `*` have a "prototype" property for instance), so leave the `*` `DUK_HOBJECT_FLAG_CONSTRUCTABLE` flag cleared here.

4563. **comment:** Most practical strings will go here.
label: code-design

4564. non-strict eval: env is caller's env or global env (direct vs. indirect call) `*` global code: env is is global env

4565. **comment:** XXX: use the exactly same arithmetic function here as in executor
label: code-design

4566. `-> [... lval new_rval]`

4567. sep (even before first one)

4568. **comment:** XXX: use advancing pointers instead of index macros `->` faster and smaller?
label: code-design

4569. `0x7F` is special

4570. IEEE requires that zeros compare the same regardless `*` of their signed, so if both x and y are zeroes, they `*` are caught above.

4571. Signed integers always map to 4 bytes now.

4572. Resolve Proxy targets until Proxy chain ends. No explicit check for * a Proxy loop: user code cannot create such a loop without tweaking * internal properties directly.

4573. estimate is valid

4574. Note: it's nice if size is 2^N (not $4 \times 4 = 16$ bytes on 32 bit)

4575. always true, arg is unsigned

4576. ArrayBuffer: unlike any other argument variant, create * a view into the existing buffer.

4577. DUK_TOK_DO

4578. * Slow path: potentially requires function calls for coercion

4579. 'ArrayBuffer'

4580. Initialize index so that we skip internal control keys.

4581. flags have been already cleared

4582. * ArrayBuffer, DataView, and TypedArray constructors

4583. **comment:** * XXX: for now, indicate that an expensive catch binding * declarative environment is always needed. If we don't * need it, we don't need the const_varname either.
label: code-design

4584. -> [... this timeval_new timeval_new]

4585. DUK_TOK_CONST

4586. **comment:** XXX: unnecessary copying of values? Just set 'top' to * b + c, and let the return handling fix up the stack frame?
label: code-design

4587. Slot C

4588. **comment:** Array is dense and contains only strings, but ASIZE may * be larger than used part and there are UNUSED entries.
label: code-design

4589. At least: ... [err]

4590. wrapped

4591. whole or fraction digit

4592. * Constructor calls

4593. only 'length'

4594. DUK_USE_DEBUGGER_DUMPHEAP

4595. 'toISOString'

4596. 'res' contains expression value

4597. No other escape beginning with a digit in strict mode

4598. **comment:** Use Murmurhash2 directly for short strings, and use "block skipping" * for long strings: hash an initial part and then sample the rest of * the string with reasonably sized chunks. An initial offset for the * sampling is computed based on a hash of the initial part of the string; * this is done to (usually) avoid the case where all long strings have * certain offset ranges which are never sampled. * * Skip should depend on length and bound the total time to roughly * logarithmic. With current values: * * 1M string => 256 * 241 = 61696 bytes (0.06M) of hashing * 1G string => 256 * 16321 = 4178176 bytes (3.98M) of hashing * * XXX: It would be better to compute the skip offset more "smoothly" * instead of having a few boundary values.
label: code-design

4599. [key setter this val key] -> [key retval]

4600. point to start of 'reserved area'

4601. Shared helper for match() steps 3-4, search() steps 3-4.

4602. dummy

4603. Just call the "original" Object.defineProperties() to * finish up.

4604. Although these could be parsed as PatternCharacters unambiguously (here), * E5 Section 15.10.1 grammar explicitly forbids these as PatternCharacters.

4605. line terminator will be handled on next round

4606. * Debug dumping

4607. x in [2**31, 2**32[

4608. Error path.

4609. if a site already exists, nop: max one label site per statement

4610. Flag ORed to err_code to indicate __FILE__ / __LINE__ is not * blamed as source of error for error fileName / lineNumber.

4611. Start in paused state.

4612. **comment:** XXX: much to improve (code size)
label: code-design

4613. [regexp string res_arr]

4614. range conversion with a "skip"

4615. **comment:** Update function min/max line from current token. Needed to improve * function line range information for debugging, so that e.g. opening * curly brace is covered by line range even when no opcodes are emitted * for the line containing the brace.
label: code-design

4616. * Regular expression structs, constants, and bytecode defines.

4617. **comment:** should never be zero, because we (Duktape.Thread.yield) are on the stack
label: code-design

4618. **comment:** XXX: This will now return false for non-numbers, even though they would * coerce to NaN (as a general rule). In particular, duk_get_number() * returns a NaN for non-numbers, so should this function also return * true for non-numbers?
label: code-design

4619. **comment:** The handler is looked up with a normal property lookup; it may be an * accessor or the handler object itself may be a proxy object. If the * handler is a proxy, we need to extend the valstack as we make a * recursive proxy check without a function call in between (in fact * there is no limit to the potential recursion here). * * (For sanity, proxy creation rejects another proxy object as either * the handler or the target at the moment so recursive proxy cases * are not realized now.)
label: code-design

4620. Flag handling currently assumes that flags are consistent. This is OK * because the call sites are now strictly controlled.

4621. DUK_TOK_DEFAULT

4622. [key value] or [key undefined]

4623. Pad significand with "virtual" zero digits so that Dragon4 will * have enough (apparent) precision to work with.

4624. * Set lexer input position and reinitialize lookup window.

4625. nargs

4626. With lightfuncs, act 'func' may be NULL

4627. return '(?:)'

4628. DUK_USE_REFZERO_FINALIZER_TORTURE

4629. next to use, highest used is top - 1

4630. **comment:** XXX: actually single step levels would work just fine, clean up
label: code-design

4631. * DUK_CALL_FLAG_IGNORE_RECLIMIT causes duk_handle_call() to ignore C * recursion depth limit (and won't increase it either). This is * dangerous, but useful because it allows the error handler to run * even if the original error is caused by C recursion depth limit. * * The heap level DUK_HEAP_FLAG_ERRHANDLER_RUNNING is set for the * duration of the error handler and cleared afterwards. This flag * prevents the error handler from running recursively. The flag is * heap level so that the flag properly controls even coroutines * launched by an error handler. Since the flag is heap level, it is * critical to restore it correctly. * * We ignore errors now: a success return and an error value both * replace the original error value. (This would be easy to change.)

4632. **comment:** XXX: remove heap->dbg_exec_counter, use heap->inst_count_interrupt instead?
label: code-design

4633. DUK_USE_JSON_DECSTRING_FASTPATH

4634. The actual detached_cb call is postponed to message loop so * we don't need any special precautions here (just skip to EOM * on the already closed connection).
4635. arg2 would be clobbered so reassign it to a temp.
4636. standard behavior, step 3.f.i
4637. avoid degenerate cases, so that (len - 1) won't underflow
4638. Note: %06d for positive value, %07d for negative value to include * sign and 6 digits.
4639. 'let'
4640. * Halt execution helper
4641. 1110 xxxx; 3 bytes
4642. ch1 = (r_increment << 8) + byte
4643. variable name reg/const, if variable not register-bound
4644. Cast converts magic to 16-bit signed value
4645. * Remove the object from the refzero list. This cannot be done * before a possible finalizer has been executed; the finalizer * may trigger a mark-and-sweep, and mark-and-sweep must be able * to traverse a complete refzero_list.
4646. Argument variants. When the argument is an ArrayBuffer a view to * the same buffer is created; otherwise a new ArrayBuffer is always * created.
4647. entry part size
4648. * Property not found in prototype chain.
4649. function executes in strict mode
4650. 'data' is reachable through every compiled function which * contains a reference.
4651. Caller must trigger recomputation of active breakpoint list. To * ensure stale values are not used if that doesn't happen, clear the * active breakpoint list here.
4652. * Longjmp types, also double as identifying continuation type for a rethrow (in 'finally')
4653. note: long live range
4654. * Object will be kept; queue object back to heap_allocated (to tail)
4655. no change
4656. DUK_USE_ASSERTIONS
4657. force the property to 'undefined' to create a slot for it
4658. Optimized for speed.
4659. **comment:** prevent multiple in-progress detaches
label: requirement
4660. Set a high interrupt counter; the original executor * interrupt invocation will rewrite before exiting.
4661. DUK_TOK_THIS
4662. index
4663. * Expression parsing: duk__expr_nud(), duk__expr_led(), duk__expr_lbp(), and helpers. * * - duk__expr_nud(): ("null denotation"): process prev_token as a "start" of an expression (e.g. literal) * - duk__expr_led(): ("left denotation"): process prev_token in the "middle" of an expression (e.g. operator) * - duk__expr_lbp(): ("left-binding power"): return left-binding power of curr_token
4664. 3-letter log level strings
4665. **comment:** * Convert char offset to byte offset * * Avoid using the string cache if possible: for ASCII strings byte and * char offsets are equal and for short strings direct scanning may be * better than using the string cache (which may evict a more important * entry). * * Typing now assumes 32-bit string byte/char offsets (duk_uint_fast32_t). * Better typing might be to use duk_size_t.
label: code-design
4666. Must prevent finalizers which may have arbitrary side effects.
4667. DUK_EXCEPTION_H_INCLUDED
4668. invalidates h_buf pointer
4669. Special case: original qmin was zero so there is nothing * to repeat. Emit an atom copy but jump over it here.
4670. **comment:** Higher footprint, less churn.
label: code-design
4671. -> [... holder name val new_elem]
4672. NUL term or -1 (EOF), NUL check would suffice
4673. core property functions
4674. 25%, i.e. less than 25% used -> abandon
4675. Allow empty fraction (e.g. "123.")
4676. 0xff0...0xff
4677. don't allow const
4678. this is just beneath bottom
4679. * Duktape.Buffer: constructor
4680. 'Uint8Array'
4681. * Helpers for writing multiple properties
4682. exhaustive
4683. refcounting requires direct heap frees, which in turn requires a dual linked heap
4684. lastIndexOf() needs to be a vararg function because we must distinguish * between an undefined fromIndex and a "not given" fromIndex; indexOf() is * made vararg for symmetry although it doesn't strictly need to be.
4685. [... error]
4686. XXX: share final setting code for value and flags? difficult because * refcount code is different. Share entry allocation? But can't allocate * until array index checked.
4687. Terminating conditions. For fixed width output, we just ignore the * terminating conditions (and pretend that tc1 == tc2 == false). The * the current shortcut for fixed-format output is to generate a few * extra digits and use rounding (with carry) to finish the output.
4688. Return value when returning to this activation (points to caller * reg, not callee reg); index is absolute (only set if activation is * not topmost). * * Note: idx_bottom is always set, while idx_retval is only applicable * for activations below the topmost one. Currently idx_retval for * the topmost activation is considered garbage (and it not initialized * on entry or cleared on return; may contain previous or garbage * values).
4689. Unlike non-obsolete String calls, substr() algorithm in E5.1 * specification will happily coerce undefined and null to strings * ("undefined" and "null").
4690. DecimalEscape, only \0 is allowed, no leading zeroes are allowed
4691. parse new token
4692. 1e9
4693. Negative value checked so that a "time jump" works * reasonably. * * Same interval is now used for status sending and * peeking.
4694. * Misc shared helpers.
4695. don't set 'n' at all, inherited value is used as name
4696. '
4697. switch cmd
4698. **comment:** XXX: could add a fast path to process chunks of input codepoints, * but relative benefit would be quite small.
label: code-design
4699. Copy interrupt counter/init value state to new thread (if any). * It's OK for new_thr to be the same as curr_thr.
4700. User totalLength overrides a computed length, but we'll check * every copy in the copy loop. Note that duk_to_uint() can * technically have arbitrary side effects so we need to recheck * the buffers in the copy loop.
4701. **comment:** XXX: some overlapping code; cleanup
label: code-design
4702. stack[0] = searchElement * stack[1] = fromIndex * stack[2] = object * stack[3] = length (not needed, but not popped above)
4703. from curr pc
4704. * Maximum size check
4705. isError flag for yield

4706. * callstack_top - 1 --> this function * callstack_top - 2 --> caller (may not exist) ** If called directly from C, callstack_top might be 1. If calling * activation doesn't exist, call must be indirect.

4707. no need to unwind callstack

4708. * Array part

4709. **comment:** * Number should already be in NaN-normalized form, * but let's normalize anyway.
label: code-design

4710. [array totalLength bufres buf]

4711. When src_size == 0, src_data may be NULL (if source * buffer is dynamic), and dst_data may be NULL (if * target buffer is dynamic). Avoid zero-size memcpy() * with an invalid pointer.

4712. [r1,r2] is the range

4713. * NormalizePropertyDescriptor() related helper. ** Internal helper which validates and normalizes a property descriptor * represented as an Ecmascript object (e.g. argument to defineProperty()). * The output of this conversion is a set of defprop_flags and possibly * some values pushed on the value stack; some subset of: property value, * getter, setter. Caller must manage stack top carefully because the * number of values pushed depends on the input property descriptor. ** The original descriptor object must not be altered in the process.

4714. duk_handle_call / duk_handle_safe_call recursion depth limiting

4715. DUK_HOBJECT_FLAG_NEWENV: handled below

4716. A leading digit is not required in some cases, e.g. accept ".123". * In other cases (JSON.parse()) a leading digit is required. This * is checked for after the loop.

4717. common case, already closed, so skip

4718. ignore millisecond fractions after 3

4719. * unshift()

4720. E5 Section 10.4.3

4721. No built-in functions are constructable except the top * level ones (Number, etc).

4722. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed, * and we're in an infinite loop.

4723. Raw helper for getting a value from the stack, checking its tag. * The tag cannot be a number because numbers don't have an internal * tag in the packed representation.

4724. Bernstein hash init value is normally 5381

4725. * DECLVAR ** See E5 Sections: * 10.4.3 Entering Function Code * 10.5 Declaration Binding Instantiation * 12.2 Variable Statement * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution ** Variable declaration behavior is mainly discussed in Section 10.5, * and is not discussed in the execution semantics (Sections 11-13). ** Conceptually declarations happen when code (global, eval, function) * is entered, before any user code is executed. In practice, register- * bound identifiers are 'declared' automatically (by virtue of being * allocated to registers with the initial value 'undefined'). Other * identifiers are declared in the function prologue with this primitive. * * Since non-register bindings eventually back to an internal object's * properties, the 'prop_flags' argument is used to specify binding * type: * * - Immutable binding: set DUK_PROPDESC_FLAG_WRITABLE to false * - Non-deletable binding: set DUK_PROPDESC_FLAG_CONFIGURABLE to false * - The flag DUK_PROPDESC_FLAG_ENUMERABLE should be set, although it * doesn't really matter for internal objects * * All bindings are non-deletable mutable bindings except: * * - Declarations in eval code (mutable, deletable) * - 'arguments' binding in strict function code (immutable) * - Function name binding of a function expression (immutable) ** Declarations may go to declarative environment records (always * so for functions), but may also go to object environment records * (e.g. global code). The global object environment has special * behavior when re-declaring a function (but not a variable); see * E5.1 specification, Section 10.5, step 5.e. ** Declarations always go to the 'top-most' environment record, i.e. * we never check the record chain. It's not an error even if a * property (even an immutable or non-deletable one) of the same name * already exists. ** If a declared variable already exists, its value needs to be updated * (if possible). Returns 1 if a PUTVAR needs to be done by the caller; * otherwise returns 0.

4726. **comment:** duk_handle_call_xxx: call ignores C recursion limit (for errhandler calls)
label: code-design

4727. expt 0xffff is infinite/NaN

4728. Augment the error if called as a normal function. __FILE__ and __LINE__ * are not desirable in this case.

4729. %s

4730. e.g. "x" < "xx"

4731. **comment:** XXX: for fastints, could use a variant which assumes a double duk_tval * (and doesn't need to check for fastint again).
label: code-design

4732. SameValue

4733. How much stack to require on entry to object/array encode

4734. * Parse code after the clause. Possible terminators are * 'case', 'default', and '}'. ** Note that there may be no code at all, not even an empty statement, * between case clauses. This must be handled just like an empty statement * (omitting seemingly pointless JUMPs), to avoid situations like * test-bug-case-fallthrough.js.

4735. ... name ': ' message

4736. to exit

4737. parenthesis count, 0 = top level

4738. * Useful Unicode codepoints * * Integer constants must be signed to avoid unexpected coercions * in comparisons.

4739. input string (may be a user pointer)

4740. * round_idx points to the digit which is considered for rounding; the * digit to its left is the final digit of the rounded value. If round_idx * is zero, rounding will be performed; the result will either be an empty * rounded value or if carry happens a '1' digit is generated.

4741. Careful with carry condition: * - If carry not added: 0x12345678 + 0 + 0xffffffff = 0x12345678 (< 0x12345678) * - If carry added: 0x12345678 + 1 + 0xffffffff = 0x12345678 (== 0x12345678)

4742. shared helper

4743. * Fast buffer writer with spare management.

4744. success/error path both do this

4745. DUK_TOK_MUL

4746. attempt to change from accessor to data property

4747. allowed ascii whitespace

4748. [... source? filename?]

4749. continue jump

4750. -> [... key val]

4751. Prepare value stack for a method call through an object property. * May currently throw an error e.g. when getting the property.

4752. no need to create environment record now; leave as NULL

4753. * Thread state check and book-keeping.

4754. "/=" and not in regexp mode

4755. for lastIndexOf, result may be -1 (mark immediate termination)

4756. fatal_func should be noreturn, but noreturn declarations on function * pointers has a very spotty support apparently so it's not currently * done.

4757. requested identifier

4758. Function expression. Note that any statement beginning with 'function' * is handled by the statement parser as a function declaration, or a * non-standard function expression/statement (or a SyntaxError). We only * handle actual function expressions (occurring inside an expression) here. ** O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().

4759. * JSON.stringify() fast path * * Otherwise supports full JSON, JX, and JC features, but bails out on any * possible side effect which might change the value being serialized. The * fast path can take advantage of the fact that the value being serialized * is unchanged so that we can walk directly through property tables etc.

4760. **comment:** XXX: helper
label: code-design

4761. no return value -> don't replace created value

4762. 'Arguments' object and has arguments exotic behavior (non-strict callee)

4763. [requested_id require require.id resolved_id last_comp Duktape.modLoaded Duktape.modLoaded[id]]

4764. Note: left shift, should mask

4765. 31 to 36
4766. two level break
4767. don't allow an empty match at the end of the string
4768. 'thr' is now reachable
4769. Parse argument list. Arguments are written to temps starting from * "next temp". Returns number of arguments parsed. Expects left paren * to be already eaten, and eats the right paren before returning.
4770. Allow 'Infinity'
4771. Opcode slot C is used in a non-standard way, so shuffling * is not allowed.
4772. * Helpers for managing property storage size
4773. if-digit-else-ctrl
4774. Safe write calls which will ensure space first.
4775. having this as a separate function provided a size benefit
4776. code emission
4777. **comment:** * obj->props is intentionally left as NULL, and duk_hobject_props.c must deal * with this properly. This is intentional: empty objects consume a minimum * amount of memory. Further, an initial allocation might fail and cause * 'obj' to "leak" (require a mark-and-sweep) since it is not reachable yet.
label: code-design
4778. Term was '.', backtrack resolved name by one component. * q[-1] = previous slash (or beyond start of buffer) * q[-2] = last char of previous component (or beyond start of buffer)
4779. **comment:** This could also be thrown internally (set the error, goto check_longjmp), * but it's better for internal errors to bubble outwards so that we won't * infinite loop in this catchpoint.
label: code-design
4780. * Debugging related API calls
4781. -> [... func this arg1 ... argN _Args length]
4782. 'Float64Array'
4783. 1 1 1 <32 bits> * Encode in two parts to avoid bitencode 24-bit limitation
4784. -> [this]
4785. **comment:** Note: array entries are always writable, so the writability check * above is pointless for them. The check could be avoided with some * refactoring but is probably not worth it.
label: code-design
4786. retval indicates delete failed
4787. formatters always get one-based month/day-of-month
4788. continue matching, set neg_tzoffset flag
4789. bound function chain has already been resolved
4790. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property get right now.
4791. standard JSON; array gaps
4792. Note: array part values are [[Writable]], [[Enumerable]], * and [[Configurable]] which matches the required attributes * here.
4793. 0=before period/exp, * 1=after period, before exp * 2=after exp, allow '+' or '-' * 3=after exp and exp sign
4794. just check 'env'
4795. **comment:** XXX: primitive to make array from valstack slice
label: code-design
4796. **comment:** * Dragon4 setup. * * Convert double from IEEE representation for conversion; * normal finite values have an implicit leading 1-bit. The * slow path algorithm doesn't handle zero, so zero is special * cased here but still creates a valid nc_ctx, and goes * through normal formatting in case special formatting has * been requested (e.g. forced exponential format: 0 -> "0e+0").
label: code-design
4797. consistency requires this
4798. DUK_TOK_CASE
4799. Nothing to initialize, strsf[] is in ROM.
4800. currently 6 characters of lookup are actually needed (duk_lexer.c)
4801. index/length check guarantees
4802. left.x1 -> res.x1
4803. * Case conversion tables generated using src/extract_caseconv.py.
4804. is global code
4805. **comment:** * Macros to set a duk_tval and update refcount of the target (decref the * old value and incref the new value if necessary). This is both performance * and footprint critical; any changes made should be measured for size/speed.
label: code-design
4806. We don't duk_require_stack() here now, but rely on the caller having * enough space.
4807. * Regexp executor. * * Safety: the EcmaScript executor should prevent user from reading and * replacing regexp bytecode. Even so, the executor must validate all * memory accesses etc. When an invalid access is detected (e.g. a 'save' * opcode to invalid, unallocated index) it should fail with an internal * error but not cause a segmentation fault. * * Notes: * * - Backtrack counts are limited to unsigned 32 bits but should * technically be duk_size_t for strings longer than 4G chars. * This also requires a regexp bytecode change.
4808. Note: not a valid stack index if num_stack_args == 0
4809. * Convenience (independent of representation)
4810. no refcounting
4811. duk_handle_ecma_call_setup: setup for a resume()
4812. escape any non-ASCII characters
4813. if 0, 'str' used, if > 0, 'strlist' used
4814. These limits are based on bytecode limits. Max temps is limited * by duk_hcompiledfunction nargs/nregs fields being 16 bits.
4815. * pop(), push()
4816. 'DecEnv'
4817. number of elements guaranteed to be user accessible * (in addition to call arguments) on Duktape/C function entry.
4818. NOTE: The Array special behaviors are NOT invoked by duk_xdef_prop_index() * (which differs from the official algorithm). If no error is thrown, this * doesn't matter as the length is updated at the end. However, if an error * is thrown, the length will be unset. That shouldn't matter because the * caller won't get a reference to the intermediate value.
4819. Torture option to shake out finalizer side effect issues: * make a bogus function call for every finalizable object, * essentially simulating the case where everything has a * finalizer.
4820. [... key]
4821. **comment:** This seems faster than emitting bytes one at a time and * then potentially rewinding.
label: code-design
4822. * Writability check
4823. may be changed by call
4824. DUK_USE_NONSTD_FUNC_CALLER_PROPERTY
4825. DUK_USE_JSON_STRINGIFY_FASTPATH
4826. Update all labels with matching label_id.
4827. guaranteed by duk_to_string()
4828. already declared, must update binding value
4829. DUK_USE_TRACEBACKS
4830. **comment:** Formatting function pointers is tricky: there is no standard pointer for * function pointers and the size of a function pointer may depend on the * specific pointer type. This helper formats a function pointer based on * its memory layout to get something useful on most platforms.

label: code-design

4831. ensures callstack_top - 1 >= 0
4832. Count free operations toward triggering a GC but never actually trigger * a GC from a free. Otherwise code which frees internal structures would * need to put in NULLs at every turn to ensure the object is always in * consistent state for a mark-and-sweep.
4833. detached
4834. 'constructor'
4835. disabled for now
4836. idx_replacer
4837. type to represent a straight register reference, with <0 indicating none
4838. continue jump not patched, an INVALID opcode remains there
4839. -> [... enum_target res trap_result]
4840. current (next) array index
4841. fmin() with args -0 and +0 is not guaranteed to return * -0 as Ecmascript requires.
4842. 'const'
4843. all codepoints up to U+10FFFF
4844. **comment:** This is performance critical because it's needed for every DECREF. * Take advantage of the fact that the first heap allocated tag is 8, * so that bit 3 is set for all heap allocated tags (and never set for * non-heap-allocated tags).
label: code-design
4845. if no NaN handling flag, may still be NaN here, but not Inf
4846. thr argument only used for thr->heap, so specific thread doesn't matter
4847. Preshifted << 4. Must use 16-bit entry to allow negative value signaling.
4848. [0x00000000, 0xffffffff]
4849. fall through to error
4850. [... key_obj key key]
4851. Specification stripPrefix maps to DUK_S2N_FLAG_ALLOW_AUTO_HEX_INT. * * Don't autodetect octals (from leading zeroes), require user code to * provide an explicit radix 8 for parsing octal. See write-up from Mozilla: * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt#ECMAScript_5_Removes_Octal_Interpretation
4852. Process messages until we're no longer paused or we peek * and see there's nothing to read right now.
4853. XXX: callstack unwind may now throw an error when closing * scopes; this is a sandboxing issue, described in: * <https://github.com/svaarala/duktape/issues/476>
4854. since no recursive error handler calls
4855. **comment:** XXX: other properties of function instances; 'arguments', 'caller'.
label: code-design
4856. numeric label_id (-1 reserved as marker)
4857. irrelevant when out->value == NULL
4858. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed.
4859. **comment:** * Helper to sort array index keys. The keys are in the enumeration object * entry part, starting from DUK__ENUM_START_INDEX, and the entry part is dense. * * We use insertion sort because it is simple (leading to compact code,) * works nicely in-place, and minimizes operations if data is already sorted * or nearly sorted (which is a very common case here). It also minimizes * the use of element comparisons in general. This is nice because element * comparisons here involve re-parsing the string keys into numbers each * time, which is naturally very expensive. * * Note that the entry part values are all "true", e.g. * * "1" -> true, "3" -> true, "2" -> true * * so it suffices to only work in the key part without exchanging any keys, * simplifying the sort. * * http://en.wikipedia.org/wiki/Insertion_sort * * (Compiles to about 160 bytes now as a stand-alone function.)
label: code-design
4860. The function object is now reachable and refcounts are fine, * so we can pop off all the temporaries.
4861. Care must be taken to avoid pointer wrapping in the index * validation. For instance, on a 32-bit platform with 8-byte * duk_tval the index 0x20000000UL would wrap the memory space * once.
4862. Convert a duk_tval fastint (caller checks) to a 32-bit index.
4863. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2016 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
4864. Insert bytes in the middle of the buffer from a slice already * in the buffer. Source offset is interpreted "before" the operation.
4865. DUK_USE_STRTAB_CHAIN
4866. Original function instance/template had NAMEBINDING. * Must create a lexical environment on loading to allow * recursive functions like 'function foo() { foo(); }'.
4867. **comment:** the message should be a compile time constant without formatting (less risk); * we don't care about assertion text size because they're not used in production * builds.
label: code-design
4868. assume value is a number
4869. **comment:** combines steps 3, 6; step 7 is not needed
label: requirement
4870. * Debugger support
4871. **comment:** XXX: spare handling, slow now
label: code-design
4872. normal and constructor calls have identical semantics
4873. * A token is interpreted as any possible production of InputElementDiv * and InputElementRegExp, see E5 Section 7 in its entirety. Note that * the E5 "Token" production does not cover all actual tokens of the * language (which is explicitly stated in the specification, Section 7.5). * Null and boolean literals are defined as part of both ReservedWord * (E5 Section 7.6.1) and Literal (E5 Section 7.8) productions. Here, * null and boolean values have literal tokens, and are not reserved * words. * * Decimal literal negative/positive sign is -not- part of DUK_TOK_NUMBER. * The number tokens always have a non-negative value. The unary minus * operator in "-1.0" is optimized during compilation to yield a single * negative constant. * * Token numbering is free except that reserved words are required to be * in a continuous range and in a particular order. See genstrings.py.
4874. Same coercion behavior as for Number.
4875. the outer loop will recheck and exit
4876. **comment:** These are macros for now, but could be separate functions to reduce code * footprint (check call site count before refactoring).
label: code-design
4877. x == -Infinity
4878. [... source? func_template]
4879. other call
4880. lastIndex already set up for next match
4881. * Macros to access the 'props' allocation.
4882. **comment:** * Struct size/alignment if platform requires it * * There are some compiler specific struct padding pragmas etc in use, this * selftest ensures they're correctly detected and used.
label: code-design
4883. [... source? filename?] (depends on flags)

4884. ToUint16() coercion is mandatory in the E5.1 specification, but * this non-compliant behavior makes more sense because we support * non-BMP codepoints. Don't use CESU-8 because that'd create * surrogate pairs.

4885. match string

4886. [targetBuffer targetStart sourceStart sourceEnd]

4887. x = sign(x) * floor(abs(x)), i.e. truncate towards zero, keep sign

4888. **comment:** XXX: here again finalizer thread is the heap_thread which needs * to be coordinated with finalizer thread fixes.
label: code-design

4889. at least one opcode emitted

4890. Slice offsets are element (not byte) offsets, which only matters * for TypedArray views, Node.js Buffer and ArrayBuffer have shift * zero so byte and element offsets are the same. Negative indices * are counted from end of slice, crossed indices are allowed (and * result in zero length result), and final values are clamped * against the current slice. There's intentionally no check * against the underlying buffer here.

4891. side effects, in theory (referenced by global env)

4892. * Object.seal() and Object.freeze() (E5 Sections 15.2.3.8 and 15.2.3.9) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: virtual (non-concrete) properties which are non-configurable but * writable would pose some problems, but such properties do not currently * exist (all virtual properties are non-configurable and non-writable). * If they did exist, the non-configurability does NOT prevent them from * becoming non-writable. However, this change should be recorded somehow * so that it would turn up (e.g. when getting the property descriptor), * requiring some additional flags in the object.

4893. additional flags which are passed in the same flags argument as property * flags but are not stored in object properties.

4894. **comment:** XXX: Is this INCREF necessary? 'func' is always a borrowed * reference reachable through the value stack? If changed, stack * unwind code also needs to be fixed to match.
label: code-design

4895. **comment:** XXX: shared decoding of 'b' and 'c'?
label: code-design

4896. **comment:** The fast path for array property put is not fully compliant: * If one places conflicting number-indexed properties into * Array.prototype (for example, a non-writable Array.prototype[7]) * the fast path will incorrectly ignore them. * * This fast path could be made compliant by falling through * to the slow path if the previous value was UNUSED. This would * also remove the need to check for extensibility. Right now a * non-extensible array is slower than an extensible one as far * as writes are concerned. * * The fast path behavior is documented in more detail here: * tests/ecmascript/test-misc-array-fast-write.js
label: code-design

4897. **comment:** XXX: because we unwind stacks above, thr->heap->curr_thread is at * risk of pointing to an already freed thread. This was indeed the * case in test-bug-multithread-valgrind.c, until duk_handle_call() * was fixed to restore thr->heap->curr_thread before rethrowing an * uncaught error.
label: code-design

4898. Allow naked fraction (e.g. ".123")

4899. * Rate limit check for sending status update or peeking into * the debug transport. Both can be expensive operations that * we don't want to do on every opcode. * * Making sure the interval remains reasonable on a wide variety * of targets and bytecode is difficult without a timestamp, so * we use a Date-provided timestamp for the rate limit check. * But since it's also expensive to get a timestamp, a bytecode * counter is used to rate limit getting timestamps.

4900. >>> struct.unpack('>d', '4000112233445566'.decode('hex')) * (2.008366013071895,)

4901. Other longjmp types are handled by executor before propagating * the error here.

4902. * E5 Section 11.8.6 describes the main algorithm, which uses * [[HasInstance]]. [[HasInstance]] is defined for only * function objects: * * - Normal functions: * E5 Section 15.3.5.3 * - Functions established with Function.prototype.bind(): * E5 Section 15.3.4.5.3 * * For other objects, a TypeError is thrown. * * Limited Proxy support: don't support 'getPrototypeOf' trap but * continue lookup in Proxy target if the value is a Proxy.

4903. * Ecmascript/native function call or lightfunc call

4904. Halt execution and enter a debugger message loop until execution is resumed * by the client. PC for the current activation may be temporarily decremented * so that the "current" instruction will be shown by the client. This helper * is callable from anywhere, also outside bytecode executor.

4905. computed live

4906. **comment:** Load a function from bytecode. The function object returned here must * match what is created by duk_js_push_closure() with respect to its flags, * properties, etc. * * NOTE: there are intentionally no input buffer length / bound checks. * Adding them would be easy but wouldn't ensure memory safety as untrusted * or broken bytecode is unsafe during execution unless the opcodes themselves * are validated (which is quite complex, especially for indirect opcodes).
label: code-design

4907. * For an external string, the NUL-terminated string data is stored * externally. The user must guarantee that data behind this pointer * doesn't change while it's used.

4908. Note: masking of 'x' is not necessary because of * range check and shifting -> no bits overlapping * the marker should be set.

4909. **comment:** * Object.defineProperty() related helper (E5 Section 15.2.3.6) * * Inlines all [[DefineOwnProperty]] exotic behaviors. * * Note: Ecmascript compliant [[DefineOwnProperty]](P, Desc, Throw) is not * implemented directly, but Object.defineProperty() serves its purpose. * We don't need the [[DefineOwnProperty]] internally and we don't have a * property descriptor with 'missing values' so it's easier to avoid it * entirely. * * Note: this is only called for actual objects, not primitive values. * This must support virtual properties for full objects (e.g. Strings) * but not for plain values (e.g. strings). Lightfuncs, even though * primitive in a sense, are treated like objects and accepted as target * values.
label: code-design

4910. callstack limits

4911. A...P

4912. Module table: * - module.exports: initial exports table (may be replaced by user) * - module.id is non-writable and non-configurable, as the CommonJS * spec suggests this if possible * - module.filename: not set, defaults to resolved ID if not explicitly * set by modSearch() (note capitalization, not .fileName, matches Node.js) * - module.name: not set, defaults to last component of resolved ID if * not explicitly set by modSearch()

4913. String constructor needs to distinguish between an argument not given at all * vs. given as 'undefined'. We're a vararg function to handle this properly.

4914. Explicit length is only needed if it differs from 'nargs'.

4915. Force restart caused by a function return; must recheck * debugger breakpoints before checking line transitions, * see GH-303. Restart and then handle interrupt_counter * zero again.

4916. Decode 'bits' bits from the input stream (bits must be 1...24). * When reading past bitstream end, zeroes are shifted in. The result * is signed to match duk_bd_decode_flagged.

4917. (?:

4918. Must be restored here to handle e.g. yields properly.

4919. The base ID of the current require() function, resolution base

4920. non-Reference deletion is always 'true', even in strict mode

4921. jump to next loop, using reg_v34_iter as iterated value

4922. Scan error: this shouldn't normally happen; it could happen if * string is not valid UTF-8 data, and clen/blen are not consistent * with the scanning algorithm.

4923. max possible

4924. punctuators (unlike the spec, also includes "/" and "=")

4925. DUK_BUFOBJ_FLOAT32ARRAY

4926. [... ptr]

4927. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property delete right now.

4928. 12: getDate

4929. * Helper for valstack space * * Caller of DUK_ASSERT_VALSTACK_SPACE() estimates the number of free stack entries * required for its own use, and any child calls which are not (a) Duktape API calls * or (b) Duktape calls which involve extending the valstack (e.g. getter call).

4930. 'modLoaded'

4931. * Assertions before

4932. **comment:** * Buffer writer (dynamic buffer only) * * Helper for writing to a dynamic buffer with a concept of a "spare" area * to reduce resizes. You can ensure there is enough space beforehand and * then write for a while without further checks, relying on a stable data * pointer. Spare handling is automatic so call sites only indicate how * much data they need right now. * * There are several ways to write using bufwriter. The best approach * depends mainly on how much performance matters over code footprint. * The key issues are (1) ensuring there is space and (2) keeping the * pointers consistent. Fast code should ensure space

for multiple writes * with one ensure call. Fastest inner loop code can temporarily borrow * the 'p' pointer but must write it back eventually. ** Be careful to ensure all macro arguments (other than static pointers like * 'thr' and 'bw_ctx') are evaluated exactly once, using temporaries if * necessary (if that's not possible, there should be a note near the macro). * Buffer write arguments often contain arithmetic etc so this is * particularly important here.

label: code-design

4933. Ordering should not matter (E5 Section 11.8.5, step 3.a) but * preserve it just in case.

4934. for zero size, don't push anything on valstack

4935. Here we'd have the option of decoding unpadded base64 * (e.g. "xxxxyy" instead of "xxxxyy==". Currently not * accepted.

4936. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway

4937. * Conceptually, we look for the identifier binding by starting from * 'env' and following to chain of environment records (represented * by the prototype chain). ** If 'env' is NULL, the current activation does not yet have an * allocated declarative environment record; this should be treated * exactly as if the environment record existed but had no bindings * other than register bindings. ** Note: we assume that with the DUK_HOBJECT_FLAG_NEWENV cleared * the environment will always be initialized immediately; hence * a NULL 'env' should only happen with the flag set. This is the * case for: (1) function calls, and (2) strict, direct eval calls.

4938. heap level "stash" object (e.g., various reachability roots)

4939. * For global or eval code this is straightforward. For functions * created with the Function constructor we only get the source for * the body and must manufacture the "function ..." part. ** For instance, for constructed functions (v8): * * > a = new Function("foo", "bar", "print(foo)"); * [Function] * > a.toString() * 'function anonymous(foo,bar) {nprint(foo)n}' * * Similarly for e.g. getters (v8): * * > x = { get a(foo,bar) { print(foo); } } * { a: [Getter] } * > Object.getOwnPropertyDescriptor(x, 'a').get.toString() * 'function a(foo,bar) { print(foo); }'

4940. c recursion check

4941. XXX: optimize by adding the token numbers directly into the * always interned duk_hstring objects (there should be enough * flag bits free for that)?

4942. Continue parsing after padding, allows concatenated, * padded base64.

4943. Check that 'this' is a duk_hbufferobject and return a pointer to it * (NULL if not).

4944. idx_step is +1 for indexOf, -1 for lastIndexOf

4945. is a setter/getter

4946. env created above to stack top

4947. cannot be e.g. arguments exotic, since exotic 'traits' are mutually exclusive

4948. DUK_TOK_LNOT

4949. XXX: sufficient to check 'strict', assert for 'is function'

4950. check final character validity

4951. * Init lexer context

4952. * Incref and decref functions. ** Decref may trigger immediate refzero handling, which may free and finalize * an arbitrary number of objects. *

4953. Generate pc2line data for an instruction sequence, leaving a buffer on stack top.

4954. a dummy undefined value is pushed to make valstack * behavior uniform for caller

4955. idx_step is +1 for reduce, -1 for reduceRight

4956. PatternCharacter, all excluded characters are matched by cases above

4957. Note: -nargs alone would fail for nargs == 0, this is OK

4958. curr and desc are accessors

4959. * Throw the error in the resumed thread's context; the * error value is pushed onto the resumee valstack. ** Note: the callstack of the target may empty in this case * too (i.e. the target thread has never been resumed). The * value stack will contain the initial function in that case, * which we simply ignore.

4960. For string-to-number, pretend we never have the lowest mantissa as there * is no natural "precision" for inputs. Having lowest_mantissa == 0, we'll * fall into the base cases for both e >= 0 and e < 0.

4961. Number built-in accepts a plain number or a Number object (whose * internal value is operated on). Other types cause TypeError.

4962. Encode to extended UTF-8; 'out' must have space for at least * DUK_UNICODE_MAX_XUTF8_LENGTH bytes. Allows encoding of any * 32-bit (unsigned) codepoint.

4963. func.prototype.constructor = func

4964. Accept any pointer-like value; for 'object' dvalue, read * and ignore the class number.

4965. **comment:** Clone the properties of the ROM-based global object to create a * fully RAM-based global object. Uses more memory than the inherit * model but more compliant.

label: code-design

4966. prev value can be garbage, no decref

4967. **comment:** * For/for-in statement is complicated to parse because * determining the statement type (three-part for vs. a * for-in) requires potential backtracking. ** See the helper for the messy stuff.

label: code-design

4968. Stepped? Step out is handled by callstack unwind.

4969. Copy values, the copy method depends on the arguments. ** Copy mode decision may depend on the validity of the underlying * buffer of the source argument; there must be no harmful side effects * from there to here for copy_mode to still be valid.

4970. fills window

4971. Caller must check character offset to be inside the string.

4972. valstack will be unbalanced, which is OK

4973. conservative

4974. unconditional

4975. DUK_USE_FASTINT && DUK_USE_PACKED_TVAL

4976. * NEXTENUM checks whether the enumerator still has unenumerated * keys. If so, the next key is loaded to the target register * and the next instruction is skipped. Otherwise the next instruction * will be executed, jumping out of the enumeration loop.

4977. **comment:** Providing access to e.g. act->lex_env would be dangerous: these * internal structures must never be accessible to the application. * Duktape relies on them having consistent data, and this consistency * is only asserted for, not checked for.

label: code-design

4978. Duktape.modSearch

4979. Set .buffer to the argument ArrayBuffer.

4980. * Parse a function-like expression: * * - function expression * - function declaration * - function statement (non-standard) * - setter/getter ** Adds the function to comp_ctxt->curr_func function table and returns the * function number. ** On entry, curr_token points to: * * - the token after 'function' for function expression/declaration/statement * - the token after 'set' or 'get' for setter/getter

4981. * Match loop. ** Try matching at different offsets until match found or input exhausted.

4982. flags: ''

4983. -> [res_obj]

4984. nbytes * <-----> * [... | p | x | x | q] * => [... | q | p | x | x]

4985. eat '{' on entry

4986. This native function is also used for Date.prototype.getTime() * as their behavior is identical.

4987. resume write to target

4988. if new_size < L * old_size, resize without abandon check; L = 3-bit fixed point, e.g. 9 -> 9/8 = 112.5%

4989. Note: we could try to stuff a partial hash (e.g. 16 bits) into the * shared heap header. Good hashing needs more hash bits though.

4990. Else functionality is identical for function call and constructor * call.

4991. inclusive

4992. Check that 'this' is a duk_hbufferobject and return a pointer to it.

4993. utf-8 continuation bytes have the form 10xx xxxx

4994. This is based on V8 EquivalentYear() algorithm (see src/genequivyear.py): * http://code.google.com/p/v8/source/browse/trunk/src/date.h#146

4995. 'buf' contains the string to write, 'sz_buf' contains the length * (which may be zero).

4996. ''

4997. hobject management functions
4998. Convert day from one-based to zero-based (internal). This may * cause the day part to be negative, which is OK.
4999. 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx [26 bits]
5000. There is no need to check the first character specially here * (i.e. reject digits): the caller only accepts valid initial * characters and won't call us if the first character is a digit. * This also ensures that the plain string won't be empty.
5001. \xffBytecode'
5002. Catch a double-to-int64 cast issue encountered in practice.
5003. * DumpHeap command
5004. **comment:** * Lexer for source files, ToNumber() string conversions, RegExp expressions, * and JSON. * * Provides a stream of Ecmascript tokens from an UTF-8/CESU-8 buffer. The * caller can also rewind the token stream into a certain position which is * needed by the compiler part for multi-pass scanning. Tokens are * represented as duk_token structures, and contain line number information. * Token types are identified with DUK_TOK_*. * * Characters are decoded into a fixed size lookup window consisting of * decoded Unicode code points, with window positions past the end of the * input filled with an invalid codepoint (-1). The tokenizer can thus * perform multiple character lookups efficiently and with few sanity * checks (such as access outside the end of the input), which keeps the * tokenization code small at the cost of performance. * * Character data in tokens, such as identifier names and string literals, * is encoded into CESU-8 format on-the-fly while parsing the token in * question. The string data is made reachable to garbage collection by * placing the token-related values in value stack entries allocated for * this purpose by the caller. The characters exist in Unicode code point * form only in the fixed size lookup window, which keeps character data * expansion (of especially ASCII data) low. * * Token parsing supports the full range of Unicode characters as described * in the E5 specification. Parsing has been optimized for ASCII characters * because ordinary Ecmascript code consists almost entirely of ASCII * characters. Matching of complex Unicode codepoint sets (such as in the * IdentifierStart and IdentifierPart productions) is optimized for size, * and is done using a linear scan of a bit-packed list of ranges. This is * very slow, but should never be entered unless the source code actually * contains Unicode characters. * * Ecmascript tokenization is partially context sensitive. First, * additional future reserved words are recognized in strict mode (see E5 * Section 7.6.1.2). Second, a forward slash character ('/') can be * recognized either as starting a RegExp literal or as a division operator, * depending on context. The caller must provide necessary context flags * when requesting a new token. * * Future work: * * * Make line number tracking optional, as it consumes space. * * * Add a feature flag for disabling UTF-8 decoding of input, as most * source code is ASCII. Because of Unicode escapes written in ASCII, * this does not allow Unicode support to be removed from e.g. * duk_unicode_is_identifier_start() nor does it allow removal of CESU-8 * encoding of e.g. string literals. * * * Add a feature flag for disabling Unicode compliance of e.g. identifier * names. This allows for a build more than a kilobyte smaller, because * Unicode ranges needed by duk_unicode_is_identifier_start() and * duk_unicode_is_identifier_part() can be dropped. String literals * should still be allowed to contain escaped Unicode, so this still does * not allow removal of CESU-8 encoding of e.g. string literals. * * * Character lookup tables for codepoints above BMP could be stripped. * * * Strictly speaking, E5 specification requires that source code consists * of 16-bit code units, and if not, must be conceptually converted to * that format first. The current lexer processes Unicode code points * and allows characters outside the BMP. These should be converted to * surrogate pairs while reading the source characters into the window, * not after tokens have been formed (as is done now). However, the fix * is not trivial because two characters are decoded from one codepoint. * * * Optimize for speed as well as size. Large if-else ladders are (at * least potentially) slow.
label: code-design
5005. [... buf loop]
5006. **comment:** * Heap object representation. * * Heap objects are used for Ecmascript objects, arrays, and functions, * but also for internal control like declarative and object environment * records. Compiled functions, native functions, and threads are also * objects but with an extended C struct. * * Objects provide the required Ecmascript semantics and exotic behaviors * especially for property access. * * Properties are stored in three conceptual parts: * * 1. A linear 'entry part' contains ordered key-value-attributes triples * and is the main method of string properties. * * 2. An optional linear 'array part' is used for array objects to store a * (dense) range of [0,N[array indexed entries with default attributes * (writable, enumerable, configurable). If the array part would become * sparse or non-default attributes are required, the array part is * abandoned and moved to the 'entry part'. * * 3. An optional 'hash part' is used to optimize lookups of the entry * part; it is used only for objects with sufficiently many properties * and can be abandoned without loss of information. * * These three conceptual parts are stored in a single memory allocated area. * This minimizes memory allocation overhead but also means that all three * parts are resized together, and makes property access a bit complicated.
label: code-design
5007. x <- y + z
5008. get array index related to string (or return DUK_HSTRING_NO_ARRAY_INDEX); * avoids helper call if string has no array index value.
5009. [... re_obj input bc]
5010. Keep the error as the result (coercing it might fail below, * but we don't catch that now).
5011. * Note: although arguments object variable mappings are only established * for non-strict functions (and a call to a non-strict function created * the arguments object in question), an inner strict function may be doing * the actual property write. Hence the throw_flag applied here comes from * the property write call.
5012. push before advancing to keep reachable
5013. * duk_hthread allocation and freeing.
5014. [... pattern flags escaped_source bytecode]
5015. * Helper for updating callee 'caller' property.
5016. eat 'do'
5017. saves a few instructions to have this wrapper (see comment on duk_heap_mem_alloc)
5018. [source template result]
5019. 25 user flags
5020. also stash it before constructor, * in case we need it (as the fallback value)
5021. omit print
5022. Unary plus is used to force a fastint check, so must include * downgrade check.
5023. DUK_FLD_16BIT
5024. **comment:** should never happen, but be robust
label: code-design
5025. _Formals: omitted if function is guaranteed not to need a (non-strict) arguments object
5026. **comment:** XXX: comes out as signed now
label: code-design
5027. flags
5028. **comment:** If message is undefined, the own property 'message' is not set at * all to save property space. An empty message is inherited anyway.
label: code-design
5029. writable but not deletable
5030. Decode UTF-8 codepoint from a sequence of hex escapes. The * first byte of the sequence has been decoded to 't'. * * Note that UTF-8 validation must be strict according to the * specification: E5.1 Section 15.1.3, decode algorithm step * 4.d.vii.8. URIError from non-shortest encodings is also * specifically noted in the spec.
5031. [... formals]
5032. **comment:** avoid tag 0xffff0, no risk of confusion with negative infinity
label: code-design
5033. See: tests/ecmascript/test-spec-bound-constructor.js
5034. guaranteed to succeed
5035. * Define new property * * Note: this may fail if the holder is not extensible.
5036. used elements (includes DELETED)
5037. **comment:** XXX: since the enumerator may be a memory expensive object, * perhaps clear it explicitly here? If so, break jump must * go through this clearing operation.
label: code-design
5038. -> [... key val replacer holder key]
5039. as is
5040. Step 1 is not necessary because duk_call_method() will take * care of it.

5041. * Code emission helpers ** Some emission helpers understand the range of target and source reg/const * values and automatically emit shuffling code if necessary. This is the * case when the slot in question (A, B, C) is used in the standard way and * for opcodes the emission helpers explicitly understand (like DUK_OP_CALL). ** The standard way is that: * - slot A is a target register * - slot B is a source register/constant * - slot C is a source register/constant * * If a slot is used in a non-standard way the caller must indicate this * somehow. If a slot is used as a target instead of a source (or vice * versa), this can be indicated with a flag to trigger proper shuffling * (e.g. DUK_EMIT_FLAG_B_IS_TARGET). If the value in the slot is not * register/const related at all, the caller must ensure that the raw value * fits into the corresponding slot so as to not trigger shuffling. The * caller must set a "no shuffle" flag to ensure compilation fails if * shuffling were to be triggered because of an internal error. ** For slots B and C the raw slot size is 9 bits but one bit is reserved for * the reg/const indicator. To use the full 9-bit range for a raw value, * shuffling must be disabled with the DUK_EMIT_FLAG_NO_SHUFFLE_{B,C} flag. * Shuffling is only done for A, B, and C slots, not the larger BC or ABC slots. ** There is call handling specific understanding in the A-B-C emitter to * convert call setup and call instructions into indirect ones if necessary.

5042. **comment:** XXX: make this an internal helper
label: code-design

5043. assumed to bottom relative

5044. intentionally empty

5045. Compute time value from (double) parts. The parts can be either UTC * or local time; if local, they need to be (conceptually) converted into * UTC time. The parts may represent valid or invalid time, and may be * wildly out of range (but may cancel each other and still come out in * the valid Date range).

5046. aaaaaabb bbbbcccc cccccddd

5047. [[SetPrototypeOf]] standard behavior, E6 9.1.2

5048. **comment:** XXX: could unwind catchstack here, so that call handling * didn't need to do that?
label: code-design

5049. 27: setUTCSeconds

5050. Lightfunc virtual properties are non-configurable, so * reject if match any of them.

5051. **comment:** init is unnecessary but suppresses "may be used uninitialized" warnings
label: code-design

5052. no need to unwind catchstack

5053. 'Proxy' object

5054. 'errCreate'

5055. coerce lval with ToString()

5056. see algorithm

5057. * reverse()

5058. index is above internal buffer length -> property is fully normal

5059. ensure never re-entered until rescue cycle complete

5060. **comment:** order: most frequent to least frequent
label: code-design

5061. **comment:** not very useful, used for debugging
label: code-design

5062. catch stack depth

5063. [...] res_obj]

5064. * Lexer defines.

5065. A token value. Can be memcpy()'d, but note that slot1/slot2 values are on the valstack. * Some fields (like num, str1, str2) are only valid for specific token types and may have * stale values otherwise.

5066. DUK_BUFOBJ_NODEJS_BUFFER

5067. XXX: if 'len' is low, may want to ensure array part is kept: * the caller is likely to want a dense array.

5068. XXX: version specific array format instead?

5069. * Process incoming debug requests ** Individual request handlers can push temporaries on the value stack and * rely on duk_debug_process_message() to restore the value stack top * automatically.

5070. roughly 2 kB

5071. Must avoid duk_pop() in exit path

5072. basic types from duk_features.h

5073. XXX: byte offset

5074. * instanceof

5075. ecmascript compiler limits

5076. token type

5077. -> [...] key val replacer]

5078. call_flags

5079. * Comparisons (x >= y, x > y, x <= y, x < y) ** E5 Section 11.8.5: implement 'x < y' and then use negate and eval_left_first * flags to get the rest.

5080. Negative indices are always within allocated stack but * must not go below zero index.

5081. Executor interrupt default interval when nothing else requires a * smaller value. The default interval must be small enough to allow * for reasonable execution timeout checking but large enough to keep * impact on execution performance low.

5082. * Object internal value ** Returned value is guaranteed to be reachable / incref'd, caller does not need * to incref OR decref. No proxies or accessors are invoked, no prototype walk.

5083. Exponent

5084. general temp register

5085. 0xe0...0xef

5086. Maximum write size: XUTF8 path writes max DUK_UNICODE_MAX_XUTF8_LENGTH, * percent escape path writes max two times CESU-8 encoded BMP length.

5087. [arg1 ... argN this loggerLevel loggerName buffer]

5088. duplicate zeroing, expect for (possible) NULL inits

5089. rc_varname and reg_varbind are ignored here

5090. get as double to handle huge numbers correctly

5091. enum_index

5092. XXX: clarify on when and where DUK_CONST_MARKER is allowed

5093. * Error helpers

5094. * Data following the header depends on the DUK_HBUFFER_FLAG_DYNAMIC * flag. ** If the flag is clear (the buffer is a fixed size one), the buffer * data follows the header directly, consisting of 'size' bytes. ** If the flag is set, the actual buffer is allocated separately, and * a few control fields follow the header. Specifically: * * - a "void *" pointing to the current allocation * - a duk_size_t indicating the full allocated size (always >= 'size') * * If DUK_HBUFFER_FLAG_EXTERNAL is set, the buffer has been allocated * by user code, so that Duktape won't be able to resize it and won't * free it. This allows buffers to point to e.g. an externally * allocated structure such as a frame buffer. ** Unlike strings, no terminator byte (NUL) is guaranteed after the * data. This would be convenient, but would pad aligned user buffers * unnecessarily upwards in size. For instance, if user code requested * a 64-byte dynamic buffer, 65 bytes would actually be allocated which * would then potentially round upwards to perhaps 68 or 72 bytes.

5095. 'new'

5096. evaluate to plain value, no forced register (temp/bound reg both ok)

5097. Fast path is triggered for no exponent and also for balanced exponent * and fraction parts, e.g. for "1.23e2" == "123". Remember to respect * zero sign.

5098. DUK_USE_INTERRUPT_COUNTER

5099. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property put right now (putprop protects * against it internally).

5100. **comment:** * Parser control values for tokens. The token table is ordered by the * DUK_TOK_XXX defines. ** The binding powers are for lbp() use (i.e. for use in led() context). * Binding powers are positive for typing convenience, and bits at the * top should be reserved for flags. Binding power step must be higher * than 1 so that binding power "lbp - 1" can be used for right associative * operators. Currently a step of 2 is used (which frees one more bit for * flags).

label: code-design

5101. * Array part * * Note: ordering between array and entry part must match 'abandon array' * behavior in duk_hobject_props.c: key order after an array is abandoned * must be the same.

5102. **comment:** XXX: could be eliminated with valstack adjust

label: code-design

5103. [key val]

5104. DUK_TOK_CATCH

5105. Note: we rely on duk_safe_call() to fix up the stack for the caller, * so we don't need to pop stuff here. There is no return value; * caller determines rescued status based on object refcount.

5106. * Case conversion helper, with context/local sensitivity. * For proper case conversion, one needs to know the character * and the preceding and following characters, as well as * locale/language.

5107. [... arg1 ... argN envobj argobj]

5108. flags:nonstrict

5109. pass2 allocation handles this

5110. * Call the wrapped module function. * * Use a protected call so that we can update Duktape.modLoaded[resolved_id] * even if the module throws an error.

5111. advance manually

5112. **comment:** XXX: awkward

label: code-design

5113. target out-of-bounds (but positive)

5114. free some memory

5115. constrained by string length

5116. stable; precalculated for faster lookups

5117. * Thread defines

5118. same handling for identifiers and strings

5119. -> [...] lval rval rval.prototype]

5120. 'implements'

5121. !=

5122. XXX: Not sure what the best return value would be in the API. * Return a boolean for now. Note that rc == 0 is success (true).

5123. Current convention is to use duk_size_t for value stack sizes and global indices, * and duk_idx_t for local frame indices.

5124. **comment:** XXX: where to release temp regs in intermediate expressions? * e.g. 1+2+3 -> don't inflate temp register count when parsing this. * that particular expression temp regs can be forced here.

label: code-design

5125. negative top-relative indices not allowed now

5126. instanceof

5127. avoid attempt to compact any objects

5128. expression parsing helpers

5129. return ToObject(this)

5130. one token

5131. flags field: LLLLLLFT, L = label (24 bits), F = flags (4 bits), T = type (4 bits)

5132. not protected, respect reclimit, is a constructor call

5133. DUK_TOK_BNOT

5134. E5 Section 8.6.1

5135. indicates a deleted string; any fixed non-NULL, non-hstring pointer works

5136. val is unsigned so >= 0

5137. **comment:** NOTE: we try to minimize code size by avoiding unnecessary pops, * so the stack looks a bit cluttered in this function. DUK_ASSERT_TOP() * assertions are used to ensure stack configuration is correct at each * step.

label: code-design

5138. must have start and end

5139. Check for breakpoints only on line transition. * Breakpoint is triggered when we enter the target * line from a different line, and the previous line * was within the same function. * * This condition is tricky: the condition used to be * that transition to -or across- the breakpoint line * triggered the breakpoint. This seems intuitively * better because it handles breakpoints on lines with * no emitted opcodes; but this leads to the issue * described in: <https://github.com/svaarala/duktape/issues/263>.

5140. env and act may be NULL

5141. * Process yield * * After longjmp(), processing continues in bytecode executor longjmp * handler, which will e.g. update thr->resumer to NULL.

5142. -> [...] enum key enum_target key]

5143. **comment:** XXX: store 'bcode' pointer to activation for faster lookup?

label: code-design

5144. * Unicode tables

5145. [... obj]

5146. Note: space must cater for both JX and JC.

5147. Needs to be inserted; scan backwards, since we optimize * for the case where elements are nearly in order.

5148. Normal non-bound function.

5149. Eating a left curly; regexp mode is allowed by left curly * based on duk__token_lbp[] automatically.

5150. A validated read() is always a number, so it's write coercion * is always side effect free and won't invalidate pointers etc.

5151. current and previous token for parsing

5152. probe sequence

5153. message is empty -> return name

5154. NaN timevalue: we need to coerce the arguments, but * the resulting internal timestamp needs to remain NaN. * This works but is not pretty: parts and dparts will * be partially uninitialized, but we only write to them.

5155. **comment:** It might seem that replacing 'thr->heap' with just 'heap' below * might be a good idea, but it increases code size slightly * (probably due to unnecessary spilling) at least on x64.

label: code-design

5156. Currently nothing to free; 'data' is a heap object

5157. 'prototype'

5158. 'jx'

5159. default prototype (Note: 'obj' must be reachable)

5160. **comment:** XXX: currently NULL allocations are not supported; remove if later allowed

label: code-design

5161. Unwind the topmost callstack entry before reusing it

5162. internal define property: skip write silently if exists

5163. since new_alloc_size > 0

5164. borrowed, no refcount

5165. allow e.g. '0x0009' and '00077'

5166. DUK_BUFOBJ_DATAVIEW

5167. * Compact an object. Minimizes allocation size for objects which are * not likely to be extended. This is useful for internal and non- * extensible objects, but can also be called for non-extensible objects. * May abandon the array part if it is computed to be too sparse. * * This call is relatively expensive, as it needs to scan both the * entries and the array part. * * The call may fail due to allocation error.

5168. find elements to swap

5169. currently, always the case
5170. * Detected label
5171. **comment:** XXX: an array can have length higher than 32 bits; this is not handled * correctly now.
 label: code-design
5172. * Helper macros
5173. **comment:** XXX: Simplify this algorithm, should be possible to come up with * a shorter and faster algorithm by inspecting IEEE representation * directly.
 label: code-design
5174. * CheckObjectCoercible() (E5 Section 9.10) * * Note: no API equivalent now.
5175. constants are strings or numbers now
5176. '\n'
5177. Must behave like a no-op with NULL and any pointer returned from * malloc/realloc with zero size.
5178. if (is_regexp)
5179. [thisArg arg1 ... argN func boundFunc]
5180. -Infinity
5181. Undefine local defines
5182. * Top-level include file to be used for all (internal) source files. * * Source files should not include individual header files, as they * have not been designed to be individually included.
5183. * Init built-in strings
5184. * Two's complement arithmetic.
5185. [... constructor arg1 ... argN]
5186. no need to decref previous value
5187. 'finally'
5188. Zero size compare not an issue with DUK_MEMCMP.
5189. [... closure template val]
5190. YIELDED: Ecmascript activation + Duktape.Thread.yield() activation
5191. may be NULL (but only if size is 0)
5192. write to entry part
5193. * Ecmascript bytecode executor. * * Resume execution for the current thread from its current activation. * Returns when execution would return from the entry level activation, * leaving a single return value on top of the stack. Function calls * and thread resumptions are handled internally. If an error occurs, * a longjmp() with type DUK_LJ_TYPE_THROW is called on the entry level * setjmp() jmpbuf. * * Ecmascript function calls and coroutine resumptions are handled * internally (by the outer executor function) without recursive C calls. * Other function calls are handled using duk_handle_call(), increasing * C recursion depth. * * Abrupt completions (= long control transfers) are handled either * directly by reconfiguring relevant stacks and restarting execution, * or via a longjmp. Longjmp-free handling is preferable for performance * (especially Emscripten performance), and is used for: break, continue, * and return. * * For more detailed notes, see doc/execution.rst. * * Also see doc/code-issues.rst for discussion of setjmp(), longjmp(), * and volatile.
5194. * Debug connection message write helpers
5195. We know _Formals is dense and all entries will be in the * array part. GC and finalizers shouldn't affect _Formals * so side effects should be fine.
5196. don't want an intermediate exposed state with func == NULL
5197. **comment:** * (Re)try handling the longjmp. * * A longjmp handler may convert the longjmp to a different type and * "virtually" rethrow by goto'ing to 'check_longjmp'. Before the goto, * the following must be updated: * - the heap 'lj' state * - 'thr' must reflect the "throwing" thread
 label: code-design
5198. XXX=

5199. separators: space, space, colon
5200. **comment:** Format of magic, bits: * 0...1: field type; 0:uint8, 1:uint16, 2:uint32, 3=float, 4=double, 5=unused, 6=unused, 7=unused * 3: endianness: 0=little, 1=big * 4: signed: 1=yes, 0=no * 5: typedarray: 1=yes, 0=no
 label: code-design

5201. **comment:** Note: reuse 'curr' as a temp propdesc
 label: code-design

5202. Slow path
5203. Lightweight function: never bound, so terminate.
5204. unconditionally; is_global==0
5205. Node.js Buffer: return offset + #bytes written (i.e. next * write offset).
5206. string limits
5207. **comment:** function makes one or more slow path accesses
 label: code-design

5208. DUK_USE_AUGMENT_ERROR_CREATE
5209. No need for a NULL/zero-size check because new_size > 0
5210. Assume value stack sizes (in elements) fits into duk_idx_t.
5211. **comment:** Relookup in case duk_js_tointeger() ends up e.g. coercing an object.
 label: code-design

5212. '>'
5213. * Debug connection skip primitives
5214. valstack index of 'func' and retval (relative to entry valstack_bottom)
5215. XXX: array internal?
5216. Assertion compatible inside a comma expression, evaluates to void. * Currently not compatible with DUK_USE_PANIC_HANDLER() which may have * a statement block.
5217. avoid key quotes when key is an ASCII Identifier
5218. embed: nothing
5219. **comment:** bit mask which indicates that a regconst is a constant instead of a register
 label: code-design

5220. * Memory allocation handling.
5221. current assertion is quite strong: decref's and set to unused
5222. Coerce an duk_ivalue to a 'plain' value by generating the necessary * arithmetic operations, property access, or variable access bytecode. * The duk_ivalue argument ('x') is converted into a plain value as a * side effect.

5223. element type
5224. parsing in "scanning" phase (first pass)
5225. regexp res_obj is at offset 4
5226. Flags for duk_js_compare_helper().

5227. **comment:** * Eval * * Eval needs to handle both a "direct eval" and an "indirect eval". * Direct eval handling needs access to the caller's activation so that its * lexical environment can be accessed. A direct eval is only possible from * Ecmascript code; an indirect eval call is possible also from C code. * When an indirect eval call is made from C code, there may not be a * calling activation at all which needs careful handling.
 label: code-design

5228. no fractions
5229. 'Function'
5230. 29: setUTCMinutes
5231. * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written.
5232. accept comma, expect new value
5233. -> [... reviver holder name val]
5234. [... pattern flags escaped_source buffer]

5235. reg
5236. empty match -> bump and continue
5237. regexp execution limits
5238. * Single source autogenerated distributable for Duktape 1.5.2. ** Git commit cad34ae155acb0846545ca6bf2d29f9463b22bbb (v1.5.2). * Git branch HEAD. ** See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.
5239. SCANBUILD: "Dereference of null pointer", normal
5240. catch or with binding is currently active
5241. Note: unbalanced stack on purpose
5242. Get a borrowed duk_tval pointer to the current 'this' binding. Caller must * make sure there's an active callstack entry. Note that the returned pointer * is unstable with regards to side effects.
5243. expt values [0x001,0x7fe] = normal
5244. Receiver: Proxy object
5245. **comment:** * Dump/load helpers, xxx_raw() helpers do no buffer checks
label: code-design
5246. **comment:** * Push readable string summarizing duk_tval. The operation is side effect * free and will only throw from internal errors (e.g. out of memory). * This is used by e.g. property access code to summarize a key/base safely, * and is not intended to be fast (but small and safe).
label: code-design
5247. * Type error thrower, E5 Section 13.2.3.
5248. **comment:** Corner case: see test-numconv-parse-mant-carry.js. We could * just bump the exponent and update bitstart, but it's more robust * to recompute (but avoid rounding twice).
label: code-design
5249. The JA(value) operation: encode array. ** Stack policy: [array] -> [array].
5250. XXX: init or assert catch depth etc -- all values
5251. happens when hash part dropped
5252. 1x heap size
5253. no need for decref/incref because value is a number
5254. **comment:** XXX: this may now fail, and is not handled correctly
label: code-design
5255. [key undefined] -> [key]
5256. * join(), toLocaleString() ** Note: checking valstack is necessary, but only in the per-element loop. ** Note: the trivial approach of pushing all the elements on the value stack * and then calling duk_join() fails when the array contains a large number * of elements. This problem can't be offloaded to duk_join() because the * elements to join must be handled here and have special handling. Current * approach is to do intermediate joins with very large number of elements. * There is no fancy handling; the prefix gets re-joined multiple times.
5257. stack top contains 'false'
5258. LAYOUT 2
5259. If 16-bit hash is in use, stuff it into duk_heapHdr_string flags.
5260. * Other cases, use C recursion. ** If a tail call was requested we ignore it and execute a normal call. * Since Duktape 0.11.0 the compiler emits a RETURN opcode even after * a tail call to avoid test-bug-tailcall-thread-yield-resume.js. ** Direct eval call: (1) call target (before following bound function * chain) is the built-in eval() function, and (2) call was made with * the identifier 'eval'.
5261. (

5262. NOTE! Caller must ensure that any side effects from the * coercions below are safe. If that cannot be guaranteed * (which is normally the case), caller must coerce the * argument using duk_to_number() before any pointer * validations; the result of duk_to_number() always coerces * without side effects here.
5263. **comment:** Compared to duk_handle_call(): * - protected call: never * - ignore recursion limit: never
label: code-design
5264. Note: we ask for one return value from duk_safe_call to get this * error debugging here.
5265. -> [sep ToObject(this) len str]
5266. covered by comparison
5267. Setting "no shuffle A" is covered by the assert, but it's needed * with DUK_USE_SHUFFLE_TORTURE.
5268. **comment:** XXX: .code = err_code disabled, not sure if useful
label: code-design
5269. For now, restrict result array into 32-bit length range.
5270. coerce in-place
5271. DUK_ERR_ASSERTION_ERROR: no macros needed
5272. directive prologue status at entry
5273. * ToNumber() (E5 Section 9.3) ** Value to convert must be on stack top, and is popped before exit. ** See: <http://www.cs.indiana.edu/~burger/FP-Printing-PLDI96.pdf> * http://www.cs.indiana.edu/~burger/fp/index.html ** Notes on the conversion: ** - There are specific requirements on the accuracy of the conversion * through a "Mathematical Value" (MV), so this conversion is not * trivial. ** - Quick rejects (e.g. based on first char) are difficult because * the grammar allows leading and trailing white space. ** - Quick reject based on string length is difficult even after * accounting for white space; there may be arbitrarily many * decimal digits. ** - Standard grammar allows decimal values ("123"), hex values * ("0x123") and infinities * * - Unlike source code literals, ToNumber() coerces empty strings * and strings with only whitespace to zero (not NaN).
5274. [... num]
5275. arrays MUST have a 'length' property
5276. Using an explicit 'ASCII' flag has larger footprint (one call site * only) but is quite useful for the case when there's no explicit * 'clen' in duk_hstring.
5277. **comment:** XXX: try to optimize to 8 (would now be possible, <200 used)
label: code-design
5278. * Arguments handling helpers (argument map mainly). ** An arguments object has exotic behavior for some numeric indices. * Accesses may translate to identifier operations which may have * arbitrary side effects (potentially invalidating any duk_tval * pointers).
5279. Undefined/null are considered equal (e.g. "null == undefined" -> true).
5280. * String value of 'blen+1' bytes follows (+1 for NUL termination * convenience for C API). No alignment needs to be guaranteed * for strings, but fields above should guarantee alignment-by-4 * (but not alignment-by-8).
5281. **comment:** XXX: Could be improved by coercing to a readable duk_tval (especially string escaping)
label: code-design
5282. **comment:** Inner executor, performance critical.
label: code-design
5283. significant from precision perspective
5284. [... Logger clog logfunc clog]
5285. No assertions for offset or length; in particular, \ * it's OK for length to be longer than underlying \ * buffer. Just ensure they don't wrap when added. \
5286. fits into duk_small_int_t
5287. Caller doesn't need to check exotic proxy behavior (but does so for * some fast paths).
5288. is eval code
5289. * Iterate the bitstream (line diffs) until PC is reached
5290. varmap is already in comp_ctx->curr_func.varmap_idx
5291. * Convert binary digits into an IEEE double. Need to handle * denormals and rounding correctly.
5292. [key setter this val] -> [key retval]
5293. DUK_TOK LAND
5294. **comment:** * E5 Section 15.3.4.2 places few requirements on the output of * this function: ** - The result is an implementation dependent representation * of the function; in particular ** - The result must follow the syntax of a FunctionDeclaration. * In particular, the function must have a name (even in the * case of an anonymous function or a function with an empty * name). ** - Note in particular that the output does NOT need to compile * into anything useful.

label: code-design

5295. Note that multiple catchstack entries may refer to the same * callstack entry.

5296. (-Number.MAX_VALUE).toString(2).length == 1025, + spare

5297. DUK_HNATIVEFUNCTION_H_INCLUDED

5298. check pointer at end

5299. **comment:** XXX: any way to avoid decoding magic bit; there are quite * many function properties and relatively few with magic values.

label: code-design

5300. chain jumps for 'fall-through' * after a case matches.

5301. duk_unicode_ids_noa[]

5302. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).

label: code-design

5303. reset longjmp

5304. non-strict equality for buffers compares contents

5305. all codepoints up to U+FFFF

5306. recursion tracking happens here only

5307. roughly 128 bytes

5308. default clause matches next statement list (if any)

5309. * E5 Sections 11.8.7, 8.12.6. * * Basically just a property existence check using [[HasProperty]].

5310. new value

5311. MULTIPLICATIVE EXPRESSION

5312. **comment:** * typeof * * E5 Section 11.4.3. * * Very straightforward. The only question is what to return for our * non-standard tag / object types. * * There is an unfortunate string constant define naming problem with * typeof return values for e.g. "Object" and "object"; careful with * the built-in string defines. The LC_XXX defines are used for the * lowercase variants now.

label: code-design

5313. -> [... func this]

5314. **comment:** XXX: should this happen in the callee's activation or after unwinding?

label: code-design

5315. **comment:** * Four possible outcomes: * * 1. A 'finally' in the same function catches the 'return'. * It may continue to propagate when 'finally' is finished, * or it may be neutralized by 'finally' (both handled by * ENDFIN). * * 2. The return happens at the entry level of the bytecode * executor, so return from the executor (in C stack). * * 3. There is a calling (Ecmascript) activation in the call * stack => return to it, in the same executor instance. * * 4. There is no calling activation, and the thread is * terminated. There is always a resumer in this case, * which gets the return value similarly to a 'yield' * (except that the current thread can no longer be * resumed).

label: code-design

5316. caller checks

5317. t has high mantissa

5318. * Flags * * Fixed buffer: 0 * Dynamic buffer: DUK_HBUFFER_FLAG_DYNAMIC * External buffer: DUK_HBUFFER_FLAG_DYNAMIC | DUK_HBUFFER_FLAG_EXTERNAL

5319. DUK_TOK_LCURLY

5320. **comment:** * Identifier access and function closure handling. * * Provides the primitives for slow path identifier accesses: GETVAR, * PUTVAR, DELVAR, etc. The fast path, direct register accesses, should * be used for most identifier accesses. Consequently, these slow path * primitives should be optimized for maximum compactness. * * Ecmascript environment records (declarative and object) are represented * as internal objects with control keys. Environment records have a * parent record ("outer environment reference") which is represented by * the implicit prototype for technical reasons (in other words, it is a * convenient field). The prototype chain is not followed in the ordinary * sense for variable lookups. * * See identifier-handling.rst for more details on the identifier algorithms * and the internal representation. See function-objects.rst for details on * what function templates and instances are expected to look like. * * Care must be taken to avoid duk_tval pointer invalidation caused by * e.g. value stack or object resizing. * * TODO: properties for function instances could be initialized much more * efficiently by creating a property allocation for a certain size and * filling in keys and values directly (and INCREFing both with "bulk incref" * primitives. * * XXX: duk_hobject_getprop() and duk_hobject_putprop() calls are a bit * awkward (especially because they follow the prototype chain); rework * if "raw" own property helpers are added.

label: code-design

5321. If 'day' is NaN, returns NaN.

5322. * New length not lower than old length => no changes needed * (not even array allocation).

5323. * API oriented helpers

5324. [enum_target res key true]

5325. **comment:** XXX: we could save space by using _Target OR _This. If _Target, assume * this binding is undefined. If _This, assumes this binding is _This, and * target is also _This. One property would then be enough.

label: code-design

5326. ADDITIVE EXPRESSION

5327. macros

5328. default case does not exist, or no statements present * after default case: finish case evaluation

5329. use temp_next for tracking register allocations

5330. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer

5331. DUK_REGEXP_H_INCLUDED

5332. **comment:** XXX: add fastint support?

label: requirement

5333. FOUND

5334. * Additional control information is placed into the object itself * as internal properties to avoid unnecessary fields for the * majority of functions. The compiler tries to omit internal * control fields when possible. * * Function templates: * * { * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * } * * Function instances: * * { * length: 2, * prototype: { constructor: <func> }, * caller: <thrower>, * arguments: <thrower>, * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * _Varenv: <variable environment of closure>, * _Lexenv: <lexical environment of closure (if differs from _Varenv)> * } * * More detailed description of these properties can be found * in the documentation.

5335. initialize for debug prints, needed if sce==NULL

5336. start of valstack allocation

5337. Allocate a new duk_hbuffer of a certain type and return a pointer to it * (NULL on error). Write buffer data pointer to 'out_bufdata' (only if * allocation successful).

5338. Lookup 'key' from arguments internal 'map', delete mapping if found. * Used in E5 Section 10.6 algorithm for [[Delete]]. Note that the * variable/argument itself (where the map points) is not deleted.

5339. -> [... enum_target res trap handler target]

5340. derived types

5341. Awkward inclusion condition: drop out of compilation if not needed by any * call site: object hash part or probing stringtable.

5342. next instruction to execute (points to 'func' bytecode, stable pointer), NULL for native calls

5343. [offset littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)

5344. Must be a "pointer objec", i.e. class "Pointer"

5345. ES6 proxy

5346. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array * stack[4] = regexp res_obj (if is_regexp)

5347. value of dbg_exec_counter when we last did a Date-based check
5348. idx_retval unsigned
5349. no negative sign for zero
5350. **comment:** Two value cycle, see e.g. test-bi-date-tzoffset-basic-fi.js. * In these cases, favor a higher tzoffset to get a consistent * result which is independent of iteration count. Not sure if * this is a generically correct solution.
label: code-design
5351. 'thr' is the current thread, as no-one resumes except us and we * switch 'thr' in that case.
5352. lastIndex must be ignored for non-global regexps, but get the * value for (theoretical) side effects. No side effects can * really occur, because lastIndex is a normal property and is * always non-configurable for RegExp instances.
5353. clamp anything above nargs
5354. **comment:** XXX: this could be more compact by accessing the internal properties * directly as own properties (they cannot be inherited, and are not * externally visible).
label: code-design
5355. Get/set the current user visible size, without accounting for a dynamic * buffer's "spare" (= usable size).
5356. looping should never happen
5357. Validate byte read/write for virtual 'offset', i.e. check that the * offset, taking into account h->offset, is within the underlying * buffer size. This is a safety check which is needed to ensure * that even a misconfigured duk_hbufferobject never causes memory * unsafe behavior (e.g. if an underlying dynamic buffer changes * after being setup). Caller must ensure 'buf' != NULL.
5358. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.
5359. digit position is absolute, not relative
5360. c
5361. **comment:** Same test with volatiles
label: test
5362. because callstack_top > 0
5363. There's overlap: the desired end result is that * conceptually a copy is made to avoid "trampling" * of source data by destination writes. We make * an actual temporary copy to handle this case.
5364. **comment:** XXX: just use toString() for now; permitted although not recommended. * nargs==1, so radix is passed to toString().
label: code-design
5365. %d; only 16 bits are guaranteed
5366. 'this' value: * E5 Section 6.b.i ** The only (standard) case where the 'this' binding is non-null is when * (1) the variable is found in an object environment record, and * (2) that object environment record is a 'with' block. *
5367. This is important to ensure dynamic buffer data pointer is not * NULL (which is possible if buffer size is zero), which in turn * causes portability issues with e.g. memmove() and memcpy().
5368. Since we already started writing the reply, just emit nothing.
5369. * For top-level objects, 'length' property has the following * default attributes: non-writable, non-enumerable, non-configurable * (E5 Section 15). ** However, 'length' property for Array.prototype has attributes * expected of an Array instance which are different: writable, * non-enumerable, non-configurable (E5 Section 15.4.5.2). ** This is currently determined implicitly based on class; there are * no attribute flags in the init data.
5370. -> [... errhandler undefined(= this) errval]
5371. Reconfigure value stack for return to an Ecmascript function at 'act_idx'.
5372. * First pass. ** Gather variable/function declarations needed for second pass. * Code generated is dummy and discarded.
5373. valid pc range is [0, length[
5374. Current tracedata contains 2 entries per callstack entry.
5375. [... errval]
5376. **comment:** XXX: there is a small risk here: because the ISO 8601 parser is * very loose, it may end up parsing some datetime values which * would be better parsed with a platform specific parser.
label: code-design
5377. * Return (t - LocalTime(t)) in minutes: * * t - LocalTime(t) = t - (t + LocalTZA + DaylightSavingTA(t)) * = -(LocalTZA + DaylightSavingTA(t)) * * where DaylightSavingTA() is checked for time 't'. ** Note that the sign of the result is opposite to common usage, * e.g. for EE(S)T which normally is +2h or +3h from UTC, this * function returns -120 or -180. *
5378. **comment:** XXX: no need for indirect call
label: code-design
5379. XXX: the implementation now assumes "chained" bound functions, * whereas "collapsed" bound functions (where there is ever only * one bound function which directly points to a non-bound, final * function) would require a "collapsing" implementation which * merges argument lists etc here.
5380. The current implementation of localeCompare() is simply a codepoint * by codepoint comparison, implemented with a simple string compare * because UTF-8 should preserve codepoint ordering (assuming valid * shortest UTF-8 encoding). ** The specification requires that the return value must be related * to the sort order: e.g. negative means that 'this' comes before * 'that' in sort order. We assume an ascending sort order.
5381. valstack
5382. not protected, respect reclimit, not constructor
5383. Push the resulting view object and attach the ArrayBuffer.
5384. **comment:** XXX: this should be optimized to be a raw query and avoid valstack * operations if possible.
label: code-design
5385. unbalanced stack
5386. value from hook
5387. * If object has been marked finalizable, move it to the * "to be finalized" work list. It will be collected on * the next mark-and-sweep if it is still unreachable * after running the finalizer.
5388. shared pop
5389. * Struct defs
5390. nbytes zero size case * <-----> * [... | p | x | x | q] [... | p==q] * => [... | x | x | q] [...]
5391. **comment:** not really necessary
label: code-design
5392. [... target] -> [... target keys]
5393. Leading zero.
5394. '
5395. Make underlying buffer compact to match DUK_BW_GET_SIZE().
5396. first heap allocated, match bit boundary
5397. **comment:** XXX: more specific error classes?
label: code-design
5398. 25: setUTCMilliseconds
5399. * Intermediate value helpers
5400. Limits
5401. Encoded as surrogate pair, each encoding to 3 bytes for * 6 bytes total. Codepoints above U+10FFFF encode as 6 bytes * too, see duk_unicode_encode_cesu8().
5402. * Setup value stack: clamp to 'nargs', fill up to 'nregs' ** Value stack may either grow or shrink, depending on the * number of func registers and the number of actual arguments. * If nregs >= 0, func wants args clamped to 'nargs'; else it * wants all args (= 'num_stack_args').
5403. [sep ToObject(this) len sep result]
5404. * Variant 2 or 4
5405. Lightfunc, always success.
5406. step 11.c is relevant only if non-strict (checked in 11.c.ii)
5407. XX==

5408. **comment:** A plain buffer coerces to a Duktape.Buffer because it's the * object counterpart of the plain buffer value. But it might * still make more sense to produce an ArrayBuffer here?
label: code-design

5409. Force exponential format. Used for toExponential().

5410. Cannot wrap: each object is at least 8 bytes so count is * at most 1/8 of that.

5411. wipe the capture range made by the atom (if any)

5412. Anything else is not constructable.

5413. attrs in E5 Section 15.3.5.1

5414. * Use the index in the header to find the right starting point

5415. Tolerate act_caller->func == NULL which happens in * some finalization cases; treat like unknown caller.

5416. **comment:** XXX: remove DUK_CALL_FLAG_IGNORE_RECLIMIT flag: there's now the * reclimit bump?
label: code-design

5417. maximum length of standard format tag that we support

5418. buffer size is ≥ 1

5419. ignore encoding for now

5420. enable manually for dumping

5421. (?=

5422. require to be safe

5423. Parser part count.

5424. for side effects, result ignored

5425. Now we can check offset validity.

5426. Milliseconds between status notify and transport peeks.

5427. Matching separator index is used in the control table

5428. rehash even if old and new sizes are the same to get rid of * DELETED entries.

5429. * Two pass approach to processing the property descriptors. * On first pass validate and normalize all descriptors before * any changes are made to the target object. On second pass * make the actual modifications to the target object. * * Right now we'll just use the same normalize/validate helper * on both passes, ignoring its outputs on the first pass.

5430. * Stringify implementation.

5431. ToNumber() for a double is a no-op.

5432. default: char escape (two chars)

5433. * Hobject allocation. * * Provides primitive allocation functions for all object types (plain object, * compiled function, native function, thread). The object return is not yet * in "heap allocated" list and has a refcount of zero, so caller must careful.

5434. Check actual underlying buffers for validity and that they * cover the copy. No side effects are allowed after the check * so that the validity status doesn't change.

5435. * Special parser for character classes; calls callback for every * range parsed and returns the number of ranges present.

5436. negate result

5437. * Lightfunc

5438. **comment:** * Refzero handling is skipped entirely if (1) mark-and-sweep is * running or (2) execution is paused in the debugger. The objects * are left in the heap, and will be freed by mark-and-sweep or * eventual heap destruction. * * This is necessary during mark-and-sweep because refcounts are also * updated during the sweep phase (otherwise objects referenced by a * swept object would have incorrect refcounts) which then calls here. * This could be avoided by using separate decref macros in * mark-and-sweep; however, mark-and-sweep also calls finalizers which * would use the ordinary decref macros anyway and still call this * function. * * This check must be enabled also when mark-and-sweep support has been * disabled: the flag is also used in heap destruction when running * finalizers for remaining objects, and the flag prevents objects from * being moved around in heap linked lists.
label: code-design

5439. XXX: len $\geq 0x80000000$ won't work below because a signed type * is needed by qsort.

5440. Special handling for year sign.

5441. [arg1 ... argN obj length new_length]

5442. The algorithm in E5.1 Section 15.9.1.12 normalizes month, but * does not normalize the day-of-month (nor check whether or not * it is finite) because it's not necessary for finding the day * number which matches the (year,month) pair. * * We assume that duk__day_from_year() is exact here. * * Without an explicit infinity / NaN check in the beginning, * day_num would be a bogus integer here. * * It's possible for 'year' to be out of integer range here. * If so, we need to return NaN without integer overflow. * This fixes test-bug-setyear-overflow.js.

5443. * Node.js Buffer.prototype.slice([start], [end]) * ArrayBuffer.prototype.slice(begin, [end]) * TypedArray.prototype.slice(begin, [end]) * * The API calls are almost identical; negative indices are counted from end * of buffer, and final indices are clamped (allowing crossed indices). Main * differences: * * - Copy/view behavior; Node.js .slice() and TypedArray .subarray() create * views, ArrayBuffer .slice() creates a copy * * - Resulting object has a different class and prototype depending on the * call (or 'this' argument) * * - TypedArray .subarray() arguments are element indices, not byte offsets

5444. **comment:** When alloc_size == 0 the second allocation may not * actually exist.
label: code-design

5445. statements go here (if any) on next loop

5446. 'yield'

5447. eat trailing comma

5448. pruned

5449. * Declarative environment record. * * Identifiers can never be stored in ancestors and are * always plain values, so we can use an internal helper * and access the value directly with an duk_tval ptr. * * A closed environment is only indicated by it missing * the "book-keeping" properties required for accessing * register-bound variables.

5450. Note: there is no requirement that: 'thr->callstack_preventcount == 1' * like for yield.

5451. lookup results is ignored

5452. **comment:** * ToPrimitive() (E5 Section 9.1) * * ==> implemented in the API.
label: requirement

5453. resume caller must be an ecmascript func

5454. e.g. a = -4, b = 5 --> -4 - 5 + 1 / 5 --> -8 / 5 --> -1 * a = -5, b = 5 --> -5 - 5 + 1 / 5 --> -9 / 5 --> -1 * a = -6, b = 5 --> -6 - 5 + 1 / 5 --> -10 / 5 --> -2

5455. errors out if out of memory

5456. stack contains value (if requested), 'out_desc' is set

5457. no issues with empty memcmp()

5458. [str] -> [substr]

5459. func is NULL for lightfunc

5460. property attributes for identifier (relevant if value != NULL)

5461. Default function to format objects. Tries to use toLogString() but falls * back to toString(). Any errors are propagated out without catching.

5462. don't allow continue

5463. respect reclimit, not constructor

5464. * Notation: double underscore used for internal properties which are not * stored in the property allocation (e.g. '__valstack').

5465. 0x60-0x6f

5466. DUK_BUFOBJ_INT32ARRAY

5467. Caller has already eaten the first char so backtrack one byte.

5468. controls for minimum entry part growth

5469. temp accumulation buffer

5470. Unwind.

5471. * Object built-ins

5472. callstack index

5473. **comment:** XXX: The ES5/5.1/6 specifications require that the key in 'key in obj' * must be string coerced before the internal HasProperty() algorithm is * invoked.
A fast path skipping coercion could be safely implemented for * numbers (as number-to-string coercion has no side effects). For ES6 * proxy behavior, the trap 'key' argument must be in a string coerced * form (which is a shame).

label: code-design

5474. take last entry

5475. curr is data, desc is accessor

5476. **comment:** XXX: The incref macro takes a thread pointer but doesn't * use it right now.

label: code-design

5477. An object may be in heap_allocated list with a zero * refcount also if it is a temporary object created by * a finalizer; because finalization now runs inside * mark-and-sweep, such objects will not be queued to * refzero_list and will thus appear here with refcount * zero.

5478. Accept 32-bit decimal integers, no leading zeroes, signs, etc. * Leading zeroes are not accepted (zero index "0" is an exception * handled above).

5479. **comment:** Set: currently a finalizer is disabled by setting it to * undefined; this does not remove the property at the moment. * The value could be type checked to be either a function * or something else; if something else, the property could * be deleted.

label: code-design

5480. * Bitstream decoder

5481. statement id allocation (running counter)

5482. 'public'

5483. single match always

5484. Note: 'curr' refers to 'length' propdesc

5485. Use double parts, they tolerate unnormalized time. * * Note: DUK_DATE_IDX_WEEKDAY is initialized with a bogus value (DUK__PI_TZHOUR) * on purpose. It won't be actually used by duk.bi_date_get_timeval_from_dparts(), * but will make the value initialized just in case, and avoid any * potential for Valgrind issues.

5486. Steps 8-10 have been merged to avoid a "partial" variable.

5487. Used during heap destruction: don't actually run finalizers * because we're heading into forced finalization. Instead, * queue finalizable objects back to the heap_allocated list.

5488. duk_safe_call discipline

5489. **comment:** * Unicode tables containing ranges of Unicode characters in a * packed format. These tables are used to match non-ASCII * characters of complex productions by resorting to a linear * range-by-range comparison. This is very slow, but is expected * to be very rare in practical Ecmascript source code, and thus * compactness is most important. * * The tables are matched using uni_range_match() and the format * is described in src/extract_chars.py.

label: code-design

5490. **comment:** XXX: encoding is ignored now.

label: code-design

5491. use 'undefined' for value automatically

5492. [... obj ...]

5493. **comment:** Function pointers do not always cast correctly to void * * (depends on memory and segmentation model for instance), * so they coerce to NULL.

label: code-design

5494. Buffer length is bounded to 0xffff automatically, avoid compile warning.

5495. Integer division which floors also negative values correctly.

5496. consequence of above

5497. * Self tests to ensure execution environment is sane. Intended to catch * compiler/platform problems which cannot be detected at compile time.

5498. DUK_TAG_NUMBER would logically go here, but it has multiple 'tags'

5499. backtrack (safe)

5500. DUK__FLD_32BIT

5501. * Parse a RegExp token. The grammar is described in E5 Section 15.10. * Terminal constructions (such as quantifiers) are parsed directly here. * * 0xffffffffU is used as a marker for "infinity" in quantifiers. Further, * DUK__MAX_RE_QUANT_DIGITS limits the maximum number of digits that * will be accepted for a quantifier.

5502. **comment:** Note: reuse 'curr'

label: code-design

5503. * Convert and push final string.

5504. evaluate to final form (e.g. coerce GETPROP to code), throw away temp

5505. * Selftest code

5506. use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to the existing one)

5507. duk_tval_ptr for 'func' on stack (borrowed reference) or tv_func_copy

5508. * Encoding and decoding basic formats: hex, base64. * * These are in-place operations which may allow an optimized implementation. * * Base-64: <https://tools.ietf.org/html/rfc4648#section-4>

5509. Run mark-and-sweep a few times just in case (unreachable object * finalizers run already here). The last round must rescue objects * from the previous round without running any more finalizers. This * ensures rescued objects get their FINALIZED flag cleared so that * their finalizer is called once more in forced finalization to * satisfy finalizer guarantees. However, we don't want to run any * more finalizer because that'd required one more loop, and so on.

5510. required to keep recursion depth correct

5511. [... buf loop (proplist) (gap) holder ""]

5512. Get default hash part size for a certain entry part size.

5513. Maximum number of digits generated.

5514. [... func buf] -> [... buf]

5515. **comment:** XXX: could share code with duk_js_ops.c, duk_js_compare_helper

label: code-design

5516. Math.pow(+0,y) should be Infinity when y<0. NetBSD pow() * returns -Infinity instead when y is <0 and finite. The * if-clause also catches y == -Infinity (which works even * without the fix).

5517. Free strings in the stringable and any allocations needed * by the stringtable itself.

5518. e.g. DUK_OP_PREINCV

5519. If neutered must return 0; offset is zeroed during * neutering.

5520. array of declarations: [name1, val1, name2, val2, ...] * valN = (typeN) | (fnum << 8), where fnum is inner func number (0 for vars) * record function and variable declarations in pass 1

5521. breakpoints: [0,breakpoint_count[gc reachable

5522. **comment:** XXX: inefficient loop

label: code-design

5523. * Insert a jump offset at 'offset' to complete an instruction * (the jump offset is always the last component of an instruction). * The 'skip' argument must be computed relative to 'offset', * -without- taking into account the skip field being inserted. * * ... A B C ins X Y Z ... (ins may be a JUMP, SPLIT1/SPLIT2, etc) * => ... A B C ins SKIP X Y Z * * Computing the final (adjusted) skip value, which is relative to the * first byte of the next instruction, is a bit tricky because of the * variable length UTF-8 encoding. See doc/regexp.rst for discussion.

5524. * NFY <int: 5> <int: fatal> <str: msg> <str: filename> <int: linenumber> EOM

5525. Note: not necessary to check p against re_ctx->input_end: * the memory access is checked by duk_inp_get_cp(), while * valid compiled regexps cannot write a saved[] entry * which points to outside the string.

5526. Strings and ROM objects are never placed on the heap allocated list.

5527. [... val root ""] -> [... val val']

5528. **comment:** * JSON built-ins. * * See doc/json.rst. * * Codepoints are handled as duk_uint_fast32_t to ensure that the full * unsigned 32-bit range is supported. This matters to e.g. JX. * * Input parsing doesn't do an explicit end-of-input check at all. This is * safe: input string data is always NUL-terminated (0x00) and valid JSON * inputs never contain plain NUL characters, so that as long as syntax checks * are correct, we'll never read past the NUL. This approach reduces code size * and improves parsing performance, but it's critical that syntax checks are * indeed correct!

label: code-design

5529. stack type fits into 16 bits

5530. 'set'

5531. low to high

5532. required, NULL implies detached

5533. right associative

5534. covers +Infinity

5535. no size check is necessary

5536. $r < (*f2) * s < (*(\text{expt } b (- e)) 2) == b^{(-e)} * 2$ [if $b==2 \rightarrow b^{(1-e)} * m+ < -1 * m- < 1 * k < -0 * B < -B * \text{low_ok} < -\text{round} * \text{high_ok} < -\text{round}$

5537. **comment:** XXX: best behavior for real world compatibility?

label: code-design

5538. 34: setMonth

5539. 20: getSeconds

5540. refcount macros not defined without refcounting, caller must #ifdef now

5541. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is never NULL.

5542. DUK_MEMCMP() is guaranteed to return zero (equal) for zero length * inputs so no zero length check is needed.

5543. **comment:** XXX: shared parsing?

label: code-design

5544. Note: reject negative zero.

5545. Delete entry in Duktape.modLoaded[] and rethrow.

5546. already closed

5547. '\xffFormals'

5548. 1

5549. XXX: Multiple tv_func lookups are now avoided by making a local * copy of tv_func. Another approach would be to compute an offset * for tv_func from valstack bottom and recomputing the tv_func * pointer quickly as valstack + offset instead of calling duk_get_tval().

5550. DUK_TOK_NEQ

5551. **comment:** XXX: would be nice to omit this jump when the jump is not * reachable, at least in the obvious cases (such as the case * ending with a 'break'. * * Perhaps duk_parse_stmt() could provide some info on whether * the statement is a "dead end"? * * If implemented, just set pc_prevstmt to -1 when not needed.

label: code-design

5552. no shrink

5553. XXX: duk_to_int() ensures we'll get 8 lowest bits as * as input is within duk_int_t range (capped outside it).

5554. * Ecmascript bytecode

5555. Current require() function

5556. [... errobj]

5557. [... arr]

5558. -> [... escaped_source]

5559. func

5560. ignore fclose() error

5561. DUK_TOK_RETURN

5562. * <https://github.com/svaaraala/duktape/issues/127#issuecomment-77863473>

5563. **comment:** Note: not initializing all bytes is normally not an issue: Duktape won't * read or use the uninitialized bytes so valgrind won't issue warnings. * In some special cases a harmless valgrind warning may be issued though. * For example, the DumpHeap debugger command writes out a compiled function's * 'data' area as is, including any uninitialized bytes, which causes a * valgrind warning.

label: code-design

5564. XXX: to util

5565. Byte length would overflow.

5566. register declarations in first pass

5567. implicit this value always undefined for * declarative environment records.

5568. match_caps == 0 without regexps

5569. Value coercion (in stack): ToInteger(), E5 Section 9.4 * API return value coercion: custom

5570. thr->heap->lj.jmpbuf_ptr is checked by duk_err_longjmp() so we don't * need to check that here. If the value is NULL, a panic occurs because * we can't return.

5571. * Object finalizer

5572. **comment:** never shrinks; auto-adds DUK_VALSTACK_INTERNAL_EXTRA, which is generous

label: code-design

5573. -> [... proplist enum_obj key]

5574. elems in source and dest

5575. E5 Section 15.4.5.1, step 4

5576. **comment:** * Handle special cases (NaN, infinity, zero).

label: code-design

5577. DUK_USE_ROM_OBJECTS

5578. stats for current expression being parsed

5579. **comment:** maintain highest 'used' temporary, needed to figure out nregs of function

label: code-design

5580. DUK_IVAL_XXX

5581. **comment:** Serialize uncovered backing buffer as a null; doesn't * really matter as long we're memory safe.

label: code-design

5582. * Error, fatal, and panic handling.

5583. A number can be loaded either through a constant, using * LDINT, or using LDINT+LDINTX. LDINT is always a size win, * LDINT+LDINTX is not if the constant is used multiple times. * Currently always prefer LDINT+LDINTX over a double constant.

5584. guaranteed by string limits

5585. Slot A

5586. terminates expression; e.g. post-increment/-decrement

5587. ignore arguments, return undefined (E5 Section 15.3.4)

5588. Thread state.

5589. g...v

5590. [... |] or [... | errobj (M * undefined)] where M = num_stack_rets - 1

5591. omit

5592. always >= 0

5593. Fixed buffer, no zeroing because we'll fill all the data.

5594. Ecma-to-ecma call possible, may or may not be a tail call. * Avoid C recursion by being clever.

5595. terminal type: no depth check

5596. track cpos while scanning

5597. DUK_TOK_SEQ

5598. curr_pc synced by duk_handle_ecma_call_setup()

5599. **comment:** XXX: could accept numbers larger than 32 bits, e.g. up to 53 bits?

label: code-design

5600. accessor flag not encoded explicitly

5601. For now only needed by the debugger.
5602. * Bytecode instruction representation during compilation * Contains the actual instruction and (optionally) debug info.
5603. remove value
5604. * Forward declarations
5605. mm <- mp
5606. **comment:** LDINTX is not necessarily in FASTINT range, so * no fast path for now. * * XXX: perhaps restrict LDINTX to fastint range, wider * range very rarely needed.
 label: code-design
5607. the entries [new_e_next, new_e_size_adjusted[are left uninitialized on purpose (ok, not gc reachable)
5608. rethrow
5609. * Accessor macros for function specific data areas
5610. Argument validation and func/args offset.
5611. day number for Jan 1 since epoch
5612. DUK_HOBJECT_FLAG_NATIVEFUNCTION varies
5613. compact
5614. **comment:** XXX: this generates quite large code - perhaps select the error * class based on the code and then just use the error 'name'?
 label: code-design
5615. require a short (8-bit) reg/const which fits into bytecode B/C slot
5616. **comment:** Function.prototype.bind() should never let this happen, * ugly error message is enough.
 label: code-design
5617. "123." is allowed in some formats
5618. Apply ToNumber() to specified index; if ToInteger(val) in [0,99], add * 1900 and replace value at idx_val.
5619. NB: must accept reserved words as property name
5620. no array part
5621. * Misc
5622. error message doesn't matter, ignored anyway
5623. Resolve 'res' directly into the LHS binding, and use * that as the expression value if safe. If not safe, * resolve to a temp/const and copy to LHS.
5624. * Found existing own (non-inherited) plain property. * Do an access control check and update in place.
5625. Currently nothing to free
5626. **comment:** Convenience copies from heap/vm for faster access.
 label: code-design
5627. Reply with tvals pushed by request callback
5628. If out of catchstack, cat = thr->catchstack - 1; * new_cat_top will be 0 in that case.
5629. **comment:** XXX: Could just lookup . toJSON() and continue in fast path, * as it would almost never be defined.
 label: code-design
5630. **comment:** * Raw write/read macros for big endian, unaligned basic values. * Caller ensures there's enough space. The macros update the pointer * argument automatically on resizes. The idiom seems a bit odd, but * leads to compact code.
 label: code-design
5631. **comment:** * "arguments" and "caller" must be mapped to throwers for strict * mode and bound functions (E5 Section 15.3.5). * * XXX: This is expensive to have for every strict function instance. * Try to implement as virtual properties or on-demand created properties.
 label: code-design
5632. 'index'
5633. index is above internal string length -> property is fully normal
5634. LF line terminator; CR LF and Unicode lineterms are handled in slow path
5635. * Misc helpers
5636. expt== -1023 -> bitstart=0 (leading 1); * expt== -1024 -> bitstart=-1 (one left of leading 1), etc
5637. bound chain resolved
5638. base
5639. Line format: <time> <entryLev> <loggerName>: <msg>
5640. NEXTENUM jump slot: executed when enum finished
5641. **comment:** prototype should be last, for readability
 label: code-design
5642. **comment:** XXX: direct implementation
 label: requirement
5643. Error codes.
5644. multi-character sets not allowed as part of ranges, see * E5 Section 15.10.2.15, abstract operation CharacterRange.
5645. * Property lookup
5646. Constants for duk_hashstring().
5647. roughly 1 kB
5648. advance by one character (code point) and one char_offset
5649. no need to coerce
5650. remove key and value
5651. A regexp token value.
5652. large context; around 2kB now
5653. lexer character window helpers
5654. at index 1
5655. trivial cases
5656. duplicate/invalid key checks; returns 1 if syntax error
5657. Here we could remove references to built-ins, but it may not be * worth the effort because built-ins are quite likely to be shared * with another (unterminated) thread, and terminated threads are also * usually garbage collected quite quickly. Also, doing DECREFs * could trigger finalization, which would run on the current thread * and have access to only some of the built-ins. Garbage collection * deals with this correctly already.
5658. Name will be filled from function expression, not by caller. * This case is used by Function constructor and duk_compile() * API with the DUK_COMPILE_FUNCTION option.
5659. original target at entry_top - 1
5660. not emitted
5661. **comment:** XXX: duk_ssize_t would be useful here
 label: code-design
5662. comp_ctx->lex.input and comp_ctx->lex.input_length filled by caller
5663. **comment:** XXX: pre-checks (such as no duplicate keys)
 label: code-design
5664. To handle deeper indents efficiently, make use of copies we've * already emitted. In effect we can emit a sequence of 1, 2, 4, * 8, etc copies, and then finish the last run. Byte counters * avoid multiply with gap_len on every loop.
5665. **comment:** XXX: TRYCATCH handling should be reworked to avoid creating * an explicit scope unless it is actually needed (e.g. function * instances or eval is executed inside the catch block). This * rework is not trivial because the compiler doesn't have an * intermediate representation. When the rework is done, the * opcode format can also be made more straightforward.
 label: code-design
5666. Because new_size != 0, if condition doesn't need to be * (p != NULL || new_size == 0).
5667. * Init lexer

5668. '%'

5669. * This test fails on an exotic ARM target; double-to-uint * cast is incorrectly clamped to -signed- int highest value. * *
 <https://github.com/svaarala/duktape/issues/336>

5670. DUK_TOK_NUMBER

5671. IEEE exp without bias

5672. 31 bits

5673. call handling

5674. not exposed

5675. No need to check for size of bp_active list, * it's always larger than maximum number of * breakpoints.

5676. **comment:** * Restart execution by reloading thread state. * * Note that 'thr' and any thread configuration may have changed, * so all local variables are suspect and we need to reinitialize. * * The number of local variables should be kept to a minimum: if * the variables are spilled, they will need to be loaded from * memory anyway. * * Any 'goto restart_execution;' code path in opcode dispatch must * ensure 'curr_pc' is synced back to act->curr_pc before the goto * takes place. * * The interpreter must be very careful with memory pointers, as * many pointers are not guaranteed to be 'stable' and may be * reallocated and relocated on-the-fly quite easily (e.g. by a * memory allocation or a property access). * * The following are assumed to have stable pointers: * - the current thread * - the current function * - the bytecode, constant table, inner function table of the * current function (as they are a part of the function allocation) * * The following are assumed to have semi-stable pointers: * - the current activation entry: stable as long as callstack * is not changed (reallocated by growing or shrinking), or * by any garbage collection invocation (through finalizers) * - Note in particular that ANY DECREF can invalidate the * activation pointer, so for the most part a fresh lookup * is required * * The following are not assumed to have stable pointers at all: * - the value stack (registers) of the current thread * - the catch stack of the current thread * * See execution.rst for discussion.

label: code-design

5677. For internal use: get array part value

5678. 10xx xxxx -> invalid

5679. Compiler is responsible for selecting property flags (configurability, * writability, etc).

5680. _Varmap is dense

5681. DUK_TOK_BAND

5682. **comment:** 0x00 ... 0x7f: as is * 0x80: escape generically * 0x81: slow path * 0xa0 ... 0xff: backslash + one char

label: code-design

5683. * Exposed matcher function which provides the semantics of RegExp.prototype.exec(). * * RegExp.prototype.test() has the same semantics as exec() but does not return the * result object (which contains the matching string and capture groups). Currently * there is no separate test() helper, so a temporary result object is created and * discarded if test() is needed. This is intentional, to save code space. * * Input stack: [... re_obj input] * Output stack: [... result]

5684. * Buffers have no internal references. However, a dynamic * buffer has a separate allocation for the buffer. This is * freed by duk_heap_free_heapobj_raw().

5685. indirect eval

5686. Numbers half-way between integers must be rounded towards +Infinity, * e.g. -3.5 must be rounded to -3 (not -4). When rounded to zero, zero * sign must be set appropriately. E5.1 Section 15.8.2.15. * * Note that ANSI C round() is "round to nearest integer, away from zero", * which is incorrect for negative values. Here we make do with floor().

5687. * Some assertions (E5 Section 13.2).

5688. Shared lenient buffer length clamping helper. No negative indices, no * element/byte shifting.

5689. **comment:** XXX: some code might benefit from DUK__SETTEMP_IFTEMP(ctx,x)

label: code-design

5690. preincrement and predecrement

5691. Value would normally be omitted, replace with 'null'.

5692. e.g. require('/foo'), empty terms not allowed

5693. string is internal

5694. * Parse regexp Disjunction. Most of regexp compilation happens here. * * Handles Disjunction, Alternative, and Term productions directly without * recursion. The only constructs requiring recursion are positive/negative * lookaheads, capturing parentheses, and non-capturing parentheses. * * The function determines whether the entire disjunction is a 'simple atom' * (see doc/regexp.rst discussion on 'simple quantifiers') and if so, * returns the atom character length which is needed by the caller to keep * track of its own atom character length. A disjunction with more than one * alternative is never considered a simple atom (although in some cases * that might be the case). * * Return value: simple atom character length or < 0 if not a simple atom. * Appends the bytecode for the disjunction matcher to the end of the temp * buffer. * * Regexp top level structure is: * * Disjunction = Term* * | Term* | Disjunction * * Term = Assertion * | Atom * | Atom Quantifier * * An empty Term sequence is a valid disjunction alternative (e.g. /|||c||/). * * Notes: * * * Tracking of the 'simple-ness' of the current atom vs. the entire * disjunction are separate matters. For instance, the disjunction * may be complex, but individual atoms may be simple. Furthermore, * simple quantifiers are used whenever possible, even if the * disjunction as a whole is complex. * * * The estimate of whether an atom is simple is conservative now, * and it would be possible to expand it. For instance, captures * cause the disjunction to be marked complex, even though captures * -can- be handled by simple quantifiers with some minor modifications. * * * Disjunction 'tainting' as 'complex' is handled at the end of the * main for loop collectively for atoms. Assertions, quantifiers, * and '| tokens need to taint the result manually if necessary. * Assertions cannot add to result char length, only atoms (and * quantifiers) can; currently quantifiers will taint the result * as complex though.

5695. Get a pointer to the current buffer contents (matching current allocation * size). May be NULL for zero size dynamic/external buffer.

5696. success: leave varname in stack

5697. safe

5698. [... closure template len_value]

5699. Note: negative pc values are ignored when patching jumps, so no explicit checks needed

5700. [... func this (crud) retval]

5701. DUK_USE_DATE_PRS_STRTIME

5702. **comment:** Global case is more complex.

label: code-design

5703. max, excl. varargs marker

5704. should be a safe way to compute this

5705. not an error

5706. prev_token slot2

5707. side effects, perform in-place

5708. **comment:** XXX: turkish / azeri

label: code-design

5709. current variable environment (may be NULL if delayed)

5710. May happen in some out-of-memory corner cases.

5711. * Store lexer position for a later rewind

5712. -> [... this timeval]

5713. Avoid using RegExp.prototype methods, as they're writable and * configurable and may have been changed.

5714. Index validation is strict, which differs from duk_equals(). * The strict behavior mimics how instanceof itself works, e.g. * it is a TypeError if rval is not a - callable- object. It would * be somewhat inconsistent if rval would be allowed to be * non-existent without a TypeError.

5715. prev_token slot1

5716. Note1

5717. **comment:** When DUK_USE_HEAPPTR16 (and DUK_USE_REFCOUNT16) is in use, the * struct won't align nicely to 4 bytes. This 16-bit extra field * is added to make the alignment clean; the field can be used by * heap objects when 16-bit packing is used. This field is now * conditional to DUK_USE_HEAPPTR16 only, but it is intended to be * used with DUK_USE_REFCOUNT16 and DUK_USE_DOUBLE_LINKED_HEAP; * this only matter to low memory environments anyway.

label: code-design

5718. DUK_TOK_IMPORT

5719. * Native call.

5720. input offset tracking
5721. -> [... closure template newobj closure]
5722. fixed arg count: value
5723. chain jumps for case * evaluation and checking
5724. Called for handling both a longjmp() with type DUK_LJ_TYPE_YIELD and * when a RETURN opcode terminates a thread and yields to the resumer.
5725. function call
5726. full 3-byte -> 4-char conversions
5727. regexp compilation limits
5728. **comment:** Current PC, accessed by other functions through thr->ptr_to_curr_pc. * Critical for performance. It would be safest to make this volatile, * but that eliminates performance benefits; aliasing guarantees * should be enough though.
 label: code-design
5729. 0x00: slow path * other: as is
5730. **comment:** XXX: this could be optimized; there is only one call site now though
 label: code-design
5731. reject 'in' token (used for for-in)
5732. Data area, fixed allocation, stable data ptrs.
5733. list of conversion specifiers that terminate a format tag; * this is unfortunately guesswork.
5734. Preliminaries for __proto__ and setPrototypeOf (E6 19.1.2.18 steps 1-4); * magic: 0=setter call, 1=Object.setPrototypeOf
5735. DUK_LEXER_H_INCLUDED
5736. type bit mask: types which certainly produce 'undefined'
5737. XXX: for threads, compact value stack, call stack, catch stack?
5738. **comment:** Lightfuncs are currently supported by coercing to a temporary * Function object; changes will be allowed (the coerced value is * extensible) but will be lost.
 label: code-design
5739. second, parse flags
5740. * Not found: write to global object (non-strict) or ReferenceError * (strict); see E5 Section 8.7.2, step 3.
5741. strict mode restrictions (E5 Section 12.2.1)
5742. **comment:** Forget the previous allocation, setting size to 0 and alloc to * NULL. Caller is responsible for freeing the previous allocation. * Getting the allocation and clearing it is done in the same API * call to avoid any chance of a realloc.
 label: code-design
5743. [regexp]
5744. \xffFinalizer'
5745. **comment:** The entries can either be register numbers or 'null' values. * Thus, no need to DECREF them and get side effects. DECREF'ing * the keys (strings) can cause memory to be freed but no side * effects as strings don't have finalizers. This is why we can * rely on the object properties not changing from underneath us.
 label: code-design
5746. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.
5747. For all combinations: +0 < +0, +0 < -0, -0 < +0, -0 < -0, * steps e, f, and g.
5748. output is never longer than input during resolution
5749. **comment:** XXX: make fast paths optional for size minimization?
 label: code-design
5750. **comment:** XXX: handling for array part missing now; this doesn't affect * compliance but causes array entry writes using defineProperty() * to always abandon array part.
 label: code-design
5751. Fast path the binary case
5752. * Unicode helpers
5753. **comment:** XXX: zero assumption
 label: code-design
5754. * Assertions after
5755. r <- r * s <- (* s B) * m+ <- m+ * m- <- m- * k <- (+ k 1)
5756. 'Math'
5757. 1e8: protects against deeply nested inner functions
5758. insert ranges instruction, range count patched in later
5759. * Validate and convert argument property descriptor (an Ecmascript * object) into a set of defprop_flags and possibly property value, * getter, and/or setter values on the value stack. * * Lightfunc set/get values are coerced to full Functions.
5760. tc1 = false, tc2 = true
5761. after this, return paths should 'goto finished' for decrement
5762. obj_index
5763. zero-width non-joiner
5764. activation has active breakpoint(s)
5765. Constants for built-in string data depacking.
5766. Need to set curr_token.t because lexing regexp mode depends on current * token type. Zero value causes "allow regexp" mode.
5767. start (incl) and end (excl) of trimmed part
5768. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.
5769. interpret e.g. '09' as '0', not NaN
5770. * splice()
5771. no argument given -> leave components untouched
5772. **comment:** This fast path is pretty marginal in practice. * XXX: candidate for removal.
 label: code-design
5773. character represents itself
5774. eat 'return'
5775. **comment:** If the thr != NULL, the thr may still be in the middle of * initialization. * XXX: Improve the thread viability test.
 label: code-design
5776. [value offset end]
5777. Relies on NULL encoding to zero.
5778. **comment:** XXX: the current implementation works but is quite clunky; it compiles * to almost 1.4kB of x86 code so it needs to be simplified (better approach, * shared helpers, etc). Some ideas for refactoring: * * - a primitive to convert a string into a regexp matcher (reduces matching * code at the cost of making matching much slower) * - use replace() as a basic helper for match() and split(), which are both * much simpler * - API call to get_prop and to_boolean
 label: code-design
5779. * Compilation and evaluation
5780. Note: Identifier rejects reserved words
5781. -> [obj trap_result]
5782. * Unicode letter is now taken to be the categories: * * Lu, Ll, Lt, Lm, Lo * * (Not sure if this is exactly correct.) * * The ASCII fast path consists of: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z']
5783. caller ensures
5784. * Not found (in registers or record objects). Declare * to current variable environment.
5785. **comment:** 'this' binding is not needed here

label: requirement

5786. no need to check callable; duk_call() will do that

5787. mark finalizer work list as reachability roots

5788. Initial '{' has been checked and eaten by caller.

5789. \xffMap'

5790. The directive prologue flag is cleared by default so that it is * unset for any recursive statement parsing. It is only "revived" * if a directive is detected. (We could also make directives only * allowed if 'allow_source_elem' was true.)

5791. 'NaN'

5792. unsigned 31-bit range

5793. DUK_USE_EXPLICIT_NULL_INIT

5794. valgrind whine without this

5795. **comment:** * Helper for calling a user error handler. * * 'thr' must be the currently active thread; the error handler is called * in its context. The valstack of 'thr' must have the error value on * top, and will be replaced by another error value based on the return * value of the error handler. * * The helper calls duk_handle_call() recursively in protected mode. * Before that call happens, no longjmps should happen; as a consequence, * we must assume that the valstack contains enough temporary space for * arguments and such. * * While the error handler runs, any errors thrown will not trigger a * recursive error handler call (this is implemented using a heap level * flag which will "follow" through any coroutines resumed inside the * error handler). If the error handler is not callable or throws an * error, the resulting error replaces the original error (for Duktape * internal errors, duk_error_throw.c further substitutes this error with * a DoubleError which is not ideal). This would be easy to change and * even signal to the caller. * * The user error handler is stored in 'Duktape.errCreate' or * 'Duktape.errThrow' depending on whether we're augmenting the error at * creation or throw time. There are several alternatives to this approach, * see doc/error-objects.rst for discussion. * * Note: since further longjmp(s) may occur while calling the error handler * (for many reasons, e.g. a labeled 'break' inside the handler), the * caller can make no assumptions on the thr->heap->lj state after the * call (this affects especially duk_error_throw.c). This is not an issue * as long as the caller writes to the lj state only after the error handler * finishes.**label:** code-design

5796. * Exposed calls

5797. key encountered as a setter

5798. Don't need to sync curr_pc here; duk_new() will do that * when it augments the created error.

5799. one i_step over

5800. Shared prefix for all buffer types.

5801. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. * * Delayed creation (on demand) is handled in duk_js_var.c.

5802. Reject a proxy object as the target because it would need * special handler in property lookups. (ES6 has no such restriction)

5803. must restore reliably before returning

5804. * Basic context initialization. * * Some init values are read from the bytecode header * whose format is (UTF-8 codepoints): * * uint flags * uint nsaved (even, 2n+2 where n = num captures)

5805. return 'res_arr' or 'null'

5806. **comment:** This would be nice, but parsing is faster without resetting the * value slots. The only side effect is that references to temporary * string values may linger until lexing is finished; they're then * freed normally.**label:** code-design

5807. break jump

5808. "123"

5809. Note: the only yield-preventing call is Duktape.Thread.yield(), hence check for 1, not 0

5810. range check not necessary because all 4-bit values are mapped

5811. floor(1.15 * (1 << 10))

5812. expect_eof

5813. -- p_insert -- p_curr * v v * | ... | insert | ... | curr

5814. * Stack constants

5815. call: this(fmt(arg)

5816. Combined step 11 (empty string special case) and 14-15.

5817. w/o refcounting

5818. Example: d=3.5, t=0.5 -> ret = (3 + 1) & 0xfe = 4 & 0xfe = 4 * Example: d=4.5, t=0.5 -> ret = (4 + 1) & 0xfe = 5 & 0xfe = 4

5819. Write fully.

5820. Assume IEEE round-to-even, so that shorter encoding can be used * when round-to-even would produce correct result. By removing * this check (and having low_ok == high_ok == 0) the results would * still be accurate but in some cases longer than necessary.

5821. object established using Function.prototype.bind()

5822. * Heap object refcount finalization. * * When an object is about to be freed, all other objects it refers to must * be decref'd. Refcount finalization does NOT free the object or its inner * allocations (mark-and-sweep shares these helpers), it just manipulates * the refcounts. * * Note that any of the decref's may cause a refcount to drop to zero, BUT * it will not be processed inline; instead, because refzero is already * running, the objects will just be queued to refzero list and processed * later. This eliminates C recursion.

5823. output radix

5824. resume check from proxy target

5825. native function properties

5826. 1e8

5827. Lightfunc flags packing and unpacking.

5828. Helpers exposed for internal use

5829. * Hash function duk_util_hashbytes(). * * Currently, 32-bit MurmurHash2. * * Don't rely on specific hash values; hash function may be endianness * dependent, for instance.

5830. The final object may be a normal function or a lightfunc. * We need to re-lookup tv_func because it may have changed * (also value stack may have been resized). Loop again to * do that; we're guaranteed not to come here again.

5831. **comment:** For objects JSON.stringify() only looks for own, enumerable * properties which is nice for the fast path here. * * For arrays JSON.stringify() uses [[Get]] so it will actually * inherit properties during serialization! This fast path * supports gappy arrays as long as there's no actual inherited * property (which might be a getter etc). * * Since recursion only happens for objects, we can have both * recursion and loop checks here. We use a simple, depth-limited * loop check in the fast path because the object-based tracking * is very slow (when tested, it accounted for 50% of fast path * execution time for input data with a lot of small objects!).**label:** code-design

5832. assertion omitted

5833. -> [... obj finalizer]

5834. does not fit into an uchar

5835. need to normalize, may even cancel to 0

5836. Callee/caller are throwers and are not deletable etc. They * could be implemented as virtual properties, but currently * there is no support for virtual properties which are accessors * (only plain virtual properties). This would not be difficult * to change in duk_object_props, but we can make the throwers * normal, concrete properties just as easily. * * Note that the specification requires that the *same* thrower * built-in object is used here! See E5 Section 10.6 main * algorithm, step 14, and Section 13.2.3 which describes the * thrower. See test case test-arguments-throwers.js.

5837. used for array, stack, and entry indices

5838. [... name reg_bind]

5839. Select copy mode. Must take into account element * compatibility and validity of the underlying source * buffer.

5840. invalid value which never matches

5841. Arrays never have other exotic behaviors.

5842. 0xxx xxxx [7 bits]

5843. [... env target]
5844. Setup builtins from ROM objects. All heaps/threads will share * the same readonly objects.
5845. **comment:** alloc size in elements
 label: code-design
5846. * Init remaining result fields * * 'nregs' controls how large a register frame is allocated. * * 'nargs' controls how many formal arguments are written to registers: * r0, ... r(nargs-1). The remaining registers are initialized to * undefined.
5847. note: caller 'sp' is intentionally not updated here
5848. Underlying buffer (refcounted), may be NULL.
5849. actual chars dropped (not just NUL term)
5850. free-form
5851. formals for 'func' (may be NULL if func is a C function)
5852. **comment:** bytes of indent available for copying
 label: code-design
5853. * Valstack manipulation for results.
5854. [obj key desc value get set curr_value varname]
5855. * Add ._Tracedata to an error on the stack top.
5856. force executor restart to recheck breakpoints; used to handle function returns (see GH-303)
5857. Lexer context. Same context is used for Ecmascript and Regexp parsing.
5858. **comment:** XXX: spilling
 label: code-design
5859. Note: rely on index ordering
5860. * Get enumerated keys in an Ecmascript array. Matches Object.keys() behavior * described in E5 Section 15.2.3.14.
5861. NB: duk_lexer_getpoint() is a macro only
5862. dangerous: must only lower (temp_max not updated)
5863. x <- x + y, use t as temp
5864. * Init argument related properties
5865. add BC*2^16
5866. initial stringtable size, must be prime and higher than DUK_UTIL_MIN_HASH_PRIME
5867. char format: use int
5868. Clamping only necessary for 32-bit ints.
5869. maximum size check is handled by callee
5870. -> [... val root val]
5871. Note: start_offset/end_offset can still be < 0 here.
5872. cat_idx catcher is kept, even for finally
5873. "null", "true", and "false" are always reserved words. * Note that "get" and "set" are not!
5874. AssignmentExpression
5875. DUK_TOK_MOD
5876. **comment:** XXX: this is pointless here because pass 1 is throw-away
 label: code-design
5877. Possibly truncated; there is no explicit truncation * marker so this is the best we can do.
5878. 'fatal'
5879. * Plain, boring reachable object.
5880. value of re_ctx->captures at start of atom
5881. * Find an existing key from entry part either by linear scan or by * using the hash index (if it exists). * * Sets entry index (and possibly the hash index) to output variables, * which allows the caller to update the entry and hash entries in-place. * If entry is not found, both values are set to -1. If entry is found * but there is no hash part, h_idx is set to -1.
5882. pc_label
5883. 'configurable'
5884. * Object prototype
5885. 'this' binding exists
5886. XXX: direct manipulation, or duk_replace_tval()
5887. fill window
5888. **comment:** * Helper for handling a "bound function" chain when a call is being made. * * Follows the bound function chain until a non-bound function is found. * Prepends the bound arguments to the value stack (at idx_func + 2), * updating 'num_stack_args' in the process. The 'this' binding is also * updated if necessary (at idx_func + 1). Note that for constructor calls * the 'this' binding is never updated by [[BoundThis]]. * * XXX: bound function chains could be collapsed at bound function creation * time so that each bound function would point directly to a non-bound * function. This would make call time handling much easier.
 label: code-design
5889. * Constructor arguments are currently somewhat compatible with * (keep it that way if possible): * * http://nodejs.org/api/buffer.html * * Note that the ToBuffer() coercion (duk_to_buffer()) does NOT match * the constructor behavior.
5890. * Emit compiled regexp header: flags, ncaptures * (insertion order inverted on purpose)
5891. * Logging support
5892. [key result]
5893. unchanged by Duktape.Thread.resume()
5894. arguments object would be accessible; note that shadowing * bindings are arguments or function declarations, neither * of which are deletable, so this is safe.
5895. Return 1: got EOM
5896. include removed: duk_internal.h
5897. NOTE: length may be zero
5898. SCANBUILD: NULL pointer dereference, doesn't actually trigger, * asserted above.
5899. Alignment guarantee
5900. !DUK_USE_PREFER_SIZE
5901. Note1: the specification doesn't require matching a time form with * just hours ("HH"), but we accept it here, e.g. "2012-01-02T12Z". * * Note2: the specification doesn't require matching a timezone offset * with just hours ("HH"), but accept it here, e.g. "2012-01-02T03:04:05+02"
5902. * Mark the heap.
5903. silence scan-build warning
5904. varname is popped by above code
5905. Estimating the result size beforehand would be costly, so * start with a reasonable size and extend as needed.
5906. Decode failed.
5907. * Init stringcache
5908. [... constructor arg1 ... argN final_cons]
5909. **comment:** XXX: lithuanian, explicit dot rules
 label: code-design
5910. XXX: similar coercion issue as in DUK_TOK_PERIOD
5911. **comment:** Wipe capture range and save old values for backtracking. * * XXX: this typically happens with a relatively small idx_count. * It might be useful to handle cases where the count is small * (say <= 8) by saving the values in stack instead. This would * reduce memory churn and improve performance, at the cost of a * slightly higher code footprint.
 label: code-design
5912. source out-of-bounds (but positive)
5913. 'with' binding has no catch clause, so can't be here unless a normal try-catch

5914. second incref for the entry reference
5915. **comment:** XXX: declvar takes an duk_tval pointer, which is awkward and * should be reworked.
 label: code-design
5916. Get.
5917. DUK_TOK_PRIVATE
5918. randomized pivot selection
5919. * Check whether the property already exists in the prototype chain. * Note that the actual write goes into the original base object * (except if an accessor property captures the write).
5920. These are not needed when only Duktape.Buffer is supported.
5921. duk_hobject specific fields.
5922. [val] -> []
5923. Map DUK_HBUFFEROBJECT_ELEM_xxx to duk_hobject class number. * Sync with duk_hbufferobject.h and duk_hobject.h.
5924. * Shared exit path
5925. force_exponential
5926. As an initial implementation, write flush after every EOM (and the * version identifier). A better implementation would flush only when * Duktape is finished processing messages so that a flush only happens * after all outbound messages are finished on that occasion.
5927. DUK_INVALID_INDEX won't be accepted as a valid index.
5928. * Heap thread object representation. ** duk_hthread is also the 'context' (duk_context) for exposed APIs * which mostly operate on the topmost frame of the value stack.
5929. DUK_USE_BUFFEROBJECT_SUPPORT
5930. As an initial implementation, read flush after exiting the message * loop. If transport is broken, this is a no-op (with debug logs).
5931. * Figure out how generated digits match up with the mantissa, * and then perform rounding. If mantissa overflows, need to * recompute the exponent (it is bumped and may overflow to * infinity). ** For normal numbers the leading '1' is hidden and ignored, * and the last bit is used for rounding: ** rounding pt * <-----52---->| * 1 x x x x ... x x x x |y ==> x x x x ... x x x x * * For denormals, the leading '1' is included in the number, * and the rounding point is different: ** rounding pt * <--52 or less--->| * 1 x x x x ... x|x|x x y ==> 0 0 ... 1 x x ... x x * * The largest denormals will have a mantissa beginning with * a '1' (the explicit leading bit); smaller denormals will * have leading zero bits. * * If the exponent would become too high, the result becomes * Infinity. If the exponent is so small that the entire * mantissa becomes zero, the result becomes zero. * * Note: the Dragon4 'k' is off-by-one with respect to the IEEE * exponent. For instance, k==0 indicates that the leading '1' * digit is at the first binary fraction position (0.1xxx...); * the corresponding IEEE exponent would be -1.
5932. string hash
5933. Relookup in case coerce_func() has side effects, e.g. ends up coercing an object
5934. 0 = no update
5935. regexp opcodes
5936. 'function'
5937. recursion limit
5938. utf-8 validation ensures these
5939. count, not including sep
5940. string data is external (duk_hstring_external)
5941. currently paused: talk with debug client until step/resume
5942. duk_concat() coerces arguments with ToString() in correct order
5943. 2 to 11
5944. guaranteed to finish
5945. const flag for B
5946. one i_step added at top of loop
5947. 'res' may be NULL if new allocation size is 0.
5948. **comment:** Shared entry code for many Array built-ins. Note that length is left * on stack (it could be popped, but that's not necessary).
 label: code-design
5949. thread
5950. assume PC is at most 32 bits and non-negative
5951. 1 0 <2 bits>
5952. **comment:** * Shared handling for logical AND and logical OR. ** args = (truthval << 8) + rbp * * Truthval determines when to skip right-hand-side. * For logical AND truthval=1, for logical OR truthval=0. ** See doc/compiler.rst for discussion on compiling logical * AND and OR expressions. The approach here is very simplistic, * generating extra jumps and multiple evaluations of truth values, * but generates code on-the-fly with only local back-patching. ** Both logical AND and OR are syntactically left-associated. * However, logical ANDs are compiled as right associative * expressions, i.e. "A && B && C" as "A && (B && C)", to allow * skip jumps to skip over the entire tail. Similarly for logical OR.
 label: code-design
5953. NULL obj->p is OK
5954. * In strict mode E5 protects 'eval' and 'arguments' from being * assigned to (or even declared anywhere). Attempt to do so * should result in a compile time SyntaxError. See the internal * design documentation for details. ** Thus, we should never come here, run-time, for strict code, * and name 'eval' or 'arguments'.
5955. Assume that thr->valstack_bottom has been set-up before getting here.
5956. len: 9
5957. * E5 Section 11.4.9
5958. deal with weak references first
5959. duk_push_sprintf constants
5960. negative: dst info not available
5961. topmost element is the result array already
5962. Fast path, handle units with just actual encoding characters.
5963. out of spec, must be configurable
5964. * Debug logging after adjustment.
5965. * Pushers
5966. **comment:** * Note: each API operation potentially resizes the callstack, * so be careful to re-lookup after every operation. Currently * these is no issue because we don't store a temporary 'act' * pointer at all. (This would be a non-issue if we operated * directly on the array part.)
 label: code-design
5967. implicit_return_value
5968. XXX: direct valstack write
5969. [...] source? filename]
5970. No resize has occurred so temp_desc->e_idx is still OK
5971. lookup for 0x000a above assumes shortest encoding now
5972. XXX: E5.1 Section 11.1.4 coerces the final length through * ToUint32() which is odd but happens now as a side effect of * 'arr_idx' type.
5973. Read tvals from the message and push them onto the valstack, * then call the request callback to process the request.
5974. true, despite side effect resizes
5975. the stack is unbalanced here on purpose; we only rely on the * initial two values: [name this].
5976. [start end str]
5977. source is eval code (not global)
5978. switch initial byte
5979. Note: pc is unsigned and cannot be negative
5980. * Casting
5981. **comment:** * User declarations, e.g. prototypes for user functions used by Duktape * macros. Concretely, if DUK_USE_PANIC_HANDLER is used and the macro * value calls a user function, it needs to be declared for Duktape * compilation to avoid warnings.

label: code-design

5982. Longjmp handling has restored jmpbuf_ptr.
5983. * Round a number upwards to a prime (not usually the nearest one). * * Uses a table of successive 32-bit primes whose ratio is roughly * constant. This keeps the relative upwards 'rounding error' bounded * and the data size small. A simple 'predict-correct' compression is * used to compress primes to one byte per prime. See genhashsizes.py * for details. * * The minimum prime returned here must be coordinated with the possible * probe sequence steps in duk_hobject and duk_heap stringtable.
5984. -> [... varname val this]
5985. processed one or more messages
5986. [... v1(filename) v2(line+flags)]
5987. must have been, since in array part
5988. slower but more compact variant
5989. TypedArray (or other non-ArrayBuffer duk_hbufferobject). * Conceptually same behavior as for an Array-like argument, * with a few fast paths.
5990. **comment:** parameter passing, not thread safe
- label:** requirement
5991. **comment:** Two temporaries are preallocated here for variants 3 and 4 which need * registers which are never clobbered by expressions in the loop * (concretely: for the enumerator object and the next enumerated value). * Variants 1 and 2 "release" these temps.
- label:** code-design
5992. Note: decimal number may start with a period, but must be followed by a digit
5993. main reachability roots
5994. DUK_HBUFFER_H_INCLUDED
5995. Inputs: explicit arguments (nargs), +1 for key, +2 for obj_index/nargs passing. * If the value stack does not contain enough args, an error is thrown; this matches * behavior of the other protected call API functions.
5996. [value offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
5997. placed in duk_heapdr_string
5998. * Main mark-and-sweep function. * * 'flags' represents the features requested by the caller. The current * heap->mark_and_sweep_base_flags is ORed automatically into the flags; * the base flags mask typically prevents certain mark-and-sweep operations * to avoid trouble.
5999. 'private'
6000. toJSON() can also be a lightfunc
6001. Special behavior for 'caller' property of (non-bound) function objects * and non-strict Arguments objects: if 'caller' -value- (!) is a strict * mode function, throw a TypeError (E5 Sections 15.3.5.4, 10.6). * Quite interestingly, a non-strict function with no formal arguments * will get an arguments object -without- special 'caller' behavior! * * The E5.1 spec is a bit ambiguous if this special behavior applies when * a bound function is the base value (not the 'caller' value): Section * 15.3.4.5 (describing bind()) states that [[Get]] for bound functions * matches that of Section 15.3.5.4 ([[Get]] for Function instances). * However, Section 13.3.5.4 has "NOTE: Function objects created using * Function.prototype.bind use the default [[Get]] internal method." * The current implementation assumes this means that bound functions * should not have the special [[Get]] behavior. * * The E5.1 spec is also a bit unclear if the TypeError throwing is * applied if the 'caller' value is a strict bound function. The * current implementation will throw even for both strict non-bound * and strict bound functions. * * See test-dev-strict-func-as-caller-prop-value.js for quite extensive * tests. * * This exotic behavior is disabled when the non-standard 'caller' property * is enabled, as it conflicts with the free use of 'caller'.
6002. move pivot out of the way
6003. final configuration
6004. -1 == not set, -2 == pending (next statement list)
6005. **comment:** temporary, must be signed and 32-bit to hold Unicode code points
- label:** code-design
6006. NB: 'length' property is automatically updated by the array setup loop
6007. index for next new key ([0,e_next[are gc reachable)
6008. idx is unsigned, < 0 check is not necessary
6009. indirect opcode follows direct
6010. * toFixed(), toExponential(), toPrecision()
6011. * Helpers for UTF-8 handling * * For bytecode readers the duk_uint32_t and duk_int32_t types are correct * because they're used for more than just codepoints.
6012. follow prototype chain
6013. [value]
6014. XXX: incref by count (here 2 times)
6015. equals
6016. true even with reattach
6017. Fatal error handling, called e.g. when a longjmp() is needed but * lj.jmpbuf_ptr is NULL. fatal_func must never return; it's not * declared as "noreturn" because doing that for typedefs is a bit * challenging portability-wise.
6018. **comment:** XXX: very messy now, but works; clean up, remove unused variables (nomimally * used so compiler doesn't complain).
- label:** code-design
6019. * Note: type is treated as a field separate from flags, so some masking is * involved in the macros below.
6020. Compute final offset in seconds, positive if local time ahead of * UTC (returned value is UTC-to-local offset). * * difftime() returns a double, so coercion to int generates quite * a lot of code. Direct subtraction is not portable, however. * XXX: allow direct subtraction on known platforms.
6021. **comment:** XXX: this is a very narrow check, and doesn't cover * zeroes, subnormals, infinities, which compare normally.
- label:** code-design
6022. Don't allow a zero divisor. Fast path check by * "verifying" with multiplication. Also avoid zero * dividend to avoid negative zero issues (0 / -1 = -0 * for instance).
6023. numeric value of a hex digit (also covers octal and decimal digits)
6024. DUK_TOK_MUL_EQ
6025. DUK_USE_DEBUG
6026. **comment:** XXX: optimize to use direct reads, i.e. avoid * value stack operations.
- label:** code-design
6027. 0x00: finish (non-white) * 0x01: continue
6028. %lu
6029. [... arg1 ... argN]
6030. executor interrupt running (used to avoid nested interrupts)
6031. [... func this]
6032. Constants: variable size encoding.
6033. * Variable declarations. * * Unlike function declarations, variable declaration values don't get * assigned on entry. If a binding of the same name already exists, just * ignore it silently.
6034. **comment:** XXX: duk_small_uint_t would be enough for this loop
- label:** code-design
6035. no res->strs[]
6036. * Debug dumping of duk_heap.
6037. Detach a debugger if attached (can be called multiple times) * safely.
6038. if 1, doing a string-to-number; else doing a number-to-string
6039. **comment:** Faster alternative: avoid making a temporary copy of tvptr_dst and use * fast incref/decref macros.
- label:** code-design
6040. * duk_hbuffer allocation and freeing.
6041. stash to bottom of value stack to keep new_env reachable for duration of eval
6042. step 4.a

6043. Handle a RETURN opcode. Avoid using longjmp() for return handling because * it has a measurable performance impact in ordinary environments and an extreme * impact in Emscripten (GH-342). Return value is on value stack top.

6044. exports

6045. Note: must behave like a no-op with NULL and any pointer * returned from malloc/realloc with zero size.

6046. **comment:** XXX: use duk_is_valid_index() instead?
label: code-design

6047. eat closing bracket

6048. * Object.isSealed() and Object.isFrozen() (E5 Sections 15.2.3.11, 15.2.3.13) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: all virtual (non-concrete) properties are currently non-configurable * and non-writable (and there are no accessor virtual properties), so they don't * need to be considered here now.

6049. * The switch statement is pretty messy to compile. * See the helper for details.

6050. re_ctx->captures at start and end of atom parsing. * Since 'captures' indicates highest capture number emitted * so far in a DUK_REOP_SAVE, the captures numbers saved by * the atom are: [start_captures,end_captures].

6051. Loop check using a hybrid approach: a fixed-size visited[] array * with overflow in a loop check object.

6052. first value: comma must not precede the value

6053. actually used, non-NULL keys

6054. r <- (2 * b) * f

6055. [...] re_obj input bc saved_buf res_obj]

6056. * Create wrapper object and serialize

6057. t1 <- t1 - s

6058. * Label handling * * Labels are initially added with flags prohibiting both break and continue. * When the statement type is finally uncovered (after potentially multiple * labels), all the labels are updated to allow/prohibit break and continue.

6059. 112.5%, i.e. new size less than 12.5% higher -> fast resize

6060. * URI handling

6061. * ASCII character constants * * C character literals like 'x' have a platform specific value and do * not match ASCII (UTF-8) values on e.g. EBCDIC platforms. So, use * these (admittedly awkward) constants instead. These constants must * also have signed values to avoid unexpected coercions in comparisons. * * <http://en.wikipedia.org/wiki/ASCII>

6062. 32-bit value

6063. string intern table (weak refs)

6064. **comment:** * Shared error message strings * * To minimize code footprint, try to share error messages inside Duktape * code. Modern compilers will do this automatically anyway, this is mostly * for older compilers.
label: code-design

6065. Convert indices to byte offsets.

6066. T

6067. Fast path: source is a TypedArray (or any bufferobject).

6068. Note: object cannot be a finalizable unreachable object, as * they have been marked temporarily reachable for this round, * and are handled above.

6069. 'var'

6070. [args(n) [crud] formals arguments map mappednames]

6071. **comment:** Note: 'this' is not necessarily an Array object. The push() * algorithm is supposed to work for other kinds of objects too, * so the algorithm has e.g. an explicit update for the 'length' * property which is normally "magical" in arrays.
label: code-design

6072. **comment:** * XXX: array indices are mostly typed as duk_uint32_t here; duk_uarridx_t * might be more appropriate.
label: code-design

6073. 'pointer'

6074. Return value would be pointless: because throw_flag==1, we always * throw if the identifier doesn't resolve.

6075. PC points to next instruction, find offending PC, * PC == 0 for native code.

6076. DUK_UNICODE_H_INCLUDED

6077. **comment:** dead code, but ensures portability (see Linux man page notes)
label: code-design

6078. DUK__ALLOW_AUTO_SEMI_ALWAYS workaround

6079. end of valstack allocation (exclusive)

6080. previous value is assumed to be garbage, so don't touch it

6081. * Scan number and setup for Dragon4. * * The fast path case is detected during setup: an integer which * can be converted without rounding, no net exponent. The fast * path could be implemented as a separate scan, but may not really * be worth it: the multiplications for building 'f' are not * expensive when 'f' is small. * * The significand ('f') must contain enough bits of (apparent) * accuracy, so that Dragon4 will generate enough binary output digits. * For decimal numbers, this means generating a 20-digit significand, * which should yield enough practical accuracy to parse IEEE doubles. * In fact, the EcmaScript specification explicitly allows an * implementation to treat digits beyond 20 as zeroes (and even * to round the 20th digit upwards). For non-decimal numbers, the * appropriate number of digits has been precomputed for comparable * accuracy. * * Digit counts: * * [dig_lzero] * | * .-...-[dig_prec]---. * | | | * 0000123.456789012345678901234567890e+123456 | | | | | * `--+-`----[dig_frac]-----`-+--`* | * [dig_whole] [dig_expt] * * dig_frac and dig_expt are -1 if not present * dig_lzero is only computed for whole number part * * Parsing state * * Parsing whole part dig_frac < 0 AND dig_expt < 0 * Parsing fraction part dig_frac >= 0 AND dig_expt < 0 * Parsing exponent part dig_expt >= 0 (dig_frac may be < 0 or >= 0) * * Note: in case we hit an implementation limit (like exponent range), * we should throw an error, NOT return NaN or Infinity. Even with * very large exponent (or significand) values the final result may be * finite, so NaN/Infinity would be incorrect.

6082. **comment:** suppress warning, not used
label: code-design

6083. Buffer/string -> compare contents.

6084. * Update cache entry (allocating if necessary), and move the * cache entry to the first place (in an "LRU" policy).

6085. finished jump

6086. string is ASCII, clen == blen

6087. DUK_USE_HOBJECT_HASH_PART || DUK_USE_STRTAB_PROBE

6088. * Since character data is only generated by decoding the source or by * the compiler itself, we rely on the input codepoints being correct * and avoid a check here. * * Character data can also come here through decoding of Unicode * escapes ("udead\ubeef") so all 16-bit unsigned values can be * present, even when the source file itself is strict UTF-8.

6089. **comment:** XXX: more emergency behavior, e.g. find smaller hash sizes etc
label: code-design

6090. * Copy array elements to new array part.

6091. For strings, special form for short lengths.

6092. Setup function properties.

6093. **comment:** * In a fast check we assume old_size equals old_used (i.e., existing * array is fully dense). * * Slow check if: * * (new_size - old_size) / old_size > limit * new_size - old_size > limit * old_size * new_size > (1 + limit) * old_size || limit' is 3 bits fixed point * new_size > (1 + (limit' / 8)) * old_size || * 8 * 8 * new_size > (8 + limit') * old_size || : 8 * new_size > (8 + limit') * (old_size / 8) * new_size > limit' * (old_size / 8) || limit' = 9 -> max 25% increase * arr_idx + 1 > limit' * (old_size / 8) * * This check doesn't work well for small values, so old_size is rounded * up for the check (and the '+ 1' of arr_idx can be ignored in practice): * * arr_idx > limit' * ((old_size + 7) / 8)
label: code-design

6094. **comment:** * A few notes on the algorithm: * * - Terms are not allowed to begin with a period unless the term * is either '.' or '..'. This simplifies implementation (and * is within CommonJS modules specification). * * - There are few output bound checks here. This is on purpose: * the resolution input is length checked and the output is never * longer than the input. The resolved output is written directly * over the input because it's never longer than the input at any * point in the

algorithm. ** - Non-ASCII characters are processed as individual bytes and * need no special treatment. However, U+0000 terminates the * algorithm; this is not an issue because U+0000 is not a * desirable term character anyway.

label: code-design

6095. * Utilities

6096. * Helpers ** The fast path checks are done within a macro to ensure "inlining" * while the slow path actions use a helper (which won't typically be * inlined in size optimized builds).

6097. **comment:** If the platform doesn't support the entire Ecmascript range, we need * to return 0 so that the caller can fall back to the default formatter. ** For now, assume that if time_t is 8 bytes or more, the whole Ecmascript * range is supported. For smaller time_t values (4 bytes in practice), * assumes that the signed 32-bit range is supported. ** XXX: detect this more correctly per platform. The size of time_t is * probably not an accurate guarantee of strftime() supporting or not * supporting a large time range (the full Ecmascript range).

label: code-design

6098. **comment:** XXX: now that pc2line is used by the debugger quite heavily in * checked execution, this should be optimized to avoid value stack * and perhaps also implement some form of pc2line caching (see * future work in debugger.rst).

label: code-design

6099. * Case clause. ** Note: cannot use reg_case as a temp register (for SEQ target) * because it may be a constant.

6100. **comment:** XXX: not sure what the correct semantic details are here, * e.g. handling of missing values (gaps), handling of non-array * trap results, etc. ** For keys, we simply skip non-string keys which seems to be * consistent with how e.g. Object.keys() will process proxy trap * results (ES6, Section 19.1.2.14).

label: code-design

6101. **comment:** XXX: 'copy properties' API call?

label: code-design

6102. * Determine whether to use the constructor return value as the created * object instance or not.

6103. Stack top contains plain value

6104. make current token the previous; need to fiddle with valstack "backing store"

6105. '{_func:true}'

6106. prediction: portable variant using doubles if 64-bit values not available

6107. [... arr val]

6108. Casting convenience.

6109. The stack has a variable shape here, so force it to the * desired one explicitly.

6110. **comment:** XXX: does not work if thr->catchstack is NULL

label: code-design

6111. Object.getOwnPropertyNames

6112. 'undefined'

6113. **comment:** XXX: with 'caller' property the callstack would need * to be unwound to update the 'caller' properties of * functions in the callstack.

label: code-design

6114. Overestimate required size; debug code so not critical to be tight.

6115. **comment:** no special formatting

label: code-design

6116. proplist is very rare

6117. * Handling DUK_OP_ADD this way is more compact (experimentally) * than a separate case with separate argument decoding.

6118. currently processing messages or breakpoints: don't enter message processing recursively (e.g. no breakpoints when processing debugger eval)

6119. DUK_USE_JC

6120. function refers to 'arguments' identifier

6121. don't want an intermediate exposed state with invalid pc

6122. Use loop to minimize code size of relookup after bound function case

6123. 'Undefined'

6124. **comment:** * Variable already declared, ignore re-declaration. * The only exception is the updated behavior of E5.1 for * global function declarations, E5.1 Section 10.5, step 5.e. * This behavior does not apply to global variable declarations.

label: code-design

6125. **comment:** Avoid fake finalization for the duk__refcount_fake_finalizer function * itself, otherwise we're in infinite recursion.

label: code-design

6126. **comment:** Borrow is detected based on wrapping which relies on exact 32-bit * types.

label: code-design

6127. if abandon_array, new_a_size must be 0

6128. Parse enumeration target and initialize enumerator. For 'null' and 'undefined', * INITENUM will creates a 'null' enumerator which works like an empty enumerator * (E5 Section 12.6.4, step 3). Note that INITENUM requires the value to be in a * register (constant not allowed).

6129. pointer is aligned, guaranteed for fixed buffer

6130. See MPUTOBJ comments above.

6131. Pause on the next opcode executed. This is always safe to do even * inside the debugger message loop: the interrupt counter will be reset * to its proper value when the message loop exits.

6132. Ensure there is internal valstack spare before we exit; this may * throw an alloc error. The same guaranteed size must be available * as before the call. This is not optimal now: we store the valstack * allocated size during entry; this value may be higher than the * minimal guarantee for an application.

6133. '\xffPc2line'

6134. **comment:** XXX: better coercion

label: code-design

6135. guaranteed to finish, as hash is never full

6136. shared flags for a subset of types

6137. nothing to incref

6138. The short string/integer initial bytes starting from 0x60 don't have * defines now.

6139. **comment:** XXX: use a helper for prototype traversal; no loop check here

label: code-design

6140. FunctionDeclaration: not strictly a statement but handled as such. ** O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnnum().

6141. unpack args

6142. hash lookup

6143. * Cleanup: restore original function, restore valstack state.

6144. don't touch property attributes or hash part

6145. Note: duk_uint8_t type yields larger code

6146. * The error object has been augmented with a traceback and other * info from its creation point -- usually another thread. The * error handler is called here right before throwing, but it also * runs in the resumer's thread. It might be nice to get a traceback * from the resumee but this is not the case now.

6147. Push function object, init flags etc. This must match * duk_js_push_closure() quite carefully.

6148. return arg as-is

6149. NULL accepted

6150. Lightfunc, not blamed now.

6151. 0 = no throw

6152. **comment:** XXX: macros

label: code-design

6153. Interrupt counter for triggering a slow path check for execution * timeout, debugger interaction such as breakpoints, etc. The value * is valid for the current running thread, and both the init and * counter values are copied whenever a thread switch occurs. It's * important for the counter to be conveniently accessible for

the * bytecode executor inner loop for performance reasons.

6154. start value for current countdown

6155. maximum recursion depth for loop detection stacks

6156. us

6157. [... holder name val]

6158. Allow automatic detection of hex base ("0x" or "0X" prefix), * overrides radix argument and forces integer mode.

6159. * Init arguments properties, map, etc.

6160. **comment:** NUL terminator handling doesn't matter here
label: code-design

6161. **comment:** XXX: error handling is incomplete. It would be cleanest if * there was a setjmp catchpoint, so that all init code could * freely throw errors. If that were the case, the return code * passing here could be removed.
label: code-design

6162. keep in on valstack, use borrowed ref below

6163. [... Logger clog logfunc clog(=this) msg]

6164. -> [... func this arg1 ... argN _Args]

6165. * Found * * Arguments object has exotic post-processing, see E5 Section 10.6, * description of [[GetOwnProperty]] variant for arguments.

6166. compiler's responsibility

6167. [... func this arg1 ... argN] (not tail call) * [this | arg1 ... argN] (tail call) * * idx_args updated to match

6168. func is NULL for lightfuncs

6169. shared exit path now

6170. Finish the wrapped module source. Force module.filename as the * function .filename so it gets set for functions defined within a * module. This also ensures loggers created within the module get * the module ID (or overridden filename) as their default logger name. * (Note capitalization: .filename matches Node.js while .fileName is * used elsewhere in Duktape.)

6171. **comment:** This limitation would be fixable but adds unnecessary complexity.
label: code-design

6172. -> [O toISOString O]

6173. * Finish

6174. -> [timeval this timeval]

6175. cached: valstack_end - valstack (in entries, not bytes)

6176. **comment:** XXX: awkward, especially the bunch of separate output values -> output struct?
label: code-design

6177. According to E5.1 Section 15.4.4.4 nonexistent trailing * elements do not affect 'length' of the result. Test262 * and other engines disagree, so update idx_last here too.

6178. * Bitstream encoder.

6179. * Once the whole refzero cascade has been freed, check for * a voluntary mark-and-sweep.

6180. Resolve start/end offset as element indices first; arguments * at idx_start/idx_end are element offsets. Working with element * indices first also avoids potential for wrapping.

6181. IdentifierStart production with ASCII and non-BMP excluded

6182. temp object for tracking / detecting duplicate keys

6183. negative

6184. * First create all built-in bare objects on the empty valstack. * * Built-ins in the index range [0,DUK_NUM_BUILTINS-1] have value * stack indices matching their eventual thr->builtins[] index. * * Built-ins in the index range [DUK_NUM_BUILTINS,DUK_NUM_ALL_BUILTINS] * will exist on the value stack during init but won't be placed * into thr->builtins[]. These are objects referenced in some way * from thr->builtins[] roots but which don't need to be indexed by * Duktape through thr->builtins[] (e.g. user custom objects).

6185. DUK_BUFOBJ_INT8ARRAY

6186. DUK_USE_ROM_STRINGS

6187. get pointer offsets for tweaking below

6188. Note: not normalized, but duk_push_number() will normalize

6189. NULL if not an object

6190. * Exposed number-to-string API * * Input: [number] * Output: [string]

6191. XXX: could add fast path for u8 compatible views

6192. Note2

6193. 'resume'

6194. should never happen for a strict callee

6195. **comment:** XXX: This is VERY inefficient now, and should be e.g. a * binary search or perfect hash, to be fixed.
label: code-design

6196. [... env]

6197. Lookup an identifier name in the current varmap, indicating whether the * identifier is register-bound and if not, allocating a constant for the * identifier name. Returns 1 if register-bound, 0 otherwise. Caller can * also check (out_reg_varbind >= 0) to check whether or not identifier is * register bound. The caller must NOT use out_rc_varname at all unless * return code is 0 or out_reg_varbind is < 0; this is because out_rc_varname * is unsigned and doesn't have a "unused" / none value.

6198. 'export'

6199. DUK_ERR_URI_ERROR: no macros needed

6200. array case is handled comprehensively above

6201. * Bytecode executor call. * * Execute bytecode, handling any recursive function calls and * thread resumptions. Returns when execution would return from * the entry level activation. When the executor returns, a * single return value is left on the stack top. * * The only possible longjmp() is an error (DUK_LJ_TYPE_THROW), * other types are handled internally by the executor.

6202. **comment:** XXX: refactor into an internal helper, pretty awkward
label: code-design

6203. Allocate a new valstack. * * Note: cannot use a plain DUK_REALLOC() because a mark-and-sweep may * invalidate the original thr->valstack base pointer inside the realloc * process. See doc/memory-management.rst.

6204. Note that this is the same operation for strict and loose equality: * - E5 Section 11.9.3, step 1.c (loose) * - E5 Section 11.9.6, step 4 (strict)

6205. force_no_namebind

6206. DUK_TOK_NEW

6207. * Breakpoint and step state checks

6208. **comment:** XXX: this behavior is quite useless now; it would be nice to be able * to create pointer values from e.g. numbers or strings. Numbers are * problematic on 64-bit platforms though. Hex encoded strings?
label: code-design

6209. global object doesn't have array part

6210. **comment:** Then evaluate RHS fully (its value becomes the expression value too). * Technically we'd need the side effect safety check here too, but because * we always throw using INVLHS the result doesn't matter.
label: code-design

6211. **comment:** XXX: specific getter
label: code-design

6212. [... escape_source bytecode]

6213. First we need to insert a jump in the middle of previously * emitted code to get the control flow right. No jumps can * cross the position where the jump is inserted. See doc/compiler.rst * for discussion on the intricacies of control flow and side effects * for variants 3 and 4.

6214. Cannot simulate individual finalizers because finalize_list only * contains objects with actual finalizers. But simulate side effects * from finalization by doing a bogus function call and resizing the * stacks.
6215. lastIndex is initialized to zero by new RegExp()
6216. **comment:** Note: we could return the hash index here too, but it's not * needed right now.
 label: code-design
6217. >>>
6218. [... closure template formals]
6219. inherit
6220. filename being compiled (ends up in functions' '_filename' property)
6221. filter out flags from exprop rbp_flags here to save space
6222. **comment:** * Debugging macros, DUK_DPRINT() and its variants in particular. * * DUK_DPRINT() allows formatted debug prints, and supports standard * and Duktape specific formatters. See duk_debug_vsnprintf.c for details. * * DUK_D(x), DUK_DD(x), and DUK_DDD(x) are used together with log macros * for technical reasons. They are concretely used to hide 'x' from the * compiler when the corresponding log level is disabled. This allows * clean builds on non-C99 compilers, at the cost of more verbose code. * Examples: * * DUK_D(DUK_DPRINT("foo")); * DUK_DD(DUK_DPRINT("foo")); * DUK_DDD(DUK_DDDPRINT("foo")); * * This approach is preferable to the old "double parentheses" hack because * double parentheses make the C99 solution worse: __FILE__ and __LINE__ can * no longer be added transparently without going through globals, which * works poorly with threading.
 label: code-design
6223. prop index in 'array part', < 0 if not there
6224. caller handles sign change
6225. **comment:** load factor too low or high, count actually used entries and resize
 label: code-design
6226. * Coercion and fast path processing.
6227. Not strictly necessary because if key == NULL, flag MUST be ignored.
6228. make the new thread reachable
6229. XXX: print built-ins array?
6230. temp reg
6231. * Externs and prototypes
6232. no incref
6233. Test signaling NaN and alias assignment in all endianness combinations.
6234. 1 GB
6235. XXX: fast path for array arguments?
6236. -> [... new_global new_globalenv new_global new_global]
6237. E5.1 Section 15.1.3.3: uriReserved + uriUnescaped + '#'
6238. DUK_USE_PACKED_TVAL
6239. **comment:** * Enumeration semantics come from for-in statement, E5 Section 12.6.4. * If called with 'null' or 'undefined', this opcode returns 'null' as * the enumerator, which is special cased in NEXTENUM. This simplifies * the compiler part
 label: code-design
6240. Insert a reserved area somewhere in the buffer; caller fills it. * Evaluates to a (duk_uint_t *) pointing to the start of the reserved * area for convenience.
6241. values for the state field
6242. How large a loop detection stack to use
6243. pop enum
6244. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)
6245. <
6246. [... closure template undefined]
6247. **comment:** * To determine whether to use an optimized Ecmascript-to-Ecmascript * call, we need to know whether the final, non-bound function is an * Ecmascript function. * * This is now implemented so that we start to do an ecma-to-ecma call * setup which will resolve the bound chain as the first thing. If the * final function is not eligible, the return value indicates that the * ecma-to-ecma call is not possible. The setup will overwrite the call * target at DUK_REGP(idx) with the final, non-bound function (which * may be a lightfunc), and fudge arguments if necessary. * * XXX: If an ecma-to-ecma call is not possible, this initial call * setup will do bound function chain resolution but won't do the * "effective this binding" resolution which is quite confusing. * Perhaps add a helper for doing bound function and effective this * binding resolution - and call that explicitly? Ecma-to-ecma call * setup and normal function handling can then assume this prestep has * been done by the caller.
 label: code-design
6248. dynamic buffer ops
6249. not found in 'curr', next in prototype chain; impose max depth
6250. 13: getUTCDate
6251. **comment:** Truncate to 16 bits here, so that a computed hash can be compared * against a hash stored in a 16-bit field.
 label: code-design
6252. Buffer has virtual properties similar to string, but indexed values * are numbers, not 1-byte buffers/strings which would perform badly.
6253. [... this name]
6254. * Date/time parsing helper. * * Parse a datetime string into a time value. We must first try to parse * the input according to the standard format in E5.1 Section 15.9.1.15. * If that fails, we can try to parse using custom parsing, which can * either be platform neutral (custom code) or platform specific (using * existing platform API calls). * * Note in particular that we must parse whatever toString(), toUTCString(), * and toISOString() can produce; see E5.1 Section 15.9.4.2. * * Returns 1 to allow tail calling. * * There is much room for improvement here with respect to supporting * alternative datetime formats. For instance, V8 parses '2012-01-01' as * UTC and '2012/01/01' as local time.
6255. **comment:** XXX: check flags
 label: requirement
6256. ASCII (and EOF) fast path -- quick accept and reject
6257. 0xc0...0xfc
6258. '-' as a range indicator
6259. -> [... enum_target res handler undefined target]
6260. [arg1 ... argN this loggerLevel loggerName 'fmt' arg]
6261. "123.456"
6262. Helper for creating the arguments object and adding it to the env record * on top of the value stack. This helper has a very strict dependency on * the shape of the input stack.
6263. If an implicit return value is needed by caller, it must be * initialized to 'undefined' because we don't know whether any * non-empty (where "empty" is a continuation type, and different * from an empty statement) statements will be executed. * * However, since 1st pass is a throwaway one, no need to emit * it here.
6264. 'false'
6265. Some internal code now assumes that all duk_uint_t values * can be expressed with a duk_size_t.
6266. data property or accessor without getter
6267. * Args validation
6268. DUK_TAG_NUMBER is intentionally first, as it is the default clause in code * to support the 8-byte representation. Further, it is a non-heap-allocated * type so it should come before DUK_TAG_STRING. Finally, it should not break * the tag value ranges covered by case-clauses in a switch-case.
6269. * String built-ins
6270. * Clear (reachable) flags of refzero work list.
6271. **comment:** * Number should already be in NaN-normalized form, but let's * normalize anyway.
 label: code-design
6272. catchstack

6273. * C call recursion depth check, which provides a reasonable upper * bound on maximum C stack size (arbitrary C stack growth is only * possible by recursive handle_call / handle_safe_call calls).

6274. Call sites don't need the result length so it's not accumulated.

6275. non-zero: make copy

6276. Return non-zero (true) if we have a good reason to believe * the notify was delivered; if we're still attached at least * a transport error was not indicated by the transport write * callback. This is not a 100% guarantee of course.

6277. fromPresent = false

6278. Check that there's room to push one value.

6279. comma check

6280. backtrack to last output slash (dups already eliminated)

6281. **comment:** Fastint range is signed 48-bit so longest value is $-2^{47} = -140737488355328$ * (16 chars long), longest signed 64-bit value is $-2^{63} = -922372036854775808$ * (20 chars long). Alloc space for 64-bit range to be safe.
label: code-design

6282. Flags for call handling.

6283. jump to end

6284. * RegExp built-ins

6285. allocation size

6286. no -> decref members, then free

6287. If any non-Array value had enumerable virtual own * properties, they should be serialized here. Standard * types don't.

6288. XXX: C recursion limit if proxies are allowed as handler/target values

6289. try part

6290. 'caller' must only take on 'null' or function value

6291. sign doesn't matter when writing

6292. **comment:** XXX: use macros for the repetitive tval/refcount handling.
label: code-design

6293. **comment:** Builtin-objects; may or may not be shared with other threads, * threads existing in different "compartments" will have different * built-ins. Must be stored on a per-thread basis because there * is no intermediate structure for a thread group / compartment. * This takes quite a lot of space, currently $43 \times 4 = 172$ bytes on * 32-bit platforms. * * In some cases the builtins array could be ROM based, but it's * sometimes edited (e.g. for sandboxing) so it's better to keep * this array in RAM.
label: code-design

6294. **comment:** unused
label: code-design

6295. * Other file level defines

6296. Fast path for numbers (one of which may be a fastint)

6297. maximum length for a SKIP-1 diffstream: 35 bits per entry, rounded up to bytes

6298. [enum_target enum res]

6299. orig value

6300. **comment:** XXX use get tval ptr, more efficient
label: code-design

6301. See comments in duk_pcall().

6302. The 'left' value must not be a register bound variable * because it may be mutated during the rest of the expression * and E5.1 Section 11.2.1 specifies the order of evaluation * so that the base value is evaluated first. * See: test-bug-nested-prop-mutate.js.

6303. A bit tricky overflow test, see doc/code-issues.rst.

6304. key must not already exist in entry part

6305. Digit generation

6306. reset function state (prepare for pass 2)

6307. +0 = catch

6308. avoid side effects!

6309. **comment:** NOTE: "get" and "set" are not officially ReservedWords and the lexer * currently treats them always like ordinary identifiers (DUK_TOK_GET * and DUK_TOK_SET are unused). They need to be detected based on the * identifier string content.
label: code-design

6310. **comment:** XXX: regetting the pointer may be overkill - we're writing * to a side-effect free array here.
label: code-design

6311. * Because buffer values may be looped over and read/written * from, an array index fast path is important.

6312. duk_small_uint_fast_t c = DUK_DEC_C(ins);

6313. For all duk_hbufferobjects, get the plain buffer inside * without making a copy. This is compatible with Duktape 1.2 * but means that a slice/view information is ignored and the * full underlying buffer is returned. * * If called as a constructor, a new Duktape.Buffer object * pointing to the same plain buffer is created below.

6314. Copy values by index reads and writes. Let virtual * property handling take care of coercion.

6315. **comment:** No coercions or other side effects, so safe
label: requirement

6316. 0x20-0x2f

6317. may become sparse...

6318. Pointers may be NULL for a while when 'buf' size is zero and before any * ENSURE calls have been made. Once an ENSURE has been made, the pointers * are required to be non-NULL so that it's always valid to use memcpy() and * memmove(), even for zero size.

6319. * Create and throw an Ecmascript error object based on a code and a message. * * Used when we throw errors internally. Ecmascript generated error objects * are created by Ecmascript code, and the throwing is handled by the bytecode * executor.

6320. XXXXXX--

6321. lightfuncs are treated like objects and not coerced

6322. jump to end or else part

6323. **comment:** Registers 'bc' and 'bc + 1' are written in longjmp handling * and if their previous values (which are temporaries) become * unreachable -and- have a finalizer, there'll be a function * call during error handling which is not supported now (GH-287). * Ensure that both 'bc' and 'bc + 1' have primitive values to * guarantee no finalizer calls in error handling. Scrubbing also * ensures finalizers for the previous values run here rather than * later. Error handling related values are also written to 'bc' * and 'bc + 1' but those values never become unreachable during * error handling, so there's no side effect problem even if the * error value has a finalizer.
label: code-design

6324. no_block

6325. The original value needs to be preserved for filter(), hence * this funny order. We can't re-get the value because of side * effects.

6326. for duk_error_augment.c

6327. just one 'int' for C++ exceptions

6328. '~'

6329. reset voluntary gc trigger count

6330. If the separator is a RegExp, make a "clone" of it. The specification * algorithm calls [[Match]] directly for specific indices; we emulate this * by tweaking lastIndex and using a "force global" variant of duk_regexp_match() * which will use global-style matching even when the RegExp itself is non-global.

6331. * Defines for JSON, especially duk_bi_json.c.

6332. * Fast path tables

6333. -> [... handler trap]

6334. DUK_TOK_ALSHIFT_EQ

6335. Note: assumes 'data' is always a fixed buffer

6336. * Stringtable entry for fixed size stringtable
6337. sometimes stack and array indices need to go on the stack
6338. hash size ratio goal, must match genhashsizes.py
6339. * Heap structure. ** Heap contains allocated heap objects, interned strings, and built-in * strings for one or more threads.
6340. Require a lot of stack to force a value stack grow/shrink.
6341. **comment:** XXX: lastIndex handling produces a lot of asm
 label: code-design
6342. DUK_DEBUG_H_INCLUDED
6343. input size is good output starting point
6344. XXX: Current putvar implementation doesn't have a success flag, * add one and send to debug client?
6345. **comment:** must be able to emit code, alloc consts, etc.
 label: code-design
6346. **comment:** Should never happen but avoid infinite loop just in case.
 label: code-design
6347. unchanged from Duktape.Thread.yield()
6348. value1 -> return value, pseudo-type to indicate a return continuation (for ENDFIN)
6349. Negative offsets cause a RangeError.
6350. side effect free
6351. main expression parser function
6352. may happen if size is very close to 2^32-1
6353. fail: restore saves
6354. DUK_USE_DATE_PRS_GETDATE
6355. needed for stepping
6356. **comment:** XXX: could map/decode be unified with duk_unicode_support.c code? * Case conversion needs also the character surroundings though.
 label: code-design
6357. clen == blen -> pure ascii
6358. * The attempt to allocate may cause a GC. Such a GC must not attempt to resize * the stringtable (though it can be swept); finalizer execution and object * compaction must also be postponed to avoid the pressure to add strings to the * string table. Call site must prevent these.
6359. current setjmp() catchpoint
6360. Result is undefined.
6361. [...] proplist
6362. This may throw an error though not for valid E5 strings.
6363. * Marking functions for heap types: mark children recursively
6364. * Any catch bindings ("catch (e)") also affect identifier binding. ** Currently, the varmap is modified for the duration of the catch * clause to ensure any identifier accesses with the catch variable * name will use slow path.
6365. XXX: set with duk_hobject_set_length() when tracedata is filled directly
6366. **comment:** XXX: optimize by creating array into correct size directly, and * operating on the array part directly; values can be memcpy()'d from * value stack directly as long as recounts are increased.
 label: code-design
6367. Slow path.
6368. 'compile'
6369. **comment:** XXX: does not work if thr->catchstack is allocated but lowest pointer
 label: code-design
6370. DUK_TOK_RCURLY
6371. **comment:** XXX: check .caller writability?
 label: code-design
6372. replacement handler
6373. 2**31-1 ~ 2G properties
6374. part of string
6375. Bytecode instructions: endian conversion needed unless * platform is big endian.
6376. DUK_TOK_EXTENDS
6377. XXX: optimize reconfig valstack operations so that resize, clamp, and setting * top are combined into one pass.
6378. fastint downgrade check for return values
6379. **comment:** bytes of indent still needed
 label: code-design
6380. DUK_JS_COMPILER_H_INCLUDED
6381. nop
6382. Restore the previous setjmp catcher so that any error in * error handling will propagate outwards rather than re-enter * the same handler. However, the error handling path must be * designed to be error free so that sandboxing guarantees are * reliable, see e.g. <https://github.com/svaarala/duktape/issues/476>.
6383. Double error
6384. * Valstack spare check
6385. This would be pointless: unexpected type and lightfunc would both return NULL
6386. For internal use: get non-accessor entry value
6387. trailing comma followed by rcurly
6388. Prevent mark-and-sweep for the pending finalizers, also prevents * refzero handling from moving objects away from the heap_allocated * list. (The flag meaning is slightly abused here.)
6389. [...] value this_binding]
6390. DUK_TOK_INCREMENT
6391. * Shared handling of binary operations ** args = (is_extraop << 16) + (opcode << 8) + rbp
6392. insert code for matching the remainder - infinite or finite
6393. **comment:** XXX: shared api error strings, and perhaps even throw code for rare cases?
 label: code-design
6394. Although there are writable virtual properties (e.g. plain buffer * and buffer object number indices), they are handled before we come * here.
6395. * Fast paths
6396. **comment:** XXX: static alloc is OK until separate chaining stringtable * resizing is implemented.
 label: code-design
6397. truncate towards zero
6398. 37: setUTCFullYear
6399. **comment:** Function name: missing/undefined is mapped to empty string, * otherwise coerce to string.
 label: code-design
6400. * Assert helpers
6401. resume lookup from target
6402. **comment:** XXX: assume these?
 label: code-design
6403. When creating built-ins, some of the built-ins may not be set * and we want to tolerate that when throwing errors.
6404. standard prototype
6405. **comment:** XXX: consolidated algorithm step 15.f -> redundant?
 label: code-design

6471. Careful here: state must be wiped before the call * so that we can cleanly handle a re-attach from * inside the callback.
6472. assume 0 <=> NULL
6473. skip leading digit
6474. B -> object register * C -> C+0 contains key, C+1 closure (value)
6475. caller guarantees
6476. **comment:** array part must not contain any non-unused properties, as they would * be configurable and writable.
 label: code-design
6477. Note: '\b' in char class is different than outside (assertion), * '\B' is not allowed and is caught by the duk_unicode_is_identifier_part() * check below.
6478. Note: combining __FILE__, __LINE__, and __func__ into fmt would be * possible compile time, but waste some space with shared function names.
6479. at least this function exists
6480. hash part is a 'weak reference' and does not contribute
6481. 'pc'
6482. 'errThrow'
6483. DUK_TOK_SUPER
6484. This is guaranteed by debugger code.
6485. [... (sep) str1 str2 ... strN buf]
6486. 0: no change
6487. Bigint is 2^52. Used to detect normalized IEEE double mantissa values * which are at the lowest edge (next floating point value downwards has * a different exponent). The lowest mantissa has the form: * * 1000.....000 (52 zeroes; only "hidden bit" is set)
6488. reg(ignore)
6489. undefined coerces to zero which is correct
6490. overwrite any previous binding of the same name; the effect is * that last argument of a certain name wins.
6491. returned after EOF (infinite amount)
6492. **comment:** XXX: investigate whether write protect can be handled above, if we * just update length here while ignoring its protected status
 label: code-design
6493. **comment:** This is a bit clunky because it is ANSI C portable. Should perhaps * relocate to another file because this is potentially platform * dependent.
 label: code-design
6494. Unwind catchstack entries referring to the callstack entry we're reusing
6495. **comment:** XXX: many operations actually want toforcedtemp() -- brand new temp?
 label: code-design
6496. e.g. DUK_OP_POSTINCR
6497. 'Uint8ClampedArray'
6498. No copy, leave zero bytes in the buffer. There's no * ambiguity with Float32/Float64 because zero bytes also * represent 0.0.
6499. no typedef
6500. [... func_template]
6501. Note: any entries above the callstack top are garbage and not zeroed. * Also topmost activation idx_retval is garbage (not zeroed), and must * be ignored.
6502. [arg1 ... argN obj length]
6503. Previous value of 'func' caller, restored when unwound. Only in use * when 'func' is non-strict.
6504. When doing string-to-number, lowest_mantissa is always 0 so * the exponent check, while incorrect, won't matter.
6505. If the value has a prototype loop, it's critical not to * throw here. Instead, assume the value is not to be * augmented.
6506. **comment:** unused with some debug level combinations
 label: code-design
6507. Line number range for function. Needed during debugging to * determine active breakpoints.
6508. **comment:** slow init, do only for slow path cases
 label: code-design
6509. DUK_BUFOBJ_DUKTAPE_BUFFER
6510. positive
6511. **comment:** XXX: Here we have a nasty dependency: the need to manipulate * the callstack means that catchstack must always be unwound by * the caller before unwinding the callstack. This should be fixed * later.
 label: code-design
6512. e.g. DUK_OP_PREINCP
6513. DUK_JS_H_INCLUDED
6514. DUK_USE_DATE_NOW_WINDOWS
6515. !DUK_SINGLE_FILE
6516. this is added to checks to allow for Duktape * API calls in addition to function's own use
6517. [... key_obj key]
6518. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.
6519. is_setget
6520. flags always encodes to 1 byte
6521. Ensure space for 1:1 output plus one escape.
6522. [... obj key value]
6523. * Byte-based matching would be possible for case-sensitive * matching but not for case-insensitive matching. So, we * match by decoding the input and bytecode character normally. * * Bytecode characters are assumed to be already canonicalized. * Input characters are canonicalized automatically by * duk_inp_get_cp0() if necessary. * * There is no opcode for matching multiple characters. The * regexp compiler has trouble joining strings efficiently * during compilation. See doc/regexec.rst for more discussion.
6524. C array of duk_labelinfo
6525. **comment:** XXX: double evaluation for 'tv' argument.
 label: code-design
6526. with stack depth (affects identifier lookups)
6527. [... Duktape.modSearch resolved_id last_comp fresh_require exports module]
6528. * Detect recursive invocation
6529. **comment:** known to be 32-bit
 label: code-design
6530. * Directive prologue tracking.
6531. Handle lightfuncs through object coercion for now.
6532. exponent 0
6533. Identifier found in registers (always non-deletable) * or declarative environment record and non-configurable.
6534. duk_handle_ecma_call_setup: setup for a tail call
6535. match: keep wiped/resaved values
6536. E5 Section 11.7.2, steps 7 and 8
6537. **comment:** * Since there are no references in the catcher structure, * unwinding is quite simple. The only thing we need to * look out for is popping a possible lexical environment * established for an active catch clause.
 label: code-design
6538. * String cache. * * Provides a cache to optimize indexed string lookups. The cache keeps * track of (byte offset, char offset) states for a fixed number of strings. * Otherwise we'd need to scan from either end of the string, as we store * strings in (extended) UTF-8.
6539. heap thread, used internally and for finalization

6540. Don't allow a constant for the object (even for a number etc), as * it goes into the 'A' field of the opcode.
6541. For now all calls except Ecma-to-Ecma calls prevent a yield.
6542. If tv1==tv2 this is a NOP, no check is needed
6543. Dispatch loop.
6544. DUK_TOK_SUB
6545. Shared Windows helpers.
6546. This is a pretty awkward control flow, but we need to recheck the * key coercion here.
6547. catches EOF and invalid digits
6548. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur. If we end up not making the * call we must restore the value.
6549. **comment:** XXX: better macro for DUK_TVAL_IS_UNDEFINED_OR_NULL(tv)
label: code-design
6550. * Call the constructor function (called in "constructor mode").
6551. minimum prime
6552. set by atom case clauses
6553. [... error func fileName]
6554. reset callstack limit
6555. right paren eaten
6556. '['
6557. idx_desc
6558. E5 Section 11.7.1, steps 7 and 8
6559. coerced value is updated to value stack even when RangeError thrown
6560. Note: special DUK__EMIT_FLAG_B_IS_TARGETSOURCE * used to indicate that B is both a source and a * target register. When shuffled, it needs to be * both input and output shuffled.
6561. * Number is special because it doesn't have a specific * tag in the 8-byte representation.
6562. TypedArray views need an automatic ArrayBuffer which must be * provided as .buffer property of the view. Just create a new * ArrayBuffer sharing the same underlying buffer.
6563. Align 'p' to 4; the input data may have arbitrary alignment. * End of string check not needed because blen >= 16.
6564. Expression_opt
6565. Exact halfway, round to even.
6566. * Arguments check
6567. **comment:** Very minimal inlining to handle common idioms '!0' and '!1', * and also boolean arguments like '!false' and '!true'.
label: code-design
6568. name is empty -> return message
6569. * Rescue or free.
6570. grow by at most one
6571. [A B C D E F G H] rel_index = 2, del_count 3, item count 3 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)
6572. **comment:** XXX: helper, rely on Boolean.prototype as being non-writable, non-configurable
label: code-design
6573. Note: proxy handling must happen before key is string coerced.
6574. DUK_HBUFFEROBJECT_H_INCLUDED
6575. [... res]
6576. Note: unconditional throw
6577. DUK_ERRMSG_H_INCLUDED
6578. Neutered buffer, zero length seems * like good behavior here.
6579. no mark-and-sweep gc
6580. * DUK_BSWAP macros
6581. literals (E5 Section 7.8), except null, true, false, which are treated * like reserved words (above).
6582. loggerName
6583. will result in undefined
6584. DUK_TOK_QUESTION
6585. * Delayed initialization only occurs for 'NEWENV' functions.
6586. Allow automatic detection of octal base, overrides radix * argument and forces integer mode.
6587. Load constants onto value stack but don't yet copy to buffer.
6588. [hobject props enum(props) key desc]
6589. 'Float32Array'
6590. eat comma
6591. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT32
6592. Since stack indices are not reliable, we can't do anything useful * here. Invoke the existing setjmp catcher, or if it doesn't exist, * call the fatal error handler.
6593. [A B C D E F G H] rel_index = 2, del_count 3, item count 4 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F F G H] (actual result at this point)
6594. Commands and notifies initiated by Duktape.
6595. DUK_TOK_GT
6596. * Stringtable
6597. offset is now target of the pending split (right after jump)
6598. U+03A3 = GREEK CAPITAL LETTER SIGMA
6599. always 0 args
6600. exit bytecode executor by rethrowing an error to caller
6601. Note: e_next indicates the number of gc-reachable entries * in the entry part, and also indicates the index where the * next new property would be inserted. It does *not* indicate * the number of non-NULL keys present in the object. That * value could be counted separately but requires a pass through * the key list.
6602. initialize built-ins - either by copying or creating new ones
6603. Note: thr->catchstack_top may be 0, so that cat < thr->catchstack * initially. This is OK and intended.
6604. SCANBUILD: scan-build complains here about assigned value * being garbage or undefined. This is correct but operating * on undefined values has no ill effect and is ignored by the * caller in the case where this happens.
6605. IsAccessorDescriptor(desc) == true
6606. avoid issues with relocation
6607. avoid recursive string table call
6608. res.x1 -> res.x2
6609. -> [... x y fn]
6610. 'taint' result as complex -- this is conservative, * as lookaheads do not backtrack.
6611. LHS is already side effect free
6612. **comment:** XXX: hack, remove when const lookup is not O(n)
label: code-design
6613. Here the specification requires correct array index enumeration * which is a bit tricky for sparse arrays (it is handled by the * enum setup code). We now enumerate ancestors too, although the * specification is not very clear on whether that is required.
6614. object has an array part (a_size may still be 0)
6615. 0x20...0x2f

6616. eat dup slashes
6617. Get the current data pointer (caller must ensure buf != NULL) as a * duk_uint8_t ptr.
6618. initial estimate based on format string
6619. enable exotic behaviors last
6620. **comment:** if 'src < src_end_safe', safe to read 4 bytes
 label: requirement
6621. **comment:** Straightforward algorithm, makes fewer compiler assumptions.
 label: code-design
6622. [obj key desc]
6623. **comment:** Convenience for some situations; the above macros don't allow NULLs * for performance reasons.
 label: code-design
6624. [... env callee varmap key val]
6625. currently required
6626. * Yield the current thread. * * The thread must be in yieldable state: it must have a resumer, and there * must not be any yield-preventing calls (native calls and constructor calls, * currently) in the thread's call stack (otherwise a resume would not be * possible later). This method must be called from an Ecmascript function. * * Args: * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.
6627. **comment:** No need to length check string: it will never exceed even * the 16-bit length maximum.
 label: code-design
6628. (flags<<32) + (line), flags = 0
6629. insert range count
6630. XXX: Set 32-bit result (but must then handle signed and * unsigned results separately).
6631. [... new_glob new_env]
6632. Count actually used (non-NULL, non-DELETED) entries.
6633. no refcount check
6634. Parse statements until a closing token (EOF or '}') is found.
6635. duk_hthread_callstack_unwind() will decrease this on unwind
6636. Note: array dump will include elements beyond * 'length'.
6637. Note: currently the catch binding is handled without a register * binding because we don't support dynamic register bindings (they * must be fixed for an entire function). So, there is no need to * record regbases etc.
6638. Shared helper to provide toString() and valueOf(). Checks 'this', gets * the primitive value to stack top, and optionally coerces with ToString().
6639. size stored in duk_heapheader unused flag bits
6640. genbuiltins.py ensures
6641. -> [... val]
6642. CR LF again a special case
6643. Note: computes with 'idx' in assertions, so caller beware. * 'idx' is preincremented, i.e. '1' on first call, because it * is more convenient for the caller.
6644. Full, aligned 4-byte reads.
6645. stack[0] = start * stack[1] = end * stack[2] = ToObject(this) * stack[3] = ToUint32(length) * stack[4] = result array
6646. * Zero the struct, and start initializing roughly in order
6647. stack[0...nargs-1] = unshift args (vararg) * stack[nargs] = ToObject(this) * stack[nargs+1] = ToUint32(length)
6648. duk_unicode_idp_m_ids_noa[]
6649. mark-and-sweep is currently running
6650. **comment:** * Error codes: defined in duktape.h * * Error codes are used as a shorthand to throw exceptions from inside * the implementation. The appropriate Ecmascript object is constructed * based on the code. Ecmascript code throws objects directly. The error * codes are defined in the public API header because they are also used * by calling code.
 label: code-design
6651. DUK_API_INTERNAL_H_INCLUDED
6652. '\xffTarget'
6653. FUNCTION EXPRESSIONS
6654. 0x50-0x5f
6655. Note: there is no standard formatter for function pointers
6656. default: no change
6657. internal type tag
6658. * Ecmascript bytecode executor.
6659. **comment:** 0 = DUK_HOBJECT_CLASS_UNUSED
 label: code-design
6660. p points to digit part ('%xy', p points to 'x')
6661. the compressed pointer is zeroed which maps to NULL, so nothing to do.
6662. 'count' is more or less comparable to normal trigger counter update * which happens in memory block (re)allocation.
6663. bottom of current frame
6664. Shared offset/length coercion helper.
6665. E5 Section 15.10.2.11
6666. 'writable'
6667. These functions have trouble working as lightfuncs. * Some of them have specific asserts and some may have * additional properties (e.g. 'require.id' may be written).
6668. Two lowest bits of opcode are used to distinguish * variants. Bit 0 = inc(0)/dec(1), bit 1 = pre(0)/post(1).
6669. internal spare
6670. value (error) is at stack top
6671. input byte length
6672. gap (if empty string, NULL)
6673. [args [crud] arguments]
6674. lightfunc
6675. higher value conserves memory; also note that linear scan is cache friendly
6676. **comment:** XXX: to be implemented?
 label: code-design
6677. **comment:** XXX: convert to fixed buffer?
 label: code-design
6678. **comment:** Debug macro to print all parts and dparts (used manually because of debug level).
 label: code-design
6679. Assume arrays are dense in the fast path.
6680. cannot happen: strings are not put into refzero list (they don't even have the next/prev pointers)
6681. * Value stack top handling
6682. Set property slot to an empty state. Careful not to invoke * any side effects while using desc.e_idx so that it doesn't * get invalidated by a finalizer mutating our object.
6683. x <= y --> not (x > y) --> not (y < x)
6684. [key setter this val]
6685. little endian
6686. Typing: use duk_small_(u)int_fast_t when decoding small * opcode fields (op, A, B, C) and duk_(u)int_fast_t when * decoding larger fields (e.g. BC which is 18 bits). Use * unsigned variant by default, signed when the value is used * in signed arithmetic. Using variable names such as 'a', 'b', * 'c', 'bc', etc makes it easier to

spot typing mismatches.

6687. * Create required objects: * - 'arguments' object: array-like, but not an array * - 'map' object: internal object, tied to 'arguments' * - 'mappedNames' object: temporary value used during construction

6688. * Misc util stuff

6689. Contract, either: * - Push string to value stack and return 1 * - Don't push anything and return 0

6690. Node.js return value for failed writes is offset + #bytes * that would have been written.

6691. NaN sign bit is platform specific with unpacked, un-normalized NaNs

6692. **comment:** XXX: union would be more correct

label: code-design

6693. * Allocate an duk_hobject. * * The allocated object has no allocation for properties; the caller may * want to force a resize if a desired size is known. * * The allocated object has zero reference count and is not reachable. * The caller MUST make the object reachable and increase its reference * count before invoking any operation that might require memory allocation.

6694. **comment:** XXX: Stack discipline is annoying, could be changed in numconv.

label: code-design

6695. 'stack'

6696. * Heap compiled function (Ecmascript function) representation. * * There is a single data buffer containing the Ecmascript function's * bytecode, constants, and inner functions.

6697. Behavior for nargs < 2 is implementation dependent: currently we'll * set a NaN time value (matching V8 behavior) in this case.

6698. ".123"

6699. currently used -> new size

6700. caller is responsible for ensuring this

6701. must be signed

6702. uppercase

6703. **comment:** This variant is needed by String.prototype.split(); it needs to perform * global-style matching on a cloned RegExp which is potentially non-global.

label: code-design

6704. **comment:** XXX: optimize this filling behavior later

label: code-design

6705. replaced

6706. **comment:** * Binary bitwise operations use different coercions (ToInt32, ToUint32) * depending on the operation. We coerce the arguments first using * ToInt32(), and then cast to an 32-bit value if necessary. Note that * such casts must be correct even if there is no native 32-bit type * (e.g., duk_int32_t and duk_uint32_t are 64-bit). * * E5 Sections 11.10, 11.7.1, 11.7.2, 11.7.3

label: code-design

6707. [... regexp input] -> [res_obj]

6708. Mostly API and built-in method related

6709. E5 Sections 15.9.3.1, B.2.4, B.2.5

6710. alloc and init

6711. prediction corrections for prime list (see genhashsizes.py)

6712. Make _Target and _Handler non-configurable and non-writable. * They can still be forcibly changed by C code (both user and * Duktape internal), but not by Ecmascript code.

6713. DUK_USE_REGEX_EXP_SUPPORT

6714. the DUK_TOK_RCURLY is eaten by duk_parse_stmts()

6715. * Interrupt counter fixup (for development only).

6716. Note: recursive call

6717. maximum bytecode length in instructions

6718. get const for value at valstack top

6719. value1 -> label number, pseudo-type to indicate a break continuation (for ENDFIN)

6720. line tracking

6721. first call

6722. * All done

6723. undefined is accepted

6724. * Wrap up

6725. this case can no longer occur because refcount is unsigned

6726. automatic semi will be inserted

6727. Special shuffling for INITGET/INITSET, where slot C * identifies a register pair and cannot be shuffled * normally. Use an indirect variant instead.

6728. reachable through activation

6729. [... filename &comp_stk]

6730. **comment:** XXX: faster initialization (direct access or better primitives)

label: code-design

6731. Loop structure ensures that we don't compute t1^2 unnecessarily * on the final round, as that might create a bignum exceeding the * current DUK_BL_MAX_PARTS limit.

6732. **comment:** * Check for invalid backreferences; note that it is NOT an error * to back-reference a capture group which has not yet been introduced * in the pattern (as in ^1(foo)/); in fact, the backreference will * always match! It IS an error to back-reference a capture group * which will never be introduced in the pattern. Thus, we can check * for such references only after parsing is complete.

label: code-design

6733. + 1 for rounding

6734. note: any entries above the callstack top are garbage and not zeroed

6735. must be, since env was created when catcher was created

6736. side effects (currently none though)

6737. DUK_BUILTIN_PROTOS_H_INCLUDED

6738. * Find a matching label catcher or 'finally' catcher in * the same function. * * A label catcher must always exist and will match unless * a 'finally' captures the break/continue first. It is the * compiler's responsibility to ensure that labels are used * correctly.

6739. shadowed, ignore

6740. 'Uint16Array'

6741. -> [... enum_target res]

6742. * Round-up limit. * * For even values, divides evenly, e.g. 10 -> roundup_limit=5. * * For odd values, rounds up, e.g. 3 -> roundup_limit=2. * If radix is 3, 0/3 -> down, 1/3 -> down, 2/3 -> up.

6743. +1 = one retval

6744. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. * * XXX: this is an overkill for some paths, so optimize this later * (or maybe switch to a stack arguments model entirely).

label: code-design

6745. %p' and NULL is portable, but special case it * anyway to get a standard NULL marker in logs.

6746. [... obj key val]

6747. **comment:** XXX: for Object.keys() we should check enumerability of key

label: code-design

6748. idx_value may be < 0 (no value), set and get may be NULL

6749. numeric value (character, count)

6750. Note: first character is checked against this. But because * IdentifierPart includes all IdentifierStart characters, and * the first character (if unescaped) has already been checked * in the if condition, this is OK.

6751. assume 'var' has been eaten
6752. empty separator can always match
6753. * Close environment record(s) if they exist. * * Only variable environments are closed. If lex_env != var_env, it * cannot currently contain any register bound declarations. * * Only environments created for a NEWENV function are closed. If an * environment is created for e.g. an eval call, it must not be closed.
6754. coerce to string
6755. * Not found in registers, proceed to the parent record. * Here we need to determine what the parent would be, * if 'env' was not NULL (i.e. same logic as when initializing * the record). * * Note that environment initialization is only deferred when * DUK_HOBJECT_HAS_NEWENV is set, and this only happens for: * - Function code * - Strict eval code * * We only need to check _Lexenv here; _Varenv exists only if it * differs from _Lexenv (and thus _Lexenv will also be present).
6756. no check
6757. **comment:** XXX: optimize to 16 bytes
label: code-design
6758. * Endian conversion
6759. nrets
6760. * Second (and possibly third) pass. * * Generate actual code. In most cases the need for shuffle * registers is detected during pass 1, but in some corner cases * we'll only detect it during pass 2 and a third pass is then * needed (see GH-115).
6761. embed: integer value
6762. 0=base, 1=esc, 2=class, 3=class+esc
6763. XXX: proper flags?
6764. mark-and-sweep flags automatically active (used for critical sections)
6765. heap string indices are autogenerated in duk_strings.h
6766. some assertions
6767. array writes autoincrement length
6768. **comment:** * Special behavior in E5.1. * * Note that even though parents == 0, the conflicting property * may be an inherited property (currently our global object's * prototype is Object.prototype). Step 5.e first operates on * the existing property (which is potentially in an ancestor) * and then defines a new property in the global object (and * never modifies the ancestor). * * Also note that this logic would become even more complicated * if the conflicting property might be a virtual one. Object * prototype has no virtual properties, though. * * XXX: this is now very awkward, rework.
label: code-design
6769. Avoid zero copy with an invalid pointer. If obj->p is NULL, * the 'new_a' pointer will be invalid which is not allowed even * when copy size is zero.
6770. reserved words: keywords
6771. The #ifdef clutter here handles the JX/JC enable/disable * combinations properly.
6772. [... key val]
6773. **comment:** * Check whether or not we have an error handler. * * We must be careful of not triggering an error when looking up the * property. For instance, if the property is a getter, we don't want * to call it, only plain values are allowed. The value, if it exists, * is not checked. If the value is not a function, a TypeError happens * when it is called and that error replaces the original one.
label: code-design
6774. **comment:** XXX: 'res' setup can be moved to function body level; in fact, two 'res' * intermediate values suffice for parsing of each function. Nesting is needed * for nested functions (which may occur inside expressions).
label: code-design
6775. **comment:** This seems to waste a lot of stack frame entries, but good compilers * will compute these as needed below. Some of these initial flags are * also modified in the code below, so they can't all be removed.
label: code-design
6776. [... env callee varmap]
6777. duk_push_(u)int() is guaranteed to support at least (un)signed 32-bit range
6778. [body forms source template]
6779. User callback did not return source code, so module loading * is finished: just update modLoaded with final module.exports * and we're done.
6780. **comment:** XXX: make active breakpoints actual copies instead of pointers?
label: code-design
6781. XXX: this bound function resolution also happens elsewhere, * move into a shared helper.
6782. **comment:** XXX: non-callable . toJSON() doesn't need to cause an abort * but does at the moment, probably not worth fixing.
label: code-design
6783. num args starting from idx_argbase
6784. caller must change active thread, and set thr->resumer to NULL
6785. check that the valstack has space for the final amount and any * intermediate space needed; this is unoptimal but should be safe
6786. unary plus
6787. * Shortcut for accessing global object properties
6788. * slice()
6789. may be changed
6790. step 3.e: replace 'Desc.[[Value]]'
6791. Captures which are undefined have NULL pointers and are returned * as 'undefined'. The same is done when saved[] pointers are insane * (this should, of course, never happen in practice).
6792. Only direct eval inherits strictness from calling code * (E5.1 Section 10.1.1).
6793. Get a (possibly canonicalized) input character from current sp. The input * itself is never modified, and captures always record non-canonicalized * characters even in case-insensitive matching.
6794. **comment:** * Update idx_retval of current activation. * * Although it might seem this is not necessary (bytecode executor * does this for Ecmascript-to-Ecmascript calls; other calls are * handled here), this turns out to be necessary for handling yield * and resume. For them, an Ecmascript-to-native call happens, and * the Ecmascript call's idx_retval must be set for things to work.
label: code-design
6795. [... varmap]
6796. * Type checking
6797. * ===== * Duktape authors * ===== * * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6dman <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang \u00f3z <llango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6tz <<https://github.com/jaseg>> * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6dman * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstruchtrup> * * Michael Drake (<https://github.com/tlsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn \u00e5s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * ('`sami.vaarala@iki.fi`') and I'll fix the omission.
6798. Grow array part for a new highest array index.

6799. no need to mask idx portion
6800. remove hash entry (no decref)
6801. index
6802. exclusive endpoint
6803. Append bytes from a slice already in the buffer.
6804. this also increases refcount by one
6805. whole, won't clip
6806. [source template]
6807. Cannot overlap.
6808. DUK__L0() cannot be a digit, because the loop doesn't terminate if it is
6809. apparently no hint?
6810. comment: DUK_USE_REGEXP_CANON_WORKAROUND
 label: code-design
6811. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array
6812. XXX: clumsy sign extend and masking of 16 topmost bits
6813. Note: no key on stack
6814. [... retval]
6815. DUK_USE_JX
6816. num args
6817. end of _Formals
6818. no need to pop, nothing was pushed
6819. This would be pointless: we'd return NULL for both lightfuncs and * unexpected types.
6820. 2 when called by debugger
6821. * Top level wrappers
6822. DUK_INTERNAL_H_INCLUDED
6823. Report next pc/line to be executed.
6824. Non-object argument is simply int coerced, matches * V8 behavior (except for "null", which we coerce to * 0 but V8 TypeErrors).
6825. * Build function fixed size 'data' buffer, which contains bytecode, * constants, and inner function references. * * During the building phase 'data' is reachable but incomplete. * Only incref's occur during building (no refzero or GC happens), * so the building process is atomic.
6826. fixed offsets in valstack
6827. * Math built-ins
6828. comment: Temporary structure used to pass a stack allocated region through * duk_safe_call().
 label: code-design
6829. check that there is space for at least one new entry
6830. 0xa0...0xaf
6831. eval left argument first
6832. -> [... closure template]
6833. 1 << 52
6834. must match genhashsizes.py
6835. setter and/or getter may be NULL
6836. * Encoding helpers * * Some of the typing is bytecode based, e.g. slice sizes are unsigned 32-bit * even though the buffer operations will use duk_size_t.
6837. current thread
6838. comment: Called on a read/write error: NULL all callbacks except the detached * callback so that we never accidentally call them after a read/write * error has been indicated. This is especially important for the transport * I/O callbacks to fulfill guaranteed callback semantics.
 label: code-design
6839. Emit 3 bytes and backtrack if there was padding. There's * always space for the whole 3 bytes so no check needed.
6840. Duktape.Thread.yield() should prevent
6841. encode \xffBar as _Bar if no quotes are applied, this is for * readable internal keys.
6842. LICENSE.txt
6843. [... varname val]
6844. string is 'eval' or 'arguments'
6845. * The default property attributes are correct for all * function valued properties of built-in objects now.
6846. Heap udata, used for allocator functions but also for other heap * level callbacks like pointer compression, etc.
6847. IdentifierStart production with ASCII excluded
6848. nothing to mark
6849. comment: XXX: function properties
 label: code-design
6850. prepend regexp to valstack 0 index
6851. act_caller->func may be NULL in some finalization cases, * just treat like we don't know the caller.
6852. convenience
6853. Previous value doesn't need refcount changes because its ownership * is transferred to prev_caller.
6854. * Pop built-ins from stack: they are now INCREF'd and * reachable from the builtins[] array or indirectly * through builtins[].
6855. [obj key value desc]
6856. stmt has non-empty value
6857. -> [... holder name val val ToString(i)]
6858. stage 2: delete configurable entries above target length
6859. When LHS is not register bound, always go through a * temporary. No optimization for top level assignment.
6860. keep key reachable for GC etc; guaranteed not to fail
6861. generate mantissa with a single leading whole number digit
6862. * Complex case conversion helper which decodes a bit-packed conversion * control stream generated by unicode/extract_caseconv.py. The conversion * is very slow because it runs through the conversion data in a linear * fashion to save space (which is why ASCII characters have a special * fast path before arriving here). * * The particular bit counts etc have been determined experimentally to * be small but still sufficient, and must match the Python script *(src/extract_caseconv.py). * * The return value is the case converted codepoint or -1 if the conversion * results in multiple characters (this is useful for regexp Canonicalization * operation). If 'buf' is not NULL, the result codepoint(s) are also * appended to the hbuffer. * * Context and locale specific rules must be checked before consulting * this function.
6863. Stored in duk_heaphdr unused flags.
6864. comment: * Augmenting errors at their creation site and their throw site. * * When errors are created, traceback data is added by built-in code * and a user error handler (if defined) can process or replace the * error. Similarly, when errors are thrown, a user error handler * (if defined) can process or replace the error. * * Augmentation and other processing at error creation time is nice * because an error is only created once, but it may be thrown and * rethrown multiple times. User error handler registered for processing * an error at its throw site must be careful to handle rethrowing in * a useful manner. * * Error augmentation may throw an internal error (e.g. alloc error). * * Ecmascript allows throwing any values, so all values cannot be * augmented. Currently, the built-in augmentation at error creation * only augments error values which are Error instances (= have the * built-in Error.prototype in their prototype chain) and are also * extensible. User error handlers have no limitations in this respect.
 label: code-design
6865. comment: * Emit initializers in sets of maximum max_init_values. * Corner cases such as single value initializers do not have * special handling now. * * Elided elements must not be emitted as 'undefined' values, * because such values would be enumerable (which is incorrect). * Also note that trailing elisions must be reflected in the * length of the final array but cause no elements to be actually * inserted.
 label: code-design

6866. * Initialize built-in objects. Current thread must have a valstack * and initialization errors may longjmp, so a setjmp() catch point * must exist.
6867. terminator
6868. DUK_TOK_EQUALSIGN
6869. * Comparison
6870. DUK_JSON_H_INCLUDED
6871. may have side effects
6872. increases key refcount
6873. on first round, skip
6874. **comment:** * Ecmascript specification algorithm and conversion helpers. ** These helpers encapsulate the primitive Ecmascript operation * semantics, and are used by the bytecode executor and the API * (among other places). Note that some primitives are only * implemented as part of the API and have no "internal" helper. * (This is the case when an internal helper would not really be * useful; e.g. the operation is rare, uses value stack heavily, * etc.) ** The operation arguments depend on what is required to implement * the operation: ** - If an operation is simple and stateless, and has no side * effects, it won't take an duk_hthread argument and its * arguments may be duk_tval pointers (which are safe as long * as no side effects take place). ** - If complex coercions are required (e.g. a "ToNumber" coercion) * or errors may be thrown, the operation takes an duk_hthread * argument. This also implies that the operation may have * arbitrary side effects, invalidating any duk_tval pointers. ** - For operations with potential side effects, arguments can be * taken in several ways: ** a) as duk_tval pointers, which makes sense if the "common case" * can be resolved without side effects (e.g. coercion); the * arguments are pushed to the valstack for coercion if * necessary ** b) as duk_tval values ** c) implicitly on value stack top ** d) as indices to the value stack ** Future work: ** - Argument styles may not be the most sensible in every case now. ** - In-place coercions might be useful for several operations, if * in-place coercion is OK for the bytecode executor and the API.
label: code-design
6875. Note: may be triggered even if minimal new_size would not reach the limit, * plan limit accordingly (taking DUK_VALSTACK_GROW_STEP into account).
6876. **comment:** A little tricky string approach to provide the flags string. * This depends on the specific flag values in duk_reEXP.h, * which needs to be asserted for. In practice this doesn't * produce more compact code than the easier approach in use.
label: code-design
6877. Note: intentionally use approximations to shave a few instructions: * a_used = old_used (accurate: old_used + 1) * a_size = arr_idx (accurate: arr_idx + 1)
6878. Slot C is used in a non-standard fashion (range of regs), * emitter code has special handling for it (must not set the * "no shuffle" flag).
6879. Note: if atom were to contain e.g. captures, we would need to * re-match the atom to get correct captures. Simply quantifiers * do not allow captures in their atom now, so this is not an issue.
6880. * Criteria for augmenting: ** - augmentation enabled in build (naturally) * - error value internal prototype chain contains the built-in * Error prototype object (i.e. 'val instanceof Error') ** Additional criteria for built-in augmenting: ** - error value is an extensible object
6881. **comment:** XXX: solve into closed form (smaller code)
label: code-design
6882. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. ** If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. ** If the final target function cannot be handled by an ecma-to-ecma * call, return to the caller with a return value indicating this case. * The bound chain is resolved and the caller can resume with a plain * function call.
6883. must be signed for loop structure
6884. is_extra
6885. **comment:** * Memory calls: relative to heap, GC interaction, but no error throwing. ** XXX: Currently a mark-and-sweep triggered by memory allocation will run * using the heap->heap_thread. This thread is also used for running * mark-and-sweep finalization; this is not ideal because it breaks the * isolation between multiple global environments. ** Notes: ** - DUK_FREE() is required to ignore NULL and any other possible return * value of a zero-sized alloc/realloc (same as ANSI C free()). ** - There is no DUK_REALLOC_ZEROED because we don't assume to know the * old size. Caller must zero the reallocated memory. ** - DUK_REALLOC_INDIRECT() must be used when a mark-and-sweep triggered * by an allocation failure might invalidate the original 'ptr', thus * causing a realloc retry to use an invalid pointer. Example: we're * reallocating the value stack and a finalizer resizes the same value * stack during mark-and-sweep. The indirect variant requests for the * current location of the pointer being reallocated using a callback * right before every realloc attempt; this circuitous approach is used * to avoid strict aliasing issues in a more straightforward indirect * pointer (void **) approach. Note: the pointer in the storage * location is read but is NOT updated; the caller must do that.
label: code-design
6886. Here 'x' is a Unicode codepoint
6887. [ToUInt32(len) array ToUInt32(len)] -> [ToUInt32(len) array]
6888. Force a resize so that DELETED entries are eliminated. * Another option would be duk_recheck_stab_size_probe(); * but since that happens on every intern anyway, this whole * check can now be disabled.
6889. * comp_ctx->curr_func is now ready to be converted into an actual * function template.
6890. Potentially truncated, NUL not guaranteed in any case. * The (int_rc < 0) case should not occur in practice.
6891. If not found, resume existence check from Function.prototype. * We can just substitute the value in this case; nothing will * need the original base value (as would be the case with e.g. * setters/getters).
6892. "-123456|0"
6893. Get the original arguments. Note that obj_index may be a relative * index so the stack must have the same top when we use it.
6894. curr value
6895. These are for rate limiting Status notifications and transport peeking.
6896. * Some E5/E5.1 algorithms require that array indices are iterated * in a strictly ascending order. This is the case for e.g. * Array.prototype.forEach() and JSON.stringify() PropertyList * handling. ** To ensure this property for arrays with an array part (and * arbitrary objects too, since e.g. forEach() can be applied * to an array), the caller can request that we sort the keys * here.
6897. DUK_USE_LEXER_SLIDING_WINDOW
6898. DUK_TOK_NULL
6899. * Must be careful to catch errors related to value stack manipulation * and property lookup, not just the call itself.
6900. The file/line arguments are NULL and 0, they're ignored by DUK_ERROR_RAW() * when non-verbose errors are used.
6901. **comment:** 8-byte format could be: * 1111 1111 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [41 bits] ** However, this format would not have a zero bit following the * leading one bits and would not allow 0xFF to be used as an * "invalid utf-8" marker for internal keys. Further, 8-byte * encodings (up to 41 bit code points) are not currently needed.
label: code-design
6902. -> [... obj value]
6903. [key trap_result] -> []
6904. replace in stack
6905. (maybe) truncated
6906. nothing to process
6907. Encode string in small chunks, estimating the maximum expansion so that * there's no need to ensure space while processing the chunk.
6908. * Object.getOwnPropertyDescriptor() (E5 Sections 15.2.3.3, 8.10.4) ** This is an actual function call.
6909. knock back "next temp" to this whenever possible
6910. * Detected a directive
6911. 'for'
6912. complete the millisecond field
6913. 'It is true'
6914. Pre/post inc/dec for register variables, important for loops.
6915. **comment:** Lightfunc name, includes Duktape/C native function pointer, which * can often be used to locate the function from a symbol table. * The name also includes the 16-bit duk_tval flags field because it * includes the magic value. Because a single native function often * provides different functionality depending on the magic value, it * seems reasonably to include it in the name. ** On the other hand, a complicated name increases string table * pressure in low memory environments (but only when function name * is accessed).

label: code-design

6916. **comment:** XXX: if duk_hobject_define_property_internal() was updated * to handle a pre-existing accessor property, this would be * a simple call (like for the ancestor case).

label: code-design

6917. Iteration solution

6918. **comment:** 16 bits would be enough for shared heaphdr flags and duk_hstring * flags. The initial parts of duk_heaphdr_string and duk_heaphdr * must match so changing the flags field size here would be quite * awkward. However, to minimize struct size, we can pack at least * 16 bits of duk_hstring data into the flags field.

label: code-design

6919. * Helper tables

6920. 'this'

6921. **comment:** XXX: could check for e16 == 0 because NULL is guaranteed to * encode to zero.

label: code-design

6922. approximate

6923. Avoid NaN-to-integer coercion as it is compiler specific.

6924. Reset to zero size, keep current limit.

6925. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

6926. callback for indirect reallocs, request for current pointer

6927. [... this]

6928. * Portable 12-byte representation

6929. avoid multiple computations of flags address; bypasses macros

6930. **comment:** Reg/const access macros: these are very footprint and performance sensitive * so modify with care.

label: code-design

6931. must not truncate

6932. prefix matches, lengths matter now

6933. Careful with reachability here: don't pop 'obj' before pushing * proxy target.

6934. * Replace valstack top with case converted version.

6935. Shuffle decision not changed.

6936. * Stack index validation/normalization and getting a stack duk_tval ptr. ** These are called by many API entrypoints so the implementations must be * fast and "inlined". ** There's some repetition because of this; keep the functions in sync.

6937. numeric value of token

6938. [... constructor arg1 ... argN] -> [retval]

6939. * Rebuild the hash part always from scratch (guaranteed to finish). ** Any resize of hash part requires rehashing. In addition, by rehashing * get rid of any elements marked deleted (DUK_HASH_DELETED) which is critical * to ensuring the hash part never fills up.

6940. temps

6941. 'Object'

6942. not strictly necessary

6943. IsDataDescriptor(desc) == true

6944. POSTFIX EXPRESSION

6945. If argument count is 1 and first argument is a buffer, write the buffer * as raw data into the file without a newline; this allows exact control * over stdout/stderr without an additional entrypoint (useful for now). ** Otherwise current print/alert semantics are to ToString() coerce * arguments, join them with a single space, and append a newline.

6946. * Exposed regexp compilation primitive. ** Sets up a regexp compilation context, and calls duk_parse_disjunction() to do the * actual parsing. Handles generation of the compiled regexp header and the * "boilerplate" capture of the matching substring (save 0 and 1). Also does some * global level regexp checks after recursive compilation has finished. ** An escaped version of the regexp source, suitable for use as a RegExp instance * 'source' property (see E5 Section 15.10.3), is also left on the stack. ** Input stack: [pattern flags] * Output stack: [bytecode escaped_source] (both as strings)

6947. 'Error'

6948. * Flags for __FILE__ / __LINE__ registered into tracedata

6949. Count actually used entry part entries (non-NULL keys).

6950. DUK_TOK_ARSHIFT

6951. **comment:** XXX: possibly incorrect handling of empty expression

label: code-design

6952. used entries + approx 100% -> reset load to 50%

6953. '\xffRegbase'

6954. is_decl

6955. but don't allow leading plus

6956. **comment:** * String cache should ideally be at duk_hthread level, but that would * cause string finalization to slow down relative to the number of * threads; string finalization must check the string cache for "weak" * references to the string being finalized to avoid dead pointers. ** Thus, string caches are now at the heap level now.

label: code-design

6957. For manual debugging: instruction count based on executor and * interrupt counter book-keeping. Inspect debug logs to see how * they match up.

6958. [... holder name val enum obj_key val obj_key]

6959. lj.value1 is already set

6960. number of arguments allocated to regs

6961. Optimized for size.

6962. * Default fatal error handler

6963. named "arithmetic" because result is signed

6964. Default implicit return value.

6965. lookup name from current activation record's functions' registers

6966. custom; implies DUK_HOBJECT_IS_BUFFEROBJECT

6967. max # of key-value pairs initialized in one MPUTOBJ set

6968. unicode code points, window[0] is always next

6969. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.

6970. thread not currently running

6971. Refzero head is still the same. This is the case even if finalizer * inserted more refzero objects; they are inserted to the tail.

6972. [... key arg1 ... argN]

6973. * GETIDREF: A GetIdentifierReference-like helper. ** Provides a parent traversing lookup and a single level lookup * (for HasBinding). ** Instead of returning the value, returns a bunch of values allowing * the caller to read, write, or delete the binding. Value pointers * are duk_tval pointers which can be mutated directly as long as * refcounts are properly updated. Note that any operation which may * reallocate valstacks or compact objects may invalidate the returned * duk_tval (but not object) pointers, so caller must be very careful. ** If starting environment record 'env' is given, 'act' is ignored. * However, if 'env' is NULL, the caller may identify, in 'act', an * activation which hasn't had its declarative environment initialized * yet. The activation registers are then looked up, and its parent * traversed normally. ** The 'out' structure values are only valid if the function returns * success (non-zero).

6974. Ignore the normalize/validate helper outputs on the value stack, * they're popped automatically.

6975. **comment:** A union is used here as a portable struct size / alignment trick: * by adding a 32-bit or a 64-bit (unused) union member, the size of * the struct is effectively forced to be a multiple of 4 or 8 bytes * (respectively) without increasing the size of the struct unless * necessary.

label: code-design

6976. Chop extra retvals away / extend with undefined.

6977. MPUTARR emitted by outer loop

6978. **comment:** XXX: fastint?
 label: code-design
6979. compact the exports table
6980. Code is not accepted before the first case/default clause
6981. Process one debug message. Automatically restore value stack top to its * entry value, so that individual message handlers don't need exact value * stack handling which is convenient.
6982. -> [... key val replacer holder key val]
6983. Because new_size != 0, if condition doesn't need to be * (new_valstack != NULL || new_size == 0).
6984. No debugger support.
6985. '<'
6986. treat like property not found
6987. Bound functions don't have all properties so we'd either need to * lookup the non-bound target function or reject bound functions. * For now, bound functions are rejected.
6988. Carry is detected based on wrapping which relies on exact 32-bit * types.
6989. eat (first) input slash
6990. value resides in a register or constant
6991. **comment:** "get" and "set" are tokens but NOT ReservedWords. They are currently * parsed and identifiers and these defines are actually now unused.
 label: code-design
6992. Overflow, relevant mainly when listlen is 16 bits.
6993. [... val] -> [... enum]
6994. E5.1 standard behavior when deleteCount is not given would be * to treat it just like if 'undefined' was given, which coerces * ultimately to 0. Real world behavior is to splice to the end * of array, see test-bi-arrayproto-splice-no-delcount.js.
6995. case 0: nop
6996. **comment:** XXX: the duk_hobject_enum.c stack APIs should be reworked
 label: code-design
6997. Error: try coercing error to string once.
6998. for resizing of array part, use slow path
6999. y == -Infinity
7000. Initialization and finalization (compaction), converting to other types.
7001. must match exactly the number of internal properties inserted to enumerator
7002. required
7003. + spare
7004. All element accessors are host endian now (driven by TypedArray spec).
7005. Push an arbitrary duk_tval to the stack, coerce it to string, and return * both a duk_hstring pointer and an array index (or DUK__NO_ARRAY_INDEX).
7006. must "goto restart_execution", e.g. breakpoints changed
7007. Avoid side effects that might disturb curr.e_idx until * we're done editing the slot.
7008. 23: getUTCSeconds
7009. no need to normalize
7010. * Windows Date providers ** Platform specific links: ** - [http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473(v=vs.85).aspx)
7011. **comment:** XXX: Change 'anon' handling here too, to use empty string for anonymous functions?
 label: code-design
7012. 'this binding' is just under bottom
7013. Default exports table
7014. **comment:** XXX: macro smaller than call?
 label: code-design
7015. XXX: assert? compiler is responsible for this never happening
7016. pc of label statement: * pc+1: break jump site * pc+2: continue jump site
7017. require'
7018. * Assertion helpers
7019. multiple codepoint conversion or non-ASCII mapped to ASCII * --> leave as is.
7020. Computation must not wrap, only srclen + 3 is at risk of * wrapping because after that the number gets smaller. * This limit works for 32-bit size_t: * 0x100000000
 - 3 - 1 = 4294967292
7021. 19: getUTCMinutes
7022. Inspect a property using a virtual index into a conceptual property list * consisting of (1) all array part items from [0,a_size[(even when above * .length) and (2) all entry part items from [0,e_next[. Unused slots are * indicated using dvalue 'unused'.
7023. duk_get_min_grow_e() is always >= 1
7024. * Reachable object, keep
7025. 'eval'
7026. Array part may be larger than 'length'; if so, iterate * only up to array 'length'.
7027. regCatch+0 and regCatch+1 are reserved for TRYCATCH
7028. resolved id: require(id) must return this same module
7029. We rely on a few object flag / class number relationships here, * assert for them.
7030. default case control flow patchup; note that if pc_prevcase < 0 * (i.e. no case clauses), control enters default case automatically.
7031. DUK_HEAP_H_INCLUDED
7032. note: original, uncoerced base
7033. If len <= 1, middle will be 0 and for-loop bails out * immediately (0 < 0 -> false).
7034. DUK_OP_NONE marks a 'plain' assignment
7035. resume execution from pc_base or pc_base+1 (points to 'func' bytecode, stable pointer)
7036. Stack size decreases.
7037. heap->strs: not worth dumping
7038. -> [... closure template env funcname closure]
7039. step 12
7040. Section 7.8.3 (note): NumericLiteral must be followed by something other than * IdentifierStart or DecimalDigit.
7041. **comment:** XXX: duk_ssize_t
 label: code-design
7042. DUK_ERR_UNCAUGHT_ERROR: no macros needed
7043. -> [ToObject(this) item1 ... itemN arr]
7044. C call site gets blamed next, unless flagged not to do so. * XXX: file/line is disabled in minimal builds, so disable this * too when appropriate.
7045. * Debug dump type sizes
7046. needed for line number tracking (becomes prev_token.start_line)
7047. don't coerce yet to a plain value (variant 3 needs special handling)
7048. **comment:** XXX: opcode specific assertions on when consts are allowed
 label: code-design
7049. Read value pushed on stack.
7050. * When resizing the valstack, a mark-and-sweep may be triggered for * the allocation of the new valstack. If the mark-and-sweep needs * to use our thread for something, it may cause *the same valstack* * to be resized recursively. This happens e.g. when mark-and-sweep * finalizers are called. This is taken into account carefully in * duk_resize_valstack(). * * 'new_size' is known to be <= valstack_max, which ensures that * size_t and pointer arithmetic won't wrap in duk_resize_valstack().

7051. 'null' enumerator case -> behave as with an empty enumerator
7052. Object literal set/get functions have a name (property * name) but must not have a lexical name binding, see * test-bug-getset-func-name.js.
7053. Current compiler state (if any), used for augmenting SyntaxErrors.
7054. push to stack
7055. Note: if DecimalLiteral starts with a '0', it can only be * followed by a period or an exponent indicator which starts * with 'e' or 'E'. Hence the if-check above ensures that * OctalIntegerLiteral is the only valid NumericLiteral * alternative at this point (even if y is, say, '9').
7056. 'catch'
7057. * Special cases like NaN and +/- Infinity are handled explicitly * because a plain C coercion from double to int handles these cases * in undesirable ways. For instance, NaN may coerce to INT_MIN * (not zero), and INT_MAX + 1 may coerce to INT_MIN (not INT_MAX). * * This double-to-int coercion differs from ToInteger() because it * has a finite range (ToInteger() allows e.g. +/- Infinity). It * also differs fromToInt32() because the INT_MIN/INT_MAX clamping * depends on the size of the int type on the platform. In particular, * on platforms with a 64-bit int type, the full range is allowed.
7058. element size shift: * 0 = u8/18 * 1 = u16/16 * 2 = u32/32/float * 3 = double
7059. curr_pc synced back above
7060. hash size approx. 1.25 times entries size
7061. Write in little endian
7062. source starts after dest ends
7063. **comment:** * Debugging macro calls.
 label: code-design
7064. DUK_USE_VERBOSE_ERRORS
7065. 0xxx xxxx -> fast path
7066. * Ecmascript modulus (%) does not match IEEE 754 "remainder" * operation (implemented by remainder() in C99) but does seem * to match ANSI C fmod(). * * Compare E5 Section 11.5.3 and "man fmod".
7067. unsigned shift
7068. Not a very good provider: only full seconds are available.
7069. 110x xxxx 10xx xxxx [11 bits]
7070. Note: errors are augmented when they are created, not * when they are thrown. So, don't augment here, it would * break re-throwing for instance.
7071. **comment:** Not necessary to unwind catchstack: return handling will * do it. The finally flag of 'cat' is no longer set. The * catch flag may be set, but it's not checked by return handling.
 label: code-design
7072. ensure value is 1 or 0 (not other non-zero)
7073. Accept plain buffer values like array initializers * (new in Duktape 1.4.0).
7074. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
7075. Indicate function type in the function body using a dummy * directive.
7076. 'ObjEnv'
7077. lj.value1 already set
7078. millisec -> 100ns units since jan 1, 1970
7079. **comment:** * Note: must use indirect variant of DUK_REALLOC() because underlying * pointer may be changed by mark-and-sweep.
 label: code-design
7080. DUK_TOK_SWITCH
7081. class Object, extensible
7082. idx_end
7083. _Source
7084. 1 1 0 <8 bits>
7085. **comment:** two casts to avoid gcc warning: "warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]"
 label: code-design
7086. minimum new length is highest_arr_idx + 1
7087. DUK_TOK_RBRACKET
7088. property is configurable and
7089. * Matching complete, create result array or return a 'null'. Update lastIndex * if necessary. See E5 Section 15.10.6.2. * * Because lastIndex is a character (not byte) offset, we need the character * length of the match which we conveniently get as a side effect of interning * the matching substring (0th index of result array). * * saved[0] start pointer (~ byte offset) of current match * saved[1] end pointer (~ byte offset) of current match (exclusive) * char_offset start character offset of current match (-> .index of result) * char_end_offset end character offset (computed below)
7090. **comment:** XXX: Refactor key coercion so that it's only called once. It can't * be trivially lifted here because the object must be type checked * first.
 label: code-design
7091. Caller must ensure 'tv' is indeed a double and not a fastint!
7092. Caller has already eaten the first character ('l') which we don't need.
7093. * String scanning helpers * * All bytes other than UTF-8 continuation bytes ([0x80,0xbff]) are * considered to contribute a character. This must match how string * character length is computed.
7094. [... func this | arg1 ... argN] ('this' must precede new bottom)
7095. structs from duk_forwdecl.h
7096. assumed to also be PC of "LABEL"
7097. Check that prev/next links are consistent: if e.g. h->prev is != NULL, * h->prev->next should point back to h.
7098. {'_undef':true}'
7099. DUK_TOK_ADD_EQ
7100. roughly 1.0 kB -> but rounds up to DUK_VALSTACK_GROW_STEP in practice
7101. remove the original 'template' atom
7102. The ANSI C pow() semantics differ from Ecmascript. * * E.g. when x==1 and y is +/- infinite, the Ecmascript required * result is NaN, while at least Linux pow() returns 1.
7103. * Heap native function representation.
7104. Note: DUK_HEAP_HAS_REFZERO_FREE_RUNNING(heap) may be true; a refcount * finalizer may trigger a mark-and-sweep.
7105. 'Invalid Date'
7106. dynamic
7107. **comment:** * Node.js Buffer.concat()
 label: code-design
7108. **comment:** XXX: substring in-place at idx_place?
 label: code-design
7109. * Object's finalizer was executed on last round, and * object has been happily rescued.
7110. embed: 0 or 1 (false or true)
7111. Labels can be used for iteration statements but also for other statements, * in particular a label can be used for a block statement. All cases of a * named label accept a 'break' so that flag is set here. Iteration statements * also allow 'continue', so that flag is updated when we figure out the * statement type.
7112. string 2 of token (borrowed, stored to ctx->slot2_idx)
7113. UNARY EXPRESSIONS
7114. * Not found (even in global object). * * In non-strict mode this is a silent SUCCESS (!), see E5 Section 11.4.1, * step 3.b. In strict mode this case is a compile time SyntaxError so * we should not come here.
7115. should actually never happen, but check anyway
7116. input offset for window leading edge (not window[0])
7117. * Init object properties * * Properties should be added in decreasing order of access frequency. * (Not very critical for function templates.)

7118. [... fallback constructor fallback(this) arg1 ... argN]; * Note: idx_cons points to first 'fallback', not 'constructor'.
7119. **comment:** * Canonicalize a range, generating result ranges as necessary. * Needs to exhaustively scan the entire range (at most 65536 * code points). If 'direct' is set, caller (lexer) has ensured * that the range is already canonicalization compatible (this * is used to avoid unnecessary canonicalization of built-in * ranges like \W, which are not affected by canonicalization). ** NOTE: here is one place where we don't want to support chars * outside the BMP, because the exhaustive search would be * massively larger.
label: code-design
7120. continuation byte
7121. Note: the realloc may have triggered a mark-and-sweep which may * have resized our valstack internally. However, the mark-and-sweep * MUST NOT leave the stack bottom/top in a different state. Particular * assumptions and facts: * * - The thr->valstack pointer may be different after realloc, * and the offset between thr->valstack_end <> thr->valstack * may have changed. * - The offset between thr->valstack_bottom <> thr->valstack * and thr->valstack_top <> thr->valstack MUST NOT have changed, * because mark-and-sweep must adhere to a strict stack policy. * In other words, logical bottom and top MUST NOT have changed. * - All values above the top are unreachable but are initialized * to UNDEFINED, up to the post-realloc valstack_end. * 'old_end_offset' must be computed after realloc to be correct.
7122. Note: hstring is in heap but has refcount zero and is not strongly reachable. * Caller should increase refcount and make the hstring reachable before any * operations which require allocation (and possible gc).
7123. **comment:** * "name" is a non-standard property found in at least V8, Rhino, smjs. * For Rhino and smjs it is non-writable, non-enumerable, and non-configurable; * for V8 it is writable, non-enumerable, non-configurable. It is also defined * for an anonymous function expression in which case the value is an empty string. * We could also leave name 'undefined' for anonymous functions but that would * differ from behavior of other engines, so use an empty string. ** XXX: make optional? costs something per function.
label: code-design
7124. **comment:** * duk_hbuffer operations such as resizing and inserting/appending data to * a dynamic buffer. ** Append operations append to the end of the buffer and they are relatively * efficient: the buffer is grown with a "spare" part relative to the buffer * size to minimize reallocations. Insert operations need to move existing * data forward in the buffer with memmove() and are not very efficient. * They are used e.g. by the regexp compiler to "backpatch" regexp bytecode.
label: code-design
7125. steps 11.c.ii.1 - 11.c.ii.4, but our internal book-keeping * differs from the reference model
7126. ASCII fast path
7127. Most input bytes go through here.
7128. -> [... holder name val]
7129. Assume that year, month, day are all coerced to whole numbers. * They may also be NaN or infinity, in which case this function * must return NaN or infinity to ensure time value becomes NaN. * If 'day' is NaN, the final return will end up returning a NaN, * so it doesn't need to be checked here.
7130. useful for debugging
7131. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()! ** A conservative approach would be to use duk_ivalue_totempconst() * for 'left'. However, allowing a reg-bound variable seems safe here * and is nice because "foo.bar" is a common expression. If the ivalue * is used in an expression a GETPROP will occur before any changes to * the base value can occur. If the ivalue is used as an assignment * LHS, the assignment code will ensure the base value is safe from * RHS mutation.
7132. We should never exit the loop above.
7133. may have FINALIZED
7134. LAYOUT 3
7135. **comment:** unused
label: code-design
7136. these computations are guaranteed to be exact for the valid * E5 time value range, assuming milliseconds without fractions.
7137. 'Int32Array'
7138. an error handler (user callback to augment/replace error) is running
7139. without refcounts
7140. arbitrary marker, outside valid exp range
7141. extra -1 for buffer
7142. For invalid characters the value -1 gets extended to * at least 16 bits. If either nybble is invalid, the * resulting 't' will be < 0.
7143. required to guarantee success of rehashing, * intentionally use unadjusted new_e_size
7144. XXX: A fast path for usual integers would be useful when * fastint support is not enabled.
7145. -> [... ToObject(this) ToUint32(length) final_len final_len]
7146. [...val] -> [...]
7147. allow empty expression
7148. [... closure template name]
7149. # total registers target function wants on entry (< 0 => "as is")
7150. **comment:** unnecessary shift for last byte
label: code-design
7151. varenv remains reachable through 'obj'
7152. Special coercion for Uint8ClampedArray.
7153. valstack slot for temp buffer
7154. * Node.js Buffer: constructor
7155. this optimization is important to handle negative literals * (which are not directly provided by the lexical grammar)
7156. Nothing worked -> not equal.
7157. E5 Section 11.13.1 (and others for other assignments), step 4.
7158. Note: unshift() may operate on indices above unsigned 32-bit range * and the final length may be >= 2**32. However, we restrict the * final result to 32-bit range for practicality.
7159. String and buffer enumeration behavior is identical now, * so use shared handler.
7160. 37x32 = 1184 bits
7161. heap->dbg_detached_cb: keep
7162. This check used to be for (t < 0) but on some platforms * time_t is unsigned and apparently the proper way to detect * an mktime() error return is the cast above. See e.g.: * <http://pubs.opengroup.org/onlinepubs/009695299/functions/mktime.html>
7163. **comment:** Fast variants, inline refcount operations except for refzero handling. * Can be used explicitly when speed is always more important than size. * For a good compiler and a single file build, these are basically the * same as a forced inline.
label: code-design
7164. [... key] -> [...]
7165. * Use current token to decide whether a RegExp can follow. ** We can use either 't' or 't_nores'; the latter would not * recognize keywords. Some keywords can be followed by a * RegExp (e.g. "return"), so using 't' is better. This is * not trivial, see doc/compiler.rst.
7166. Khronos/ES6 requires zeroing even when DUK_USE_ZERO_BUFFER_DATA * is not set.
7167. * Refcount memory freeing loop. ** Frees objects in the refzero_pending list until the list becomes * empty. When an object is freed, its references get decref'd and * may cause further objects to be queued for freeing. ** This could be expanded to allow incremental freeing: just bail out * early and resume at a future alloc/decref/refzero.
7168. Note: 'fast' breaks will jump to pc_label_site + 1, which will * then jump here. The double jump will be eliminated by a * peephole pass, resulting in an optimal jump here. The label * site jumps will remain in bytecode and will waste code size.
7169. **comment:** * Prototype walking starting from 'env'. ** ('act' is not needed anywhere here.)
label: code-design
7170. mark range 'direct', bypass canonicalization (see Wiki)
7171. Do debugger forwarding before raw() because the raw() function * doesn't get the log level right now.
7172. estimate year upwards (towards positive infinity), then back down; * two iterations should be enough
7173. return module.exports

7174. * Parsing an expression starting with 'new' is tricky because * there are multiple possible productions deriving from * LeftHandSideExpression which begin with 'new'. ** We currently resort to one-token lookahead to distinguish the * cases. Hopefully this is correct. The binding power must be * such that parsing ends at an LPAREN (CallExpression) but not at * a PERIOD or LBRACKET (MemberExpression). ** See doc/compiler.rst for discussion on the parsing approach, * and testcases/test-dev-new.js for a bunch of documented tests.

7175. exropop is the top level variant which resets nud/led counts

7176. control props

7177. push value to stack

7178. * shift()

7179. k+argCount-1; note that may be above 32-bit range

7180. * Assert macro: failure causes panic.

7181. **comment:** Similar workaround for INFINITY.
label: code-design

7182. * Fallback marking handler if recursion limit is reached. ** Iterates 'temproots' until recursion limit is no longer hit. Note * that temproots may reside either in heap allocated list or the * refzero work list. This is a slow scan, but guarantees that we * finish with a bounded C stack. ** Note that nodes may have been marked as temproots before this * scan begun, OR they may have been marked during the scan (as * we process nodes recursively also during the scan). This is * intended behavior.

7183. **comment:** If number would need zero padding (for whole number part), use * exponential format instead. E.g. if input number is 12300, 3 * digits are generated ("123"), output "1.23e+4" instead of "12300". * Used for toPrecision().
label: code-design

7184. number

7185. Must be element size multiple from * start offset to end of buffer.

7186. empty ("") is allowed in some formats (e.g. Number()), as zero

7187. slow path is shared

7188. * If matching with a regexp: * - non-global RegExp: lastIndex not touched on a match, zeroed * on a non-match * - global RegExp: on match, lastIndex will be updated by regexp * executor to point to next char after the matching part (so that * characters in the matching part are not matched again) ** If matching with a string: * - always non-global match, find first occurrence ** We need: * - The character offset of start-of-match for the replacer function * - The byte offsets for start-of-match and end-of-match to implement * the replacement values \$\$, \$, and \$', and to copy non-matching * input string portions (including header and trailer) verbatim. ** NOTE: the E5.1 specification is a bit vague how the RegExp should * behave in the replacement process; e.g. is matching done first for * all matches (in the global RegExp case) before any replacer calls * are made? See: test-bi-stringproto-replace.js for discussion.

7189. -> [... func]

7190. * Various defines and file specific helper macros

7191. no net exponent

7192. **comment:** integer, but may be +/- Infinite, +/- zero (not NaN, though)
label: code-design

7193. 11 bits

7194. **comment:** Lookup active label information. Break/continue distinction is necessary to handle switch * statement related labels correctly: a switch will only catch a 'break', not a 'continue'. ** An explicit label cannot appear multiple times in the active set, but empty labels (unlabelled * iteration and switch statements) can. A break will match the closest unlabelled or labelled * statement. A continue will match the closest unlabelled or labelled iteration statement. It is * a syntax error if a continue matches a labelled switch statement; because an explicit label cannot * be duplicated, the continue cannot match any valid label outside the switch. ** A side effect of these rules is that a LABEL statement related to a switch should never actually * catch a continue abrupt completion at run-time. Hence an INVALID opcode can be placed in the * continue slot of the switch's LABEL statement.
label: code-design

7195. get before side effects

7196. DUK_HSTRING_H_INCLUDED

7197. avoid empty label at the end of a compound statement

7198. no specific action, resume normal execution

7199. for manual torture testing: tight allocation, useful with valgrind

7200. XXX: skip count_free w/o debug?

7201. everything except object stay as is

7202. identifiers and environment handling

7203. back to fast loop

7204. DUK_USE_JX || DUK_USE_JC

7205. Run fake finalizer. Avoid creating new refzero queue entries * so that we are not forced into a forever loop.

7206. In some cases it may be that lo > hi, or hi < 0; these * degenerate cases happen e.g. for empty arrays, and in * recursion leaves.

7207. is resume, not a tail call

7208. * Basic stack manipulation: swap, dup, insert, replace, etc

7209. **comment:** Note: nargs (and nregs) may be negative for a native, * function, which indicates that the function wants the * input stack "as is" (i.e. handles "vararg" arguments).
label: code-design

7210. * Indirect magic value lookup for Date methods. ** Date methods don't put their control flags into the function magic value * because they wouldn't fit into a LIGHTFUNC's magic field. Instead, the * magic value is set to an index pointing to the array of control flags * below. ** This must be kept in strict sync with genbuiltins.py!

7211. strings may have inner refs (extdata) in some cases

7212. class masks

7213. keep label catcher

7214. state updated, restart bytecode execution

7215. Flags for duk_js_equals_helper().

7216. [... func this <some bound args> arg1 ... argN _Args]

7217. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module result(ignored)]

7218. DUK_USE_TAILCALL

7219. **comment:** should never be called
label: code-design

7220. denormal

7221. -> loop body

7222. note: this updates refcounts

7223. avoid calling at end of input, will DUK_ERROR (above check suffices to avoid this)

7224. t1 <- (+ r m+)

7225. denormal or zero

7226. mark-and-sweep control

7227. A 'return' statement is only allowed inside an actual function body, * not as part of eval or global code.

7228. tailcall cannot be flagged to resume calls, and a * previous frame must exist

7229. DUK_TOK_LET

7230. Remove a slice from inside buffer.

7231. valstack slot for 2nd token value

7232. one-based -> zero-based

7233. Set .buffer

7234. terminate

7235. max size: radix=2 + sign

7236. remaining actual steps are carried out if standard DefineOwnProperty succeeds

7237. **comment:** The prev/next pointers of the removed duk_heapheader are left as garbage. * It's up to the caller to ensure they're written before inserting the * object back.

label: code-design

7238. '{"_func":true}'

7239. **comment:** XXX: support unary arithmetic ivalue (useful?)

label: code-design

7240. E5 Section 15.5.5.2

7241. True if slice is full, i.e. offset is zero and length covers the entire * buffer. This status may change independently of the duk_hbufferobject if * the underlying buffer is dynamic and changes without the hbufferobject * being changed.

7242. avoid calling write callback in detach()

7243. http://en.wikipedia.org/wiki/Replacement_character#Replacement_character

7244. -> [Object res]

7245. * Property attribute defaults are defined in E5 Section 15 (first * few pages); there is a default for all properties and a special * default for 'length' properties.
Variation from the defaults is * signaled using a single flag bit in the bitstream.

7246. IdentifierStart production with Letter and ASCII excluded

7247. **comment:** * DecimalLiteral, HexIntegerLiteral, OctalIntegerLiteral * "pre-parsing", followed by an actual, accurate parser step. * * Note: the leading sign character ('+' or '-') is -not- part of * the production in E5 grammar, and that the a DecimalLiteral * starting with a '0' must be followed by a non-digit. Leading * zeroes are syntax errors and must be checked for. * * XXX: the two step parsing process is quite awkward, it would * be more straightforward to allow numconv to parse the longest * valid prefix (it already does that, it only needs to indicate * where the input ended). However, the lexer decodes characters * using a lookup window, so this is not a trivial change.

label: code-design

7248. * The escapes are same as outside a character class, except that \b has a * different meaning, and \B and backreferences are prohibited (see E5 * Section 15.10.2.19). However, it's difficult to share code because we * handle e.g. "\n" very differently: here we generate a single character * range for it.

7249. **comment:** Ensure there are no stale active breakpoint pointers. * Breakpoint list is currently kept - we could empty it * here but we'd need to handle refcounts correctly, and * we'd need a 'thr' reference for that. * * XXX: clear breakpoint on either attach or detach?

label: code-design

7250. end switch

7251. DUK_TOK_LE

7252. XXX: Currently no inspection of threads, e.g. value stack, call * stack, catch stack, etc.

7253. [... v1(func) v2(pc+flags)]

7254. Matches both +0 and -0 on purpose.

7255. Note: ToPrimitive(object,hint) == [[DefaultValue]](object,hint), * so use [[DefaultValue]] directly.

7256. The code for writing reg_temps + 0 to the left hand side has already * been emitted.

7257. start line of token (first char)

7258. 10: getMonth

7259. If caller is global/eval code, 'caller' should be set to * 'null'. * * XXX: there is no exotic flag to infer this correctly now. * The NEWENV flag is used now which works as intended for * everything (global code, non-strict eval code, and functions) * except strict eval code. Bound functions are never an issue * because 'func' has been resolved to a non-bound function.

7260. x <- y

7261. eat 'var'

7262. cannot have >4G captures

7263. **comment:** not caught by current thread, thread terminates (yield error to resumer); * note that this may cause a cascade if the resumer terminates with an uncaught * exception etc (this is OK, but needs careful testing)

label: code-design

7264. normalize NaN which may not match our canonical internal NaN

7265. Recompute argument count: bound function handling may have shifted.

7266. [...] key val] -> [...]

7267. minval

7268. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

7269. DUK_TOK_ARSHIFT_EQ

7270. Slightly smaller code without explicit flag, but explicit flag * is very useful when 'clen' is dropped.

7271. Here again we parse bytes, and non-ASCII UTF-8 will cause end of * parsing (which is correct except if there are non-shortest encodings). * There is also no need to check explicitly for end of input buffer as * the input is NUL padded and NUL will exit the parsing loop. * * Because no unescaping takes place, we can just scan to the end of the * plain string and intern from the input buffer.

7272. **comment:** XXX: merge this with duk_js_call.c, as this function implements * core semantics (or perhaps merge the two files altogether).

label: code-design

7273. * New length lower than old length => delete elements, then * update length. * * Note: even though a bunch of elements have been deleted, the 'desc' is * still valid as properties haven't been resized (and entries compacted).

7274. * Program code (global and eval code) has an implicit return value * from the last statement value (e.g. eval("1; 2+3;") returns 3). * This is not the case with functions. If implicit statement return * value is requested, all statements are coerced to a register * allocated here, and used in the implicit return statement below.

7275. * Pass 1

7276. Use unsigned arithmetic to optimize comparison.

7277. type and control flags, label number

7278. DUK_TOK_RSHIFT_EQ

7279. probe steps (see genhashsizes.py), currently assumed to be 32 entries long * (DUK_UTIL_GET_HASH_PROBE_STEP macro).

7280. [...] enum_target_res trap_result]

7281. 1111 1110 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [36 bits]

7282. * Conversions and coercions * * The conversion/coercions are in-place operations on the value stack. * Some operations are implemented here directly, while others call a * helper in duk_js_ops.c after validating arguments.

7283. **comment:** Select a thread for mark-and-sweep use. * * XXX: This needs to change later.

label: code-design

7284. Automatic filename: 'eval' or 'input'.

7285. cumulative opcode execution count (overflows are OK)

7286. Fast path for fastints

7287. **comment:** XXX: any chance of merging these three similar but still slightly * different algorithms so that footprint would be reduced?

label: code-design

7288. comp_ctx->lex has been pre-initialized by caller: it has been * zeroed and input/input_length has been set.

7289. 'input'

7290. countdown state

7291. switch

7292. * Return to bytecode executor, which will resume execution from * the topmost activation.

7293. Shared exit handling for object/array serialization.

7294. input string scan

7295. as elements

7296. * Rewind lexer. * * duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp * literal" mode with current strictness. * * curr_token line number info should be initialized for pass 2 before * generating prologue, to ensure prologue bytecode gets nice line numbers.

7297. Shared helper for Object.getOwnPropertyNames() and Object.keys(). * Magic: 0=.getOwnPropertyNames, 1=Object.keys.

7298. no need to unwind

7299. When looking for .fileName/.lineNumber, blame first * function which has a .fileName.

7300. **comment:** XXX: if attempt to push beyond allocated valstack, this double fault * handling fails miserably. We should really write the double error * directly to thr->heap->jl.value1 and avoid valstack use entirely.
label: code-design

7301. Strict functions don't get their 'caller' updated.

7302. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func]

7303. currently nop

7304. [... closure template]

7305. * A tiny random number generator. ** Currently used for Math.random(). ** http://www.woodmann.com/forum/archive/index.php/t-3100.html

7306. 'func' on stack (borrowed reference)

7307. **comment:** * "prototype" is, by default, a fresh object with the "constructor" * property. ** Note that this creates a circular reference for every function * instance (closure) which prevents refcount-based collection of * function instances. ** XXX: Try to avoid creating the default prototype object, because * many functions are not used as constructors and the default * prototype is unnecessary. Perhaps it could be created on-demand * when it is first accessed?
label: code-design

7308. -> [... val this]

7309. **comment:** code emission temporary
label: code-design

7310. * TypedArray.prototype.set() is pretty interesting to implement because: ** - The source argument may be a plain array or a typedarray. If the * source is a TypedArray, values are decoded and re-encoded into the * target (not as a plain byte copy). This may happen even when the * element byte size is the same, e.g. integer values may be re-encoded * into floats. ** - Source and target may refer to the same underlying buffer, so that * the set() operation may overlap. The specification requires that this * must work as if a copy was made before the operation. Note that this * is NOT a simple memmove() situation because the source and target * byte sizes may be different -- e.g. a 4-byte source (Int8Array) may * expand to a 16-byte target (Uint32Array) so that the target overlaps * the source both from beginning and the end (unlike in typical memmove). ** - Even if 'buf' pointers of the source and target differ, there's no * guarantee that their memory areas don't overlap. This may be the * case with external buffers. ** Even so, it is nice to optimize for the common case: ** - Source and target separate buffers or non-overlapping. ** - Source and target have a compatible type so that a plain byte copy * is possible. Note that while e.g. uint8 and int8 are compatible * (coercion one way or another doesn't change the byte representation), * e.g. int8 and uint8clamped are NOT compatible when writing int8 * values into uint8clamped typedarray (-1 would clamp to 0 for instance). ** See test-bi-typedarray-proto-set.js.

7311. (quotient-remainder (* r B) s) using a dummy subtraction loop

7312. Note: either pointer may be NULL (at entry), so don't assert

7313. String-number-buffer/object -> coerce object to primitive (apparently without hint), then try again.

7314. * Property handling ** The API exposes only the most common property handling functions. * The caller can invoke Ecmascript built-ins for full control (e.g. * defineProperty, getOwnPropertyDescriptor).

7315. Format of magic, bits: * 0...1: elem size shift (0-3) * 2...5: elem type (DUK_HBUFFEROBJECT_ELEM_xxx)

7316. **comment:** Slow gathering of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. Handling * of negative numbers is a bit non-obvious in both cases.
label: code-design

7317. Built-in 'name' is not writable by default. Function '.name' * is writable to allow user code to set a '.name' on a native * function.

7318. reuse last emitted atom for remaining 'infinite' quantifier

7319. TypeError if rval is not an object (or lightfunc which should behave * like a Function instance).

7320. * Dragon4 slow path digit generation.

7321. Rule control flags.

7322. USE_PROP_HASH_PART

7323. don't compact objects; needed during object property allocation resize

7324. r <- (remainder (* r B) s)

7325. [... error arr]

7326. String is an own (virtual) property of a lightfunc.

7327. 16: getHours

7328. 'debugger'

7329. * String manipulation

7330. fmod() return value has same sign as input (negative) so * the result here will be in the range [-2,0], 1 indicates * odd. If x is -Infinity, NaN is returned and the odd check * always concludes "not odd" which results in desired outcome.

7331. DUK_USE_DEBUGGER_INSPECT

7332. Lookup 'key' from arguments internal 'map', perform a variable write if mapped. * Used in E5 Section 10.6 algorithm for [[DefineOwnProperty]] (used by [[Put]]). * Assumes stack top contains 'put' value (which is NOT popped).

7333. * Macro for object validity check ** Assert for currently guaranteed relations between flags, for instance.

7334. **comment:** Process debug messages until we are no longer paused.
label: code-design

7335. * Entries part

7336. [in_val in_val]

7337. DUK_TOK_LBRACKET

7338. 36 bits

7339. for zero size allocations NULL is allowed

7340. -> [... obj key value]

7341. * Some change(s) need to be made. Steps 7-11.

7342. -1 = error, -2 = allowed whitespace, -3 = padding ('='), 0...63 decoded bytes

7343. duk_new() will call the constructor using duk_handle_call(). * A constructor call prevents a yield from inside the constructor, * even if the constructor is an Ecmascript function.

7344. Select a safe loop count where no output checks are * needed assuming we won't encounter escapes. Input * bound checks are not necessary as a NUL (guaranteed) * will cause a SyntaxError before we read out of bounds.

7345. end of diff run

7346. * Buffer

7347. Note: ctx.steps is intentionally not reset, it applies to the entire unanchored match

7348. **comment:** XXX: use the pointer as a seed for now: mix in time at least
label: code-design

7349. XXX: stringable emergency compaction?

7350. thr->ptr_curr_pc is set by bytecode executor early on entry

7351. Allow leading plus sign

7352. if slice not fully valid, treat as error

7353. Now two entries in the same slot, alloc list

7354. * The loop below must avoid issues with potential callstack * reallocations. A resize (and other side effects) may happen * e.g. due to finalizer/errhandler calls caused by a refzero or * mark-and-sweep. Arbitrary finalizers may run, because when * an environment record is refzero'd, it may refer to arbitrary * values which also become refzero'd. ** So, the pointer 'p' is re-looked-up below whenever a side effect * might have changed it.

7355. thr->heap->jl.value1 is already the value to throw

7356. Trailing whitespace has been eaten by duk_dec_value(), so if * we're not at end of input here, it's a SyntaxError.

7357. If user callback did not return source code, module loading * is finished (user callback initialized exports table directly).

7358. **comment:** * Emit a final RETURN. ** It would be nice to avoid emitting an unnecessary "return" opcode * if the current PC is not reachable. However, this cannot be reliably * detected; even if the previous instruction is an unconditional jump, * there may be a previous jump which jumps to current PC (which is the * case for iteration and conditional statements, for instance).

label: code-design

7359. * Because buffer values are often looped over, a number fast path * is important.

7360. starting line number

7361. $x <- x - y$, use t as temp

7362. EQUALITY EXPRESSION

7363. * Unicode support tables automatically generated during build.

7364. return thread

7365. **comment:** No catch variable, e.g. a try-finally; replace LDCONST with * NOP to avoid a bogus LDCONST.**label:** code-design

7366. 0x08583b00

7367. Signed shift, named "arithmetic" (asl) because the result * is signed, e.g. 4294967295 << 1 -> -2. Note that result * must be masked.

7368. fmax() with args -0 and +0 is not guaranteed to return * +0 as Ecmascript requires.

7369. not a wordchar

7370. XXX: a "first" flag would suffice

7371. reg count

7372. Buffer used for generated digits, values are in the range [0,B-1].

7373. assume memset with zero size is OK

7374. If ctx->offset >= ctx->length, we "shift zeroes in" * instead of croaking.

7375. Define own property without inheritance looks and such. This differs from * [[DefineOwnProperty]] because special behaviors (like Array 'length') are * not invoked by this method. The caller must be careful to invoke any such * behaviors if necessary.

7376. Parse a number, other than NaN or +/- Infinity

7377. Must be a "string object", i.e. class "String"

7378. no decref needed for a number

7379. [...] | (crud)

7380. $x <- y - z$, require $x \geq y \Rightarrow z \geq 0$, i.e. $y \geq z$

7381. 26 bits

7382. Used to support single-byte stream lookahead.

7383. Filename/line from C macros (_FILE_, _LINE_) are added as an * entry with a special format: (string, number). The number contains * the line and flags.

7384. -> [ToObject(this)]

7385. Buffer sizes for some UNIX calls. Larger than strictly necessary * to avoid Valgrind errors.

7386. \"Udeadbeef"

7387. * every(), some(), forEach(), map(), filter()

7388. Test without volatiles

7389. Lexer codepoint with additional info like offset/line number

7390. Low memory options

7391. Constants should be signed so that signed arithmetic involving them * won't cause values to be coerced accidentally to unsigned.

7392. Note: either pointer may be NULL (at entry), so don't assert.

7393. **comment:** Portability workaround is required for platforms without * unaligned access. The replacement code emulates little * endian access even on big endian architectures, which is * OK as long as it is consistent for a build.**label:** code-design

7394. tail call -> reuse current "frame"

7395. refcount is unsigned, so always true

7396. work list for objects to be finalized (by mark-and-sweep)

7397. **comment:** XXX: remove the preventcount and make yield walk the callstack? * Or perhaps just use a single flag, not a counter, faster to just * set and restore?**label:** code-design7398. **comment:** still in use with verbose messages**label:** code-design

7399. stack is unbalanced, but: [<something> buf]

7400. parse ranges until character class ends

7401. Note: buffer is dynamic so that we can 'steal' the actual * allocation later.

7402. 'delete'

7403. **comment:** unused now, not needed until Turkish/Azeri**label:** requirement

7404. No platform-specific parsing, this is not an error.

7405. mark finalizable as reachability roots

7406. **comment:** XXX: Add a proper condition. If formals list is omitted, recheck * handling for 'length' in duk_js_push_closure(); it currently relies * on _Formals being set. Removal may need to be conditional to debugging * being enabled/disabled too.**label:** code-design7407. **comment:** XXX: naming is inconsistent: ATOM_END_GROUP ends an ASSERT_START_LOOKAHEAD**label:** code-design

7408. ToUint32() coercion required

7409. 'global'

7410. not popped by side effect

7411. Add a number containing: pc, activation flags. * * PC points to next instruction, find offending PC. Note that * PC == 0 for native code.

7412. Check array density and indicate whether or not the array part should be abandoned.

7413. * Duktape.Buffer: toString(), valueOf()

7414. descriptor type specific checks

7415. DUK_USE_ERRTHROW

7416. * Longjmp and other control flow transfer for the bytecode executor. * * The longjmp handler can handle all longjmp types: error, yield, and * resume (pseudotypes are never actually thrown). * * Error policy for longjmp: should not ordinarily throw errors; if errors * occur (e.g. due to out-of-memory) they bubble outwards rather than being * handled recursively.

7417. unary minus

7418. ignore non-strings

7419. Clamp to target's end if too long. * * NOTE: there's no overflow possibility in the comparison: * both target_ustart and copy_size are >= 0 and based on * values in duk_int_t range. Adding them as duk_uint_t * values is then guaranteed not to overflow.

7420. move pivot to its final place

7421. Setting to NULL causes 'caller' to be set to * 'null' as desired.

7422. **comment:** * Global state for working around missing variadic macros**label:** code-design

7423. * Allocate and initialize a new entry, resizing the properties allocation * if necessary. Returns entry index (e_idx) or throws an error if alloc fails. * * Sets the key of the entry (increasing the key's refcount), and updates * the hash part if it exists. Caller must set value and flags, and update * the entry value refcount. A decref for the previous value is not necessary.

7424. tolerates NULL h_buf

7425. forced_reg

7426. DUK_TOK_PACKAGE

7427. B -> target register * C -> constant index of identifier name

7428. no match, next case

7429. * Interned identifier is compared against reserved words, which are * currently interned into the heap context. See genbuiltins.py. ** Note that an escape in the identifier disables recognition of * keywords; e.g. "\u0069f = 1;" is a valid statement (assigns to * identifier named "if"). This is not necessarily compliant, * see test-dec-escaped-char-in-keyword.js. ** Note: "get" and "set" are awkward. They are not officially * ReservedWords (and indeed e.g. "var set = 1;" is valid), and * must come out as DUK_TOK_IDENTIFIER. The compiler needs to * work around this a bit.

7430. NULL if tv_obj is primitive

7431. lJ value2: thread

7432. Note: use 'curr' as a temp propdesc

7433. Setter APIs detect special year numbers (0...99) and apply a +1900 * only in certain cases. The legacy getYear() getter applies -1900 * unconditionally.

7434. * substring(), substr(), slice()

7435. 30: setHours

7436. Expression, ']' terminates

7437. * Number checkers

7438. E5 Section 11.2.3, step 6.a.i

7439. -> [hobject props]

7440. XXX: shared handling for 'duk__expr_lhs'?

7441. Fix for https://github.com/svaarala/duktape/issues/155: * If 'default' is first clause (detected by pc_prevcase < 0) * we need to ensure we stay in the matching chain.

7442. eat 'in'

7443. This may trigger mark-and-sweep with arbitrary side effects, * including an attempted resize of the object we're resizing, * executing a finalizer which may add or remove properties of * the object we're resizing etc.

7444. Call with count == DUK_LEXER_WINDOW_SIZE to fill buffer initially.

7445. zero-based month

7446. * Must be very careful here, every DECREF may cause reallocation * of our valstack.

7447. expect_token

7448. periods

7449. idx of first argument on stack

7450. Add trailer if: * a) non-empty input * b) empty input and no (zero size) match found (step 11)

7451. check stack first

7452. mod_id may be NULL

7453. http://en.wikipedia.org/wiki/ANSI_escape_code

7454. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway

7455. DUK_TOK_SET

7456. * Lookup variable and update in-place if found.

7457. 'Int8Array'

7458. Last component name in resolved path

7459. Must match src/extract_chars.py, generate_match_table3().

7460. Resize target buffer for requested size. Called by the macro only when the * fast path test (= there is space) fails.

7461. Ready to make the copy. We must proceed element by element * and must avoid any side effects that might cause the buffer * validity check above to become invalid. ** Although we work through the value stack here, only plain * numbers are handled which should be side effect safe.

7462. **comment:** * Compared to a direct macro expansion this wrapper saves a few * instructions because no heap dereferencing is required.
label: code-design

7463. Force pause if we were doing "step into" in another activation.

7464. allow fmt==NULL

7465. **comment:** * Reallocate property allocation, moving properties to the new allocation. ** Includes key compaction, rehashing, and can also optionally abandoning * the array part, 'migrating' array entries into the beginning of the * new entry part. Arguments are not validated here, so e.g. new_h_size * MUST be a valid prime. ** There is no support for in-place reallocation or just compacting keys * without resizing the property allocation. This is intentional to keep * code size minimal. ** The implementation is relatively straightforward, except for the array * abandonment process. Array abandonment requires that new string keys * are interned, which may trigger GC. All keys interned so far must be * reachable for GC at all times; valstack is used for that now. ** Also, a GC triggered during this reallocation process must not interfere * with the object being resized. This is currently controlled by using * heap->mark_and_sweep_base_flags to indicate that no finalizers will be * executed (as they can affect ANY object) and no objects are compacted * (it would suffice to protect this particular object only, though). ** Note: a non-checked variant would be nice but is a bit tricky to * implement for the array abandonment process. It's easy for * everything else. ** Note: because we need to potentially resize the valstack (as part * of abandoning the array part), any tval pointers to the valstack * will become invalid after this call.
label: code-design

7466. Note: target registers a and a+1 may overlap with DUK__REGP(b) * and DUK__REGCONSTP(c). Careful here.

7467. duk_hcompiledfunction flags; quite version specific

7468. these are not necessarily 0 or 1 (may be other non-zero), that's ok

7469. +1 = finally

7470. Clamping needed if duk_int_t is 64 bits.

7471. 'id'

7472. output bufwriter

7473. -> x in [-2**31,2**31[

7474. DUK_USE_SELF_TESTS

7475. built-in strings

7476. * Process messages and send status if necessary. ** If we're paused, we'll block for new messages. If we're not * paused, we'll process anything we can peek but won't block * for more. Detach (and re-attach) handling is all localized * to duk_debug_process_messages() too. ** Debugger writes outside the message loop may cause debugger * detach1 phase to run, after which dbg_read_cb == NULL and * dbg_detaching != 0. The message loop will finish the detach * by running detach2 phase, so enter the message loop also when * detaching.

7477. * The error object has been augmented with a traceback and other * info from its creation point -- usually the current thread. * The error handler, however, is called right before throwing * and runs in the yielder's thread.

7478. * Error throwing helpers

7479. [... new_glob new_env new_glob new_glob]

7480. For lightfuncs, simply read the virtual property.

7481. 0x70-0x7f

7482. stable

7483. XXX: Could add fast path (for each transform callback) with direct byte * lookups (no shifting) and no explicit check for x < 0x80 before table * lookup.

7484. Duktape.Thread.resume()

7485. **comment:** * Note: of native Ecmascript objects, only Function instances * have a [[HasInstance]] internal property. Custom objects might * also have it, but not in current implementation. ** XXX: add a separate flag, DUK_HOBJECT_FLAG_ALLOW_INSTANCEOF?
label: code-design

7486. nothing to NULL

7487. use SameValue instead of non-strict equality

7488. a value is left on stack regardless of rc

7489. Detach is pending; can be triggered from outside the * debugger loop (e.g. Status notify write error) or by * previous message handling. Call detached callback * here, in a controlled state, to ensure a possible * reattach inside the detached_cb is handled correctly. ** Recheck for detach in a while loop: an immediate * reattach involves a call to duk_debugger_attach() * which writes a debugger handshake line immediately * inside the API call. If the transport write fails * for that handshake, we can immediately end up in a * "transport broken, detaching" case several times here. * Loop back until we're either cleanly attached or * fully detached. ** NOTE: Reset dbg_processing = 1 forcibly, in case we * re-attached; duk_debugger_attach() sets dbg_processing * to 0 at the moment.

7490. Run the finalizer, duk_hobject_run_finalizer() sets FINALIZED. * Next mark-and-sweep will collect the object unless it has * become reachable (i.e. rescued). FINALIZED prevents the * finalizer from being executed again before that.

7491. Note: this must match ToObject() behavior
7492. 'new' MemberExpression
7493. The code below is incorrect if .toString() or .valueOf() have * have been overridden. The correct approach would be to look up * the method(s) and if they resolve to the built-in function we * can safely bypass it and look up the internal value directly. * Unimplemented for now, abort fast path for boxed values.
7494. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx [21 bits]
7495. **comment:** XXX: avoid this at least when enum_target is an Array, it has an * array part, and no ancestor properties were included? Not worth * it for JSON, but maybe worth it for forEach().
label: code-design
7496. Report it to the debug client
7497. no compact
7498. restricted to regs
7499. dummy so sp won't get updated
7500. * Replace global object.
7501. require a (mutable) temporary as a result (or a const if allowed)
7502. contains value
7503. %6s' and NULL is not portable, so special case * it for debug printing.
7504. Compiler SyntaxErrors (and other errors) come first, and are * blamed by default (not flagged "noblame").
7505. Relevant PC is just before current one because PC is * post-incremented. This should match what error augment * code does.
7506. Parse formals.
7507. [... key arg1 ... argN func]
7508. exit bytecode execution with return value
7509. * Status message and helpers
7510. Note: 'func' is popped from valstack here, but it is * already reachable from the activation.
7511. points to LABEL; pc+1 = jump site for break; pc+2 = jump site for continue
7512. DUK_SELFTEST_H_INCLUDED
7513. Duktape object
7514. 0xb0-0xbf
7515. * 'key_obj' tracks keys encountered so far by associating an * integer with flags with already encountered keys. The checks * below implement E5 Section 11.1.5, step 4 for production: * * PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment
7516. [... res]
7517. * UTF-8 decoder which accepts longer than standard byte sequences. * This allows full 32-bit code points to be used.
7518. accessor with defined getter
7519. **comment:** XXX: optimize for packed duk_tval directly?
label: code-design
7520. * Augment created errors upon creation (not when they are thrown or * rethrown). __FILE__ and __LINE__ are not desirable here; the call * stack reflects the caller which is correct.
7521. allow source elements
7522. one token before
7523. 'filename'
7524. Neutered. We could go into the switch-case safely with * buf == NULL because check_length == 0. To avoid scanbuild * warnings, fail directly instead.
7525. -> x in J-2**32, 2**32[
7526. no need to incref
7527. Ecmascript function
7528. embed: void ptr
7529. Lightfuncs are always native functions and have "newenv".
7530. This upper value has been experimentally determined; debug build will check * bigint size with assertions.
7531. **comment:** XXX: cover this with the generic >1 case?
label: code-design
7532. * Variant 1 or 3
7533. 'enumerable'
7534. **comment:** * There are two [[Construct]] operations in the specification: * * - E5 Section 13.2.2: for Function objects * - E5 Section 15.3.4.5.2: for "bound" Function objects * * The chain of bound functions is resolved in Section 15.3.4.5.2, * with arguments "piling up" until the [[Construct]] internal * method is called on the final, actual Function object. Note * that the "prototype" property is looked up *only* from the * final object, *before* calling the constructor. * * Currently we follow the bound function chain here to get the * "prototype" property value from the final, non-bound function. * However, we let duk_handle_call() handle the argument "piling" * when the constructor is called. The bound function chain is * thus now processed twice. * * When constructing new Array instances, an unnecessary object is * created and discarded now: the standard [[Construct]] creates an * object, and calls the Array constructor. The Array constructor * returns an Array instance, which is used as the result value for * the "new" operation; the object created before the Array constructor * call is discarded. * * This would be easy to fix, e.g. by knowing that the Array constructor * will always create a replacement object and skip creating the fallback * object in that case. * * Note: functions called via "new" need to know they are called as a * constructor. For instance, built-in constructors behave differently * depending on how they are called.
label: code-design
7535. zero size not an issue: pointers are valid
7536. fromPresent = true
7537. The counter value is one less than the init value: init value should * indicate how many instructions are executed before interrupt. To * execute 1 instruction (after interrupt handler return), counter must * be 0.
7538. assumes that allocated pointers and alloc funcs are valid * if res exists
7539. DUK_TOK_EQ
7540. * >>> struct.pack('>d', 102030405060).encode('hex') * '4237c17c6dc40000'
7541. XXX: add 'language' argument when locale/language sensitive rule * support added.
7542. Currently there are no deletable virtual properties, but * with force_flag we might attempt to delete one.
7543. **comment:** XXX: This helper is a bit awkward because the handling for the different iteration * callers is quite different. This now compiles to a bit less than 500 bytes, so with * 5 callers the net result is about 100 bytes / caller.
label: code-design
7544. Infinity
7545. **comment:** XXX: unoptimal use of temps, resetting
label: code-design
7546. -> [... regexp string] -> [... res_obj]
7547. token type (with reserved words as DUK_TOK_IDENTIFIER)
7548. Note: assumes that these string indexes are 8-bit, genstrings.py must ensure that
7549. start variant 3/4 left-hand-side code (L1 in doc/compiler.rst example)
7550. # argument registers target function wants (< 0 => "as is")
7551. 'magic' constants for Murmurhash2
7552. If something is thrown with the debugger attached and nobody will * catch it, execution is paused before the longjmp, turning over * control to the debug client. This allows local state to be examined * before the stack is unwound. Errors are not intercepted when debug * message loop is active (e.g. for Eval).
7553. Object built-in methods
7554. Skip fully.
7555. switch (ch2)
7556. type

7557. * Allocate memory with garbage collection
7558. * Main switch for statement / source element type.
7559. Shared entry handling for object/array serialization.
7560. overwrite sep
7561. [message error message]
7562. [... parent stash stash] -> [... parent stash]
7563. Non-verbose errors for low memory targets: no file, line, or message.
7564. DUK_BUFOBJ_ARRAYBUFFER
7565. **comment:** * Recursion limit check. ** Note: there is no need for an "ignore recursion limit" flag * for duk_handle_safe_call now.
 label: code-design
7566. Exponent handling: if exponent format is used, record exponent value and * fake k such that one leading digit is generated (e.g. digits=123 -> "1.23"). ** toFixed() prevents exponent use; otherwise apply a set of criteria to * match the other API calls (toString(), toPrecision, etc).
7567. **comment:** We want the argument expression value to go to "next temp" * without additional moves. That should almost always be the * case, but we double check after expression parsing. ** This is not the cleanest possible approach.
 label: code-design
7568. seconds, doesn't fit into 16 bits
7569. unreferenced w/o tracebacks
7570. We want to avoid making a copy to process set() but that's * not always possible: the source and the target may overlap * and because element sizes are different, the overlap cannot * always be handled with a memmove() or choosing the copy * direction in a certain way. For example, if source type is * uint8 and target type is uint32, the target area may exceed * the source area from both ends! ** Note that because external buffers may point to the same * memory areas, we must ultimately make this check using * pointers. ** NOTE: careful with side effects: any side effect may cause * a buffer resize (or external buffer pointer/length update)!
7571. **comment:** NEXTENUM needs a jump slot right after the main instruction. * When the JUMP is taken, output spilling is not needed so this * workaround is possible. The jump slot PC is exceptionally * plumbed through comp_ctx to minimize call sites.
 label: code-design
7572. must be updated to work properly (e.g. creation of 'arguments')
7573. Unescaped '/' ANYWHERE in the regexp (in disjunction, * inside a character class, ...) => same escape works.
7574. 1:1 or special conversions, but not locale/context specific: script generated rules
7575. roughly 0.5 kB
7576. * Value stack resize and stack top adjustment helper. ** XXX: This should all be merged to duk_valstack_resize_raw().
7577. **comment:** * Macro support functions for reading/writing raw data. ** These are done using memcpy to ensure they're valid even for unaligned * reads/writes on platforms where alignment counts. On x86 at least gcc * is able to compile these into a bswap+mov. "Always inline" is used to * ensure these macros compile to minimal code. ** Not really bufwriter related, but currently used together.
 label: code-design
7578. If tracebacks are enabled, the '_Tracedata' property is the only * thing we need: 'fileName' and 'lineNumber' are virtual properties * which use '_Tracedata'.
7579. **comment:** XXX: could duk_is_undefined() provide defaulting undefined to 'len' * (the upper limit)?
 label: code-design
7580. DUK_TOK_WITH
7581. 0xa0-0xaf
7582. * Table for base-64 decoding
7583. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor).
 label: code-design
7584. [... buf]
7585. * Regexp compilation. ** See doc/regexp.rst for a discussion of the compilation approach and * current limitations. ** Regexp bytecode assumes jumps can be expressed with signed 32-bit * integers. Consequently the bytecode size must not exceed 0x7fffffffL. * The implementation casts duk_size_t (buffer size) to duk_(u)int32_t * in many places. Although this could be changed, the bytecode format * limit would still prevent regexps exceeding the signed 32-bit limit * from working. ** XXX: The implementation does not prevent bytecode from exceeding the * maximum supported size. This could be done by limiting the maximum * input string size (assuming an upper bound can be computed for number * of bytecode bytes emitted per input byte) or checking buffer maximum * size when emitting bytecode (slower).
7586. Production allows 'DecimalDigits', including leading zeroes
7587. Note: must coerce to a (writable) temp register, so that e.g. "!x" where x * is a reg-mapped variable works correctly (does not mutate the variable register).
7588. [] -> []
7589. key
7590. * Exception for Duktape internal throws when C++ exceptions are used * for long control transfers. ** Doesn't inherit from any exception base class to minimize the chance * that user code would accidentally catch this exception.
7591. -> [array totalLength buf]
7592. Maximum prototype traversal depth. Sanity limit which handles e.g. * prototype loops (even complex ones like 1->2->3->4->2->3->4->2->3->4).
7593. is valid size
7594. [O Properties obj]
7595. extra coercion of strings is OK
7596. **comment:** XXX: increase ctx->expr_tokens here for every consumed token * (this would be a nice statistic)?
 label: code-design
7597. FAIL
7598. Two digit octal escape, digits validated. ** The if-condition is a bit tricky. We could catch e.g. '\039' in the three-digit escape and fail it there (by * validating the digits), but we want to avoid extra * additional validation code.
7599. For n == 0, Node.js ignores totalLength argument and * returns a zero length buffer.
7600. * Push result object and init its flags
7601. number of continuation (non-initial) bytes in [0x80,0xbf]
7602. External buffer with 'curr_alloc' managed by user code and pointing to an * arbitrary address. When heap pointer compression is not used, this struct * has the same layout as duk_hbuffer_dynamic.
7603. **comment:** Note: we can reuse 'desc' here
 label: code-design
7604. empty statement with an explicit semicolon
7605. * Constructor
7606. * Array exotic behaviors can be implemented at this point. The local variables * are essentially a 'value copy' of the input descriptor (Desc), which is modified * by the Array [[DefineOwnProperty]] (E5 Section 15.4.5.1).
7607. * Expression parsing. ** Upon entry to 'expr' and its variants, 'curr_tok' is assumed to be the * first token of the expression. Upon exit, 'curr_tok' will be the first * token not part of the expression (e.g. semicolon terminating an expression * statement).
7608. The index range space is conceptually the array part followed by the * entry part. Unlike normal enumeration all slots are exposed here as * is and return 'unused' if the slots are not in active use. In particular * the array part is included for the full a_size regardless of what the * array .length is.
7609. * toString()
7610. shuffle registers if large number of regs/consts
7611. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8
7612. **comment:** XXX: add support for variables to not be register bound always, to * handle cases with a very large number of variables?
 label: code-design
7613. entry_top + 2
7614. [result]

7615. * Determine call type, then finalize activation, shift to * new value stack bottom, and call the target.
7616. [source]
7617. * Arguments object creation. ** Creating arguments objects involves many small details, see E5 Section * 10.6 for the specific requirements. Much of the arguments object exotic * behavior is implemented in duk_hobject_props.c, and is enabled by the * object flag DUK_HOBJECT_FLAG_EXOTIC_ARGUMENTS.
7618. **comment:** XXX: space in valstack? see discussion in duk_handle_call_xxx().
 label: code-design
7619. combines steps 2 and 5; -len ensures max() not needed for step 5
7620. for register-bound and declarative env identifiers
7621. bound functions are never in act 'func'
7622. Lightfuncs cannot be bound.
7623. **comment:** The resulting buffer object gets the same class and prototype as * the buffer in 'this', e.g. if the input is a Node.js Buffer the * result is a Node.js Buffer; if the input is a Float32Array, the * result is a Float32Array. ** For the class number this seems correct. The internal prototype * is not so clear: if 'this' is a bufferobject with a non-standard * prototype object, that value gets copied over into the result * (instead of using the standard prototype for that object type).
 label: code-design
7624. Don't allow mixed padding and actual chars.
7625. reject RegExp literal on next advance() call; needed for handling IdentifierName productions
7626. DUK_TOK_CONTINUE
7627. unbalanced stack on purpose
7628. arguments MUST have an initialized lexical environment reference
7629. **comment:** XXX: sanitize to printable (and maybe ASCII)
 label: code-design
7630. tags
7631. regexp compiler should catch these
7632. **comment:** Current 'this' binding is the value just below idx_bottom. * Previously, 'this' binding was handled with an index to the * (calling) valstack. This works for everything except tail * calls, which must not "cumulate" valstack temps.
 label: code-design
7633. * ctx->prev_token token to process with duk__expr_nud() * ctx->curr_token updated by caller ** Note: the token in the switch below has already been eaten.
7634. -> [... voidp true]
7635. The E5.1 specification does not seem to allow IdentifierPart characters * to be used as identity escapes. Unfortunately this includes '\$', which * cannot be escaped as '\$'; it needs to be escaped e.g. as '\u0024'. * Many other implementations (including V8 and Rhino, for instance) do * accept '\$' as a valid identity escape, which is quite pragmatic. * See: test-regexp-identity-escape-dollar.js.
7636. mp <- b^(e+1)
7637. * Debugger handling for executor restart ** Check for breakpoints, stepping, etc, and figure out if we should execute * in checked or normal mode. Note that we can't do this when an activation * is created, because breakpoint status (and stepping status) may change * later, so we must recheck every time we're executing an activation. * This primitive should be side effect free to avoid changes during check.
7638. at most sizeof(buf) - 1
7639. val
7640. Old solution: don't iterate, incorrect
7641. * API calls related to general value stack manipulation: resizing the value * stack, pushing and popping values, type checking and reading values, * coercing values, etc. * * Also contains internal functions (such as duk_get_tval()), defined * in duk_api_internal.h, with semantics similar to the public API.
7642. No BC shuffling now.
7643. syntactically left-associative but parsed as right-associative
7644. * Local result type for duk__get_identifier_reference() lookup.
7645. **comment:** XXX: would be nice to enumerate an object at specified index
 label: code-design
7646. ''
7647. * Heap stringtable handling, string interning.
7648. By default the global object is read-only which is often much * more of an issue than having read-only built-in objects (like * RegExp, Date, etc). Use a RAM-based copy of the global object * and the global environment object for convenience.
7649. Creation time error augmentation
7650. The casts through duk_intr_pt is to avoid the following GCC warning: * * warning: cast from pointer to integer of different size [-Wpointer-to-int-cast] * * This still generates a /Wp64 warning on VS2010 when compiling for x86.
7651. **comment:** * Note: currently register bindings must be fixed for the entire * function. So, even though the catch variable is in a register * we know, we must use an explicit environment record and slow path * accesses to read/write the catch binding to make closures created * within the catch clause work correctly. This restriction should * be fixable (at least in common cases) later. ** See: test-bug-catch-binding-2.js. ** XXX: improve to get fast path access to most catch clauses.
 label: code-design
7652. check that byte has the form 10xx xxxx
7653. * Memory constants
7654. return 'res' (of right part) as our result
7655. -> [func funcname env]
7656. this variant is used when an assert would generate a compile warning by * being always true (e.g. ≥ 0 comparison for an unsigned value
7657. Stringcache is used for speeding up char-offset-to-byte-offset * translations for non-ASCII strings.
7658. **comment:** Note: strictness is *not* inherited from the current Duktape/C. * This would be confusing because the current strictness state * depends on whether we're running inside a Duktape/C activation * (= strict mode) or outside of any activation (= non-strict mode). * See tests/api/test-strictness.c for more discussion.
 label: code-design
7659. * Finalizer check. ** Note: running a finalizer may have arbitrary side effects, e.g. * queue more objects on refzero_list (tail), or even trigger a * mark-and-sweep. * * Note: quick reject check should match vast majority of * objects and must be safe (not throw any errors, ever).
7660. Get Proxy target object. If the argument is not a Proxy, return it as is. * If a Proxy is revoked, an error is thrown.
7661. * Assertion helpers.
7662. is_setget
7663. * Compute new alloc size and alloc new area. ** The new area is allocated as a dynamic buffer and placed into the * valstack for reachability. The actual buffer is then detached at * the end. ** Note: heap_mark_and_sweep_base_flags are altered here to ensure * no-one touches this object while we're resizing and rehashing it. * The flags must be reset on every exit path after it. Finalizers * and compaction is prevented currently for all objects while it * would be enough to restrict it only for the current object.
7664. replacer function
7665. [...]
7666. **comment:** remove the string (mark DELETED), could also call * duk_heap_string_remove() but that would be slow and * pointless because we already know the slot.
 label: code-design
7667. +(function(){}) -> NaN
7668. **comment:** * Date built-ins ** Unlike most built-ins, Date has some platform dependencies for getting * UTC time, converting between UTC and local time, and parsing and * formatting time values. These are all abstracted behind DUK_USE_xxx * config options. There are built-in platform specific providers for * POSIX and Windows, but external providers can also be used. ** See doc/datetime.rst. *
 label: code-design
7669. Reject attempt to change a read-only object.

7670. final result is already in 'res'
7671. 21: getUTCSeconds
7672. Eval is just a wrapper now.
7673. With ROM objects "needs refcount update" is true when the value is * heap allocated and is not a ROM object.
7674. 22: getMilliseconds
7675. binary arithmetic using extraops; DUK_EXTRAOP_INSTOF etc
7676. combined algorithm matching E5 Sections 9.5 and 9.6
7677. Evaluates to (duk_uint8_t *) pointing to start of area.
7678. module.id = resolved_id
7679. Property access expressions ('a[b]') are critical to correct * LHS evaluation ordering, see test-dev-assign-eval-order*.js. * We must make sure that the LHS target slot (base object and * key) don't change during RHS evaluation. The only concrete * problem is a register reference to a variable-bound register * (i.e., non-temp).
Require temp regs for both key and base. * * Don't allow a constant for the object (even for a number * etc), as it goes into the 'A' field of the opcode.
7680. Maximum number of characters in formatted value.
7681. Read fully.
7682. * Prototypes
7683. For indirect allocs.
7684. always allow 'in', coerce to 'tr' just in case
7685. DUK_USE_DEBUGGER_SUPPORT && (DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT)
7686. Pointer object internal value is immutable
7687. Leave 'value' on stack
7688. prev char
7689. * [[DefaultValue]] (E5 Section 8.12.8) * * ==> implemented in the API.
7690. 'extends'
7691. result: hash_prime(floor(1.2 * e_size))
7692. bp is assumed to be even
7693. stridx_logfunc[] must be static to allow initializer with old compilers like BCC
7694. Positive index can be higher than valstack top but must * not go above allocated stack (equality is OK).
7695. out_desc is left untouched (possibly garbage), caller must use return * value to determine whether out_desc can be looked up
7696. If buffer has been exhausted, truncate bitstream
7697. Can be called multiple times with no harm.
7698. Helper for component setter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), modify one or more components as specified, recompute * the time value, set it as the internal value. Finally, push the * new time value as a return value to the value stack and return 1 * (caller can then tail call us).
7699. error
7700. 1:1 conversion
7701. Encode into a double constant (53 bits can encode 6*8 = 48 bits + 3-bit length
7702. not LS/PS
7703. invalid codepoint encoding or codepoint
7704. **comment:** packed advance/token number macro used by multiple functions
label: code-design
7705. The JO(value) operation: encode object. * * Stack policy: [object] -> [object].
7706. DUK_TOK_BAND_EQ
7707. remaining
7708. **comment:** * Duktape built-ins * * Size optimization note: it might seem that vararg multipurpose functions * like fin(), enc(), and dec() are not very size optimal, but using a single * user-visible Ecmascript function saves a lot of run-time footprint; each * Function instance takes >100 bytes. Using a shared native helper and a * 'magic' value won't save much if there are multiple Function instances * anyway.
label: code-design
7709. Value is required to be a number in the fast path so there * are no side effects in write coercion.
7710. Faster but value stack overruns are memory unsafe.
7711. register for writing value of 'non-empty' statements (global or eval code), -1 is marker
7712. * Possible pending array length update, which must only be done * if the actual entry write succeeded.
7713. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
7714. If XUTF-8 decoding fails, treat the offending byte as a codepoint directly * and go forward one byte. This is of course very lossy, but allows some kind * of output to be produced even for internal strings which don't conform to * XUTF-8. All standard Ecmascript strings are always CESU-8, so this behavior * does not violate the Ecmascript specification. The behavior is applied to * all modes, including Ecmascript standard JSON. Because the current XUTF-8 * decoding is not very strict, this behavior only really affects initial bytes * and truncated codepoints. * * Another alternative would be to scan forwards to start of next codepoint * (or end of input) and emit just one replacement codepoint.
7715. For initial entry use default value; zero forces an * interrupt before executing the first instruction.
7716. already in correct reg
7717. the next 'if' is guaranteed to match after swap
7718. Encoding state. Heap object references are all borrowed.
7719. -> [... repl_value]
7720. no prototype
7721. Note: 'q' is top-1
7722. **comment:** * Abandon array part because all properties must become non-configurable. * Note that this is now done regardless of whether this is always the case * (skips check, but performance problem if caller would do this many times * for the same object; not likely).
label: code-design
7723. **comment:** XXX: incref by count (2) directly
label: code-design
7724. ')'
7725. source is a function expression (used for Function constructor)
7726. * Variant 2
7727. Stored in duk_heapdr h_extra16.
7728. Note: intentionally signed.
7729. 'Date'
7730. input linenumber at input_offset (not window[0]), init to 1
7731. **comment:** * Missing bytes snip base64 example * 0 4 XXXX * 1 3 XXX= * 2 2 XX==
label: code-design
7732. * The remaining matches are emitted as sequence of SPLITs and atom * copies; the SPLITs skip the remaining copies and match the sequel. * This sequence needs to be emitted starting from the last copy * because the SPLITs are variable length due to the variable length * skip offset. This causes a lot of memory copying now. * * Example structure (greedy, match maximum # atoms): * * SPLIT1 LSEQ * (atom) * SPLIT1 LSEQ ; <- the byte length of this instruction is needed * (atom) ; to encode the above SPLIT1 correctly * ... * LSEQ:
7733. idx_value
7734. [... this_new | arg1 ... argN]
7735. -----XX XXXX----
7736. * Get old and new length
7737. **comment:** Hot variables for interpretation. Critical for performance, * but must add sparingly to minimize register shuffling.

label: code-design

7738. 38: getYear
7739. initjs data is NUL terminated
7740. this.raw(buffer)
7741. return h_bufres
7742. Here we rely on duk_hstring instances always being zero * terminated even if the actual string is not.
7743. Must coerce key: if key is an object, it may coerce to e.g. 'length'.
7744. -> [... regexp_instance]
7745. Final result is at stack top.
7746. Strict equality is NOT enough, because we cannot use the same * constant for e.g. +0 and -0.
7747. right exists, [[Put]] regardless whether or not left exists
7748. * Return target object.
7749. reuse 'res' as 'left'
7750. st_used remains the same, DELETED is counted as used
7751. is_join
7752. binding power must be high enough to NOT allow comma expressions directly
7753. Note: these variables must reside in 'curr_func' instead of the global * context: when parsing function expressions, expression parsing is nested.
7754. Abandon array part, moving array entries into entries part. * This requires a props resize, which is a heavy operation. * We also compact the entries part while we're at it, although * this is not strictly required.
7755. _Formals
7756. base of known return values
7757. TimeClip(), which also handles Infinity -> NaN conversion
7758. [... obj]
7759. Encoding endianness must match target memory layout, * build scripts and genbuiltins.py must ensure this.
7760. flags used for property attributes in duk_propdesc and packed flags
7761. directly uses slow accesses
7762. * Finalizer torture. Do one fake finalizer call which causes side effects * similar to one or more finalizers on actual objects.
7763. add AC*2^32
7764. result: updated refcount
7765. Entry level info.
7766. catch depth from current func
7767. indexOf: clamp fromIndex to [-len, len] * (if fromIndex == len, for-loop terminates directly) * * lastIndexOf: clamp fromIndex to [-len - 1, len - 1] * (if clamped to -len-1 -> fromIndex becomes -1, terminates for-loop directly)
7768. decoding
7769. inline initializer for coercers[] is not allowed by old compilers like BCC
7770. * Create normalized 'source' property (E5 Section 15.10.3).
7771. variable access
7772. Double casting for pointer to avoid gcc warning (cast from pointer to integer of different size)
7773. **comment:** XXX: refactor out?
label: code-design
7774. same for both error and each subclass like TypeError
7775. [... res_obj re_obj input bc saved_buf]
7776. B -> register for writing enumerator object * C -> value to be enumerated (register)
7777. entire string is whitespace
7778. 'multiline'
7779. '+'
7780. * Parse an identifier and then check whether it is: * - reserved word (keyword or other reserved word) * - "null" (NullLiteral) * - "true" (BooleanLiteral) * - "false" (BooleanLiteral) * - anything else => identifier * * This does not follow the E5 productions cleanly, but is * useful and compact. * * Note that identifiers may contain Unicode escapes, * see E5 Sections 6 and 7.6. They must be decoded first, * and the result checked against allowed characters. * The above if-clause accepts an identifier start and an * '\ character -- no other token can begin with a '. * * Note that "get" and "set" are not reserved words in E5 * specification so they are recognized as plain identifiers * (the tokens DUK_TOK_GET and DUK_TOK_SET are actually not * used now). The compiler needs to work around this. * * Strictly speaking, following Ecmascript longest match * specification, an invalid escape for the first character * should cause a syntax error. However, an invalid escape * for IdentifierParts should just terminate the identifier * early (longest match), and let the next tokenization * fail. For instance Rhino croaks with 'foo\z' when * parsing the identifier. This has little practical impact.
7781. 'lineNumber'
7782. refer to callstack entries below current
7783. statement parsing
7784. **comment:** Pick a destination register. If either base value or key * happens to be a temp value, reuse it as the destination. * * XXX: The temp must be a "mutable" one, i.e. such that no * other expression is using it anymore. Here this should be * the case because the value of a property access expression * is neither the base nor the key, but the lookup result.
label: code-design
7785. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()!
7786. ignore retval -> [key]
7787. %ld
7788. * Octal escape or zero escape: * \0 (lookahead not DecimalDigit) * \1 ... \7 (lookahead not DecimalDigit) * \ZeroToThree OctalDigit (lookahead not DecimalDigit)
* \FourToSeven OctalDigit (no lookahead restrictions) * \ZeroToThree OctalDigit (no lookahead restrictions) * * Zero escape is part of the standard syntax. Octal escapes are * defined in E5 Section B.1.2, and are only allowed in non-strict mode. * Any other productions starting with a decimal digit are invalid.
7789. **comment:** XXX: for Object.keys() the [[OwnPropertyKeys]] result (trap result) * should be filtered so that only enumerable keys remain. Enumerability * should be checked with [[GetOwnProperty]] on the original object * (i.e., the proxy in this case). If the proxy has a getOwnPropertyDescriptor * trap, it should be triggered for every property. If the proxy doesn't have * the trap, enumerability should be checked against the target object instead. * We don't do any of this now, so Object.keys() and Object.getOwnPropertyNames() * return the same result now for proxy traps. We still do clean up the trap * result, so that Object.keys() and Object.getOwnPropertyNames() will return a * clean array of strings without gaps.
label: code-design
7790. syntax error
7791. don't push value
7792. 'buffer'
7793. inherit initial strictness from parent
7794. module
7795. Without tracebacks the concrete .fileName and .lineNumber need * to be added directly.
7796. not emergency
7797. * Raw memory calls: relative to heap, but no GC interaction
7798. Lookup 'key' from arguments internal 'map', perform a variable lookup * if mapped, and leave the result on top of stack (and return non-zero). * Used in E5 Section 10.6 algorithms [[Get]] and [[GetProperty]].
7799. Evaluate RHS only when LHS is safe.
7800. debugger detaching; used to avoid calling detach handler recursively
7801. no catchers
7802. * Variant 4
7803. 'package'

7804. nothing now
7805. * Reset trigger counter
7806. * Preliminaries
7807. count is already incremented, take into account
7808. **comment:** * Setters. * * Setters are a bit more complicated than getters. Component setters * break down the current time value into its (normalized) component * parts, replace one or more components with -unnormalized- new values, * and the components are then converted back into a time value. As an * example of using unnormalized values: * * var d = new Date(1234567890); * * is equivalent to: * * var d = new Date(0); * d.setUTCSeconds(1234567890); * * A shared native helper to provide almost all setters. Magic value * contains a set of flags and also packs the "maxnargs" argument. The * helper provides: * * setMilliseconds() * setUTCSeconds() * setUTCMinutes() * setUTCHours() * setDate() * setUTCDate() * setMonth() * setUTCMonth() * setFullYear() * setUTCFullYear() * setYear() * * Notes: * * - Date.prototype.setYear() (Section B addition): special year check * is omitted. NaN / Infinity will just flow through and ultimately * result in a NaN internal time value. * * - Date.prototype.setYear() does not have optional arguments for * setting month and day-in-month (like setFullYear()), but we indicate * 'maxnargs' to be 3 to get the year written to the correct component * index in duk_set_part_helper(). The function has nargs == 1, so only * the year will be set regardless of actual argument count.
label: code-design

7809. inline string concatenation
7810. DUK_USE_JX && DUK_USE_JC
7811. **comment:** XXX: ideally this would be just one flag (maybe a derived one) so * that a single bit test is sufficient to check the condition.
label: test

7812. [key] (coerced)
7813. **comment:** XXX: no optimized variants yet
label: requirement

7814. Encode a double (checked by caller) from stack top. Stack top may be * replaced by serialized string but is not popped (caller does that).
7815. This function is only called for objects with array exotic behavior. * The [[DefineOwnProperty]] algorithm for arrays requires that * 'length' can never have a value outside the unsigned 32-bit range, * attempt to write such a value is a RangeError. Here we can thus * assert for this. When Duktape internals go around the official * property write interface (doesn't happen often) this assumption is * easy to accidentally break, so such code must be written carefully. * See test-bi-array-push-maxlen.js.

7816. **comment:** DUK_ADVANCECHARS(lex_ctx, 2) would be correct here, but it unnecessary
label: code-design

7817. 'base64'
7818. [sep ToObject(this) len sep]
7819. **comment:** * Various sanity checks for typing
label: code-design

7820. Check whether statement list ends.
7821. invalidates tv1, tv2
7822. Note: MUST NOT wipe_and_return here, as heap->lj must remain intact
7823. -> sets 'f' and 'e'
7824. Check for maximum string length
7825. * Parse an individual source element (top level statement) or a statement. * * Handles labeled statements automatically (peeling away labels before * parsing an expression that follows the label(s)). * * Upon entry, 'curr_tok' contains the first token of the statement (parsed * in "allow regexp literal" mode). Upon exit, 'curr_tok' contains the first * token following the statement (if the statement has a terminator, this is * the token after the terminator).
7826. **comment:** loop_stack_index could be perhaps be replaced by 'depth', but it's nice * to not couple these two mechanisms unnecessarily.
label: code-design

7827. * Fast path for defining array indexed values without interning the key. * This is used by e.g. code for Array prototype and traceback creation so * must avoid interning.
7828. accessor
7829. set to 1 if any match exists (needed for empty input special case)
7830. this should never be possible, because the switch-case is * comprehensive
7831. avoid double coercion
7832. 'DataView'
7833. * Sweep stringtable
7834. [A B C D E F G H] rel_index = 2, del_count 3, item count 1 * -> [A B F G H] (conceptual intermediate step) * -> [A B . F G H] (placeholder marked) * [A B C F G H] (actual result at this point, C will be replaced)
7835. DUK_TOK_IN
7836. jump to false
7837. **comment:** XXX: resolve macro definition issue or call through a helper function?
label: code-design

7838. **comment:** Sanity workaround for handling functions with a large number of * constants at least somewhat reasonably. Otherwise checking whether * we already have the constant would grow very slow (as it is O(N^2)).
label: code-design

7839. heapobj recursion depth when deep printing is selected
7840. basic types
7841. temp reset is not necessary after duk_parse_stmt(), which already does it
7842. mark-and-sweep: reachable
7843. Error object is augmented at its creation here.
7844. DUK_USE_COMMONJS_MODULES
7845. at least effective 'this'
7846. * Default allocation functions. * * Assumes behavior such as malloc allowing zero size, yielding * a NULL or a unique pointer which is a no-op for free.
7847. DUK_USE_LIGHTFUNC_BUILTINS
7848. some derived types
7849. functions always have a NEWENV flag, i.e. they get a * new variable declaration environment, so only lex_env * matters here.
7850. x == +Infinity
7851. temp_start+0 = key, temp_start+1 = closure
7852. E5.1 Section 15.1.3.2: empty
7853. DUK_TOK_FINALLY
7854. **comment:** Static call style.
label: code-design

7855. 'deleteProperty'
7856. Compiling state of one function, eventually converted to duk_hcompiledfunction
7857. to body
7858. Coerce top into Object.prototype.toString() output.
7859. XXX: this is quite clunky. Add Unicode helpers to scan backwards and * forwards with a callback to process codepoints?
7860. actual update happens once write has been completed without * error below.
7861. * 'func' is now a non-bound object which supports [[HasInstance]] * (which here just means DUK_HOBJECT_FLAG_CALLABLE). Move on * to execute E5 Section 15.3.5.3.

7862. Target object must be checked for a conflicting * non-configurable property.
7863. property exists, either 'desc' is empty, or all values * match (SameValue)
7864. [func thisArg argArray]
7865. p overshoots

7866. prefer direct

7867. **comment:** XXX: what about statements which leave a half-cooked value in 'res' * but have no stmt value? Any such statements?
label: code-design

7868. Must restart in all cases because we NULLed thr->ptr_curr_pc.

7869. MAXVAL is inclusive

7870. "release" preallocated temps since we won't need them

7871. temp reg value for start of loop

7872. checked by Duktape.Thread.resume()

7873. * Union aliasing, see misc/clang_aliasing.c.

7874. [offset value littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)

7875. * Replacements for missing platform functions. * * Unlike the originals, fpclassify() and signbit() replacements don't * work on any floating point types, only doubles. The C typing here * mimics the standard prototypes.

7876. eat closing quote

7877. setup many variables in nc_ctx

7878. **comment:** XXX: fast int-to-double
label: code-design

7879. Native function pointer may be different from a void pointer, * and we serialize it from memory directly now (no byte swapping etc).

7880. **comment:** XXX: shorter version for 12-byte representation?
label: code-design

7881. a fresh require() with require.id = resolved target module id

7882. Constructor

7883. elision - flush

7884. temp reg for key literal

7885. Unlike break/continue, throw statement does not allow an empty value.

7886. DUK_TOK_VAR

7887. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object).
Return value is NULL if value is neither an object nor a * lightfunc.

7888. char offset in [0, h_input->clen] (both ends inclusive), checked before entry

7889. NOTE: fmod(x) result sign is same as sign of x, which * differs from what Javascript wants (see Section 9.6).

7890. [... bytecode escaped_source]

7891. [obj key desc value get set curr_value]

7892. >>

7893. msec

7894. unary plus of a number is identity

7895. equals and strict equals

7896. * Loose equality, strict equality, and SameValue (E5 Sections 11.9.1, 11.9.4, * 9.12). These have much in common so they can share some helpers. * * Future work notes: * * - Current implementation (and spec definition) has recursion; this should * be fixed if possible. * * - String-to-number coercion should be possible without going through the * value stack (and be more compact) if a shared helper is invoked.

7897. The debugger protocol doesn't support a plain integer encoding for * the full 32-bit unsigned range (only 32-bit signed). For now, * unsigned 32-bit values simply written as signed ones. This is not * a concrete issue except for 32-bit heaphdr fields. Proper solutions * would be to (a) write such integers as IEEE doubles or (b) add an * unsigned 32-bit dvalue.

7898. **comment:** * XXX: attempt to get the call result to "next temp" whenever * possible to avoid unnecessary register shuffles. * * XXX: CSPROP (and CSREG) can overwrite the call target register, and save one temp, * if the call target is a temporary register and at the top of the temp reg "stack".
label: code-design

7899. tv points to element just below prev top

7900. * Duktape/C function magic

7901. * Identifier handling

7902. error gets its 'name' from the prototype

7903. Use a new environment but there's no 'arguments' object; * delayed environment initialization. This is the most * common case.

7904. E5 Sections 11.9.1, 11.9.3

7905. Capital sigma occurred at "end of word", lowercase to * U+03C2 = GREEK SMALL LETTER FINAL SIGMA. Otherwise * fall through and let the normal rules lowercase it to * U+03C3 = GREEK SMALL LETTER SIGMA.

7906. current digit

7907. * Extern

7908. * Mark objects on finalize_list. *

7909. **comment:** XXX: repetition of stack pre-checks -> helper or macro or inline
label: code-design

7910. **comment:** * Data follows the struct header. The struct size is padded by the * compiler based on the struct members. This guarantees that the * buffer data will be aligned-by-4 but not necessarily aligned-by-8. * * On platforms where alignment does not matter, the struct padding * could be removed (if there is any). On platforms where alignment * by 8 is required, the struct size must be forced to be a multiple * of 8 by some means. Without it, some user code may break, and also * Duktape itself breaks (e.g. the compiler stores duk_tvals in a * dynamic buffer).
label: code-design

7911. 'valueOf'

7912. XXX: potential issue with signed pointers, p_end < p.

7913. * Helpers to resize properties allocation on specific needs.

7914. **comment:** The line number tracking is a bit inconsistent right now, which * affects debugger accuracy. Mostly call sites emit opcodes when * they have parsed a token (say a terminating semicolon) and called * duk__advance(). In this case the line number of the previous * token is the most accurate one (except in prologue where * prev_token.start_line is 0). This is probably not 100% correct * right now.
label: code-design

7915. shared object part

7916. "Have" flags must not be conflicting so that they would * apply to both a plain property and an accessor at the same * time.

7917. [... re_obj input]

7918. Backpointers.

7919. string access cache (codepoint offset -> byte offset) for fast string * character looping; 'weak' reference which needs special handling in GC.

7920. bottom of new func

7921. re_ctx->captures at the start of the atom parsed in this loop

7922. * Scan from shortest distance: * - start of string * - end of string * - cache entry (if exists)

7923. refzero_list treated as reachability roots

7924. * Memory calls.

7925. +0.5 is handled by floor, this is on purpose

7926. Duplicate (shadowing) labels are not allowed, except for the empty * labels (which are used as default labels for switch and iteration * statements). * * We could also allow shadowing of non-empty pending labels without any * other issues than breaking the required label shadowing requirements * of the E5 specification, see Section 12.12.

7927. internal temporary value, used for char classes

7928. **comment:** format is too large, abort
label: code-design

7929. -> [... key val']

7930. 0x40-0x4f

7931. MPUTOBJ emitted by outer loop
7932. Errors are augmented when they are created, not when they are * thrown or re-thrown. The current error handler, however, runs * just before an error is thrown.
7933. -0 is accepted here as index 0 because ToString(-0) == "0" which is * in canonical form and thus an array index.
7934. be_ctx->offset == length of encoded bitstream
7935. 'in'
7936. Duktape.modLoaded[resolved_id] = module
7937. Caller must ensure 'tv' is indeed a fastint!
7938. stage 1 guarantees
7939. typed for duk_unicode_decode_xutf8()
7940. Macros for creating and checking bitmasks for character encoding. * Bit number is a bit counterintuitive, but minimizes code size.
7941. * Compile input string into an executable function template without * arguments. * * The string is parsed as the "Program" production of EcmaScript E5. *
Compilation context can be either global code or eval code (see E5 * Sections 14 and 15.1.2.1). * * Input stack: [... filename] * Output stack: [... func_template]
7942. **comment:** XXX: macro? sets both heaphdr and object flags
label: code-design
7943. [... varname]
7944. preinitialize lexer state partially
7945. **comment:** XXX: slightly awkward
label: code-design
7946. Once recursion depth is increased, exit path must decrease * it (though it's OK to abort the fast path).
7947. require
7948. rnd in [lo,hi]
7949. must have catch and/or finally
7950. V
7951. Technically Array.prototype.push() can create an Array with length * longer than 2^32-1, i.e. outside the 32-bit range. The final length * is *not* wrapped to 32
bits in the specification. * * This implementation tracks length with a uint32 because it's much * more practical. * * See: test-bi-array-push-maxlen.js.
7952. DUK_TOK_BXOR
7953. [... re_obj input bc saved_buf res_obj val]
7954. No arr_idx update or limit check
7955. [... varname]
7956. fresh require (argument)
7957. * Resolution loop. At the top of the loop we're expecting a valid * term: '.', '..', or a non-empty identifier not starting with a period.
7958. String object internal value is immutable
7959. At this point 'res' holds the potential expression value. * It can be basically any ivalue here, including a reg-bound * identifier (if code above deems it safe) or a
unary/binary * operation. Operations must be resolved to a side effect free * plain value, and the side effects must happen exactly once.
7960. highest capture number emitted so far (used as: ++captures)
7961. arbitrary remove only works with double linked heap, and is only required by * reference counting so far.
7962. **comment:** never reached, but avoids warnings of * potentially unused variables.
label: code-design
7963. new entry: previous value is garbage; set to undefined to share write_value
7964. [... name name]
7965. nargs
7966. each call this helper serves has nargs==2
7967. assertion disabled
7968. * Types are different; various cases for non-strict comparison * * Since comparison is symmetric, we use a "swap trick" to reduce * code size.
7969. Ensure dirty state causes a Status even if never process any * messages. This is expected by the bytecode executor when in * the running state.
7970. * A C * B A * C <- sce ==> B * D D
7971. Sign extend: 0x0000##000 -> 0x##000000 -> sign extend to 0xssssss##
7972. * Virtual properties. * * String and buffer indices are virtual and always enumerable, * 'length' is virtual and non-enumerable. Array and arguments * object props
have special behavior but are concrete.
7973. * Case conversion
7974. * Helpers for dealing with the input string
7975. const limits
7976. 0xc0-0xcf
7977. The issues below can be solved with better flags
7978. curr and desc are data
7979. If forced reg, use it as destination. Otherwise try to * use either coerced ispec if it is a temporary. * * When using extraops, avoid reusing arg2 as dest because that
* would lead to an LDREG shuffle below. We still can't guarantee * dest != arg2 because we may have a forced_reg.
7980. **comment:** XXX: putvar takes a duk_tval pointer, which is awkward and * should be reworked.
label: code-design
7981. bottom of current func
7982. statement is guaranteed to be terminal (control doesn't flow to next statement)
7983. Use a fast break/continue when possible. A fast break/continue is * just a jump to the LABEL break/continue jump slot, which then jumps * to an appropriate place
(for break, going through ENDLABEL correctly). * The peephole optimizer will optimize the jump to a direct one.
7984. * Heap buffer representation. * * Heap allocated user data buffer which is either: * * 1. A fixed size buffer (data follows header statically) * 2. A dynamic size
buffer (data pointer follows header) * * The data pointer for a variable size buffer of zero size may be NULL.
7985. Bernstein hash init value is normally 5381; XOR it in in case pointer low bits are 0
7986. **comment:** Allow headroom for calls during error handling (see GH-191). * We allow space for 10 additional recursions, with one extra * for, e.g. a print() call at
the deepest level.
label: code-design
7987. http://en.wikipedia.org/wiki/Exponentiation_by_squaring
7988. chain
7989. [... closure]
7990. **comment:** Calling as a non-constructor is not meaningful.
label: code-design
7991. * Voluntary periodic GC (if enabled)
7992. Step types
7993. 'default'
7994. **comment:** XXX: awkward; we assume there is space for this, overwrite * directly instead?
label: code-design
7995. E5 Section steps 7-8
7996. result object is created and discarded; wasteful but saves code space
7997. Gather in little endian
7998. 'fmt'
7999. not caught by anything before entry level; rethrow and let the * final catcher unwind everything
8000. NUL terminate for convenient C access
8001. 'BYTES_PER_ELEMENT'
8002. We don't log or warn about freeing zero refcount objects * because they may happen with finalizer processing.
8003. Don't allow a zero divisor. Restrict both v1 and * v2 to positive values to avoid compiler specific * behavior.

8004. **comment:** XXX: logic for handling character ranges is now incorrect, it will accept * e.g. [\d-z] whereas it should croak from it? SMJS accepts this too, though. *
* Needs a read through and a lot of additional tests.
label: code-design

8005. * Node.js Buffer.prototype.fill()
8006. * Exposed debug macros: debugging disabled
8007. **comment:** No NUL term checks in this debug code.
label: code-design

8008. DUK_BUFOBJ_FLOAT64ARRAY
8009. embed: duk_hobject ptr
8010. getters
8011. DUK_TOK_DELETE
8012. DUK_TOK_BOR
8013. **comment:** pseudotypes, not used in actual longjmps
label: code-design

8014. A -> target reg * BC -> inner function index
8015. refcount code is processing refzero list
8016. [...] retval]; popped below
8017. 0x80...0x8f
8018. Get current Ecmascript time (= UNIX/Posix time, but in milliseconds).
8019. Note: all references inside 'data' need to get their refcounts * upped too. This is the case because refcounts are decreased * through every function referencing 'data' independently.
8020. **comment:** NOTE: level is used only by the debugger and should never be present * for an Ecmascript eval().
label: code-design

8021. entry_top + 1
8022. Set datetime parts from stack arguments, defaulting any missing values. * Day-of-week is not set; it is not required when setting the time value.
8023. * ToBoolean() (E5 Section 9.2)
8024. * Shift to new valstack_bottom.
8025. 0 = don't push current value
8026. "." is not accepted in any format
8027. tmp -> res
8028. [body formals source]
8029. **comment:** XXX: for simple cases like x['y'] an unnecessary LDREG is * emitted for the base value; could avoid it if we knew that * the key expression is safe (e.g. just a single literal).
label: code-design

8030. Other initial bytes.
8031. Similar to String comparison.
8032. Finally, blame the innermost callstack entry which has a *.fileName property.
8033. skip finalizers; queue finalizable objects to heap_allocated
8034. Note that duk__exptop() here can clobber any reg above current temp_next, * so any loop variables (e.g. enumerator) must be "preallocated".
8035. * Parse a statement list. ** Handles automatic semicolon insertion and implicit return value. ** Upon entry, 'curr_tok' should contain the first token of the first * statement (parsed in the "allow regexp literal" mode). Upon exit, * 'curr_tok' contains the token following the statement list terminator * (EOF or closing brace).
8036. fixed top level hashtable size (separate chaining)
8037. not present
8038. end marker
8039. handle negative values
8040. invalidated by pushes, so get out of the way
8041. h is NULL for lightfunc
8042. * Error throwing
8043. Non-buffer value is first ToString() coerced, then converted * to a buffer (fixed buffer is used unless a dynamic buffer is * explicitly requested).
8044. **comment:** unused: not accepted in inbound messages
label: code-design

8045. Reuse buffer as is unless buffer has grown large.
8046. Offset is coerced first to signed integer range and then to unsigned. * This ensures we can add a small byte length (1-8) to the offset in * bound checks and not wrap.
8047. internal align target for props allocation, must be 2*n for some n
8048. Set timeval to 'this' from dparts, push the new time value onto the * value stack and return 1 (caller can then tail call us). Expects * the value stack to contain 'this' on the stack top.
8049. target
8050. DUK_FLD_FLOAT
8051. XXX: shared helper fortoFixed(), toExponential(), toPrecision()
8052. **comment:** * Run an duk_hobject finalizer. Used for both reference counting * and mark-and-sweep algorithms. Must never throw an error. ** There is no return value. Any return value or error thrown by * the finalizer is ignored (although errors are debug logged). ** Notes: ** - The thread used for calling the finalizer is the same as the * 'thr' argument. This may need to change later. ** - The finalizer thread 'top' assertions are there because it is * critical that strict stack policy is observed (i.e. no cruft * left on the finalizer stack).
label: code-design

8053. -> x in [0, 2**32[
8054. rethrow original error
8055. \xffThis'
8056. Round up digits to a given position. If position is out-of-bounds, * does nothing. If carry propagates over the first digit, a '1' is * prepended to digits and 'k' will be updated. Return value indicates * whether carry propagated over the first digit. ** Note that nc_ctx->count is NOT updated based on the rounding position * (it is updated only if carry overflows over the first digit and an * extra digit is prepended).
8057. Check if any lookup above had a negative result.
8058. DUK_USE_DEBUGGER_SUPPORT
8059. * For/for-in main variants are: ** 1. for(ExpressionNoIn_opt; Expression_opt; Expression_opt) Statement * 2. for(var VariableDeclarationNoIn; Expression_opt; Expression_opt) Statement * 3. for(LeftHandSideExpression in Expression) Statement * 4. for(var VariableDeclarationNoIn in Expression) Statement ** Parsing these without arbitrary lookahead or backtracking is relatively * tricky but we manage to do so for now. ** See doc/compiler.rst for a detailed discussion of control flow * issues, evaluation order issues, etc.
8060. The specification is a bit vague what to do if the return * value is not a number. Other implementations seem to * tolerate non-numbers but e.g. V8 won't apparently do a * ToNumber().
8061. DUK_TOK_SUB_EQ
8062. * Do a longjmp call, calling the fatal error handler if no * catchpoint exists.
8063. Fixed header info.
8064. nothing to finalize
8065. throw flag irrelevant (false in std alg)
8066. **comment:** This approach is a bit shorter than a straight * if-else-ladder and also a bit faster.
label: code-design

8067. All lightfunc own properties are non-writable and the lightfunc * is considered non-extensible. However, the write may be captured * by an inherited setter which means we can't stop the lookup here.

8068. name
8069. accept string if next char is NUL (otherwise reject)
8070. * "WhiteSpace" production check.
8071. value1 -> yield value, iserror -> error / normal
8072. Number of characters that the atom matches (e.g. 3 for 'abc'), * -1 if atom is complex and number of matched characters either * varies or is not known.
8073. **comment:** 'funcs' is quite rarely used, so no local for it
 label: code-design
8074. reachable so pop OK
8075. may be NULL, too
8076. No entry in the catchstack which would actually catch a * throw can refer to the callstack entry being reused. * There *can* be catchstack entries referring to the current * callstack entry as long as they don't catch (e.g. label sites).
8077. * Heap creation and destruction
8078. [... regexp_object escaped_source]
8079. now expected to fit into a 32-bit integer
8080. Duktape.modSearch(resolved_id, fresh_require, exports, module).
8081. bits 0...1: shift
8082. Architecture, OS, and compiler strings
8083. overflow
8084. must fit into duk_small_int_t
8085. 'continue'
8086. 'value'
8087. Non-critical.
8088. **comment:** XXX: tail call: return duk_push_false(ctx)
 label: code-design
8089. **comment:** * Sweep garbage and remove marking flags, and move objects with * finalizers to the finalizer work list. ** Objects to be swept need to get their refcounts finalized before * they are swept. In other words, their target object refcounts * need to be decreased. This has to be done before freeing any * objects to avoid decreffing dangling pointers (which may happen * even without bugs, e.g. with reference loops) ** Because strings don't point to other heap objects, similar * finalization is not necessary for strings.
 label: code-design
8090. Note: target registers a and a+1 may overlap with DUK__REGCONSTP(b) * and DUK__REGCONSTP(c). Careful here.
8091. DUK_USE_DATE_NOW_GETTIMEofday
8092. **comment:** XXX: what's the safest way of creating a negative zero?
 label: code-design
8093. **comment:** XXX: several nice-to-have improvements here: * - Use direct reads avoiding value stack operations * - Avoid triggering getters, indicate getter values to debug client * - If side effects are possible, add error catching
 label: code-design
8094. ptr may be NULL
8095. E5 Sections 15.10.2.8, 7.3
8096. Target must be stored so that we can recheck whether or not * keys still exist when we enumerate. This is not done if the * enumeration result comes from a proxy trap as there is no * real object to check against.
8097. * Note: currently no fast path for array index writes. * They won't be possible anyway as strings are immutable.
8098. Important to do a fastint check so that constants are * properly read back as fastints.
8099. remove artificial bump
8100. important to use normalized NaN with 8-byte tagged types
8101. Only objects in heap_allocated may have finalizers. Check that * the object itself has a _Finalizer property (own or inherited) * so that we don't execute finalizers for e.g. Proxy objects.
8102. crossed offsets or zero size
8103. [... escaped_source bytecode]
8104. * E5 Section 11.4.8
8105. * Manipulate valstack so that args are on the current bottom and the * previous caller's 'this' binding (which is the value preceding the * current bottom) is replaced with the new 'this' binding: * * [... this_old | (crud) func this_new arg1 ... argN] * --> [... this_new | arg1 ... argN] ** For tail calling to work properly, the valstack bottom must not grow * here; otherwise crud would accumulate on the valstack.
8106. * ToInteger() (E5 Section 9.4)
8107. invalidates tv
8108. * Fatal error * * There are no fatal error macros at the moment. There are so few call * sites that the fatal error handler is called directly.
8109. num_stack_args
8110. property is virtual: used in duk_propdesc, never stored * (used by e.g. buffer virtual properties)
8111. \xffVarmap'
8112. Compute (extended) utf-8 length without codepoint encoding validation, * used for string interning. ** NOTE: This algorithm is performance critical, more so than string hashing * in some cases. It is needed when interning a string and needs to scan * every byte of the string with no skipping. Having an ASCII fast path * is useful if possible in the algorithm. The current algorithms were * chosen from several variants, based on x64 gcc -O2 testing. See: * <https://github.com/svaarala/duktape/pull/422> ** NOTE: must match src/dukutil.py:duk_unicode_unvalidated_utf8_length().
8113. Only allow Duktape.Buffer when support disabled.
8114. * Digit generation loop. ** Different termination conditions: * * 1. Free format output. Terminate when shortest accurate * representation found. * * 2. Fixed format output, with specific number of digits. * Ignore termination conditions, terminate when digits * generated. Caller requests an extra digit and rounds. * * 3. Fixed format output, with a specific absolute cut-off * position (e.g. 10 digits after decimal point). Note * that we always generate at least one digit, even if * the digit is below the cut-off point already.
8115. number of digits changed
8116. **comment:** re-lookup to update curr.flags * XXX: would be faster to update directly
 label: code-design
8117. * Heap Buffer object representation. Used for all Buffer variants.
8118. **comment:** Note: reuse variables
 label: code-design
8119. _Varmap
8120. = 1 + arg count
8121. * Tables generated with src/gennumdigits.py. ** duk_str2num_digits_for_radix indicates, for each radix, how many input * digits should be considered significant for string-to-number conversion. * The input is also padded to this many digits to give the Dragon4 * conversion enough (apparent) precision to work with. ** duk_str2num_exp_limits indicates, for each radix, the radix-specific * minimum/maximum exponent values (for a Dragon4 integer mantissa) * below and above which the number is guaranteed to underflow to zero * or overflow to Infinity. This allows parsing to keep bigint values * bounded.
8122. 'case'
8123. num_pairs and temp_start reset at top of outer loop
8124. Because size > 0, NULL check is correct
8125. 17: getUTCHours
8126. [... enum]-> [... next_key]
8127. if density < L, abandon array part, L = 3-bit fixed point, e.g. 2 -> 2/8 = 25%
8128. Encode to CESU-8; 'out' must have space for at least * DUK_UNICODE_MAX_CESU8_LENGTH bytes; codepoints above U+10FFFF * will encode to garbage but won't overwrite the output buffer.
8129. * Finalize stack

8130. Only a reg fits into 'A' so coerce 'res' into a register * for PUTVAR. ** XXX: here the current A/B/C split is suboptimal: we could * just use 9 bits for reg_res (and support constants) and 17 * instead of 18 bits for the varname const index.

8131. * Helper for setting up var_env and lex_env of an activation, * assuming it does NOT have the DUK_HOBJECT_FLAG_NEWENV flag.

8132. no need for 'caller' post-check, because 'key' must be an array index

8133. [arg undefined]

8134. "

8135. * String-to-number conversion

8136. We can't shortcut zero here if it goes through special formatting * (such as forced exponential notation).

8137. 0x10...0x1f

8138. Main operation

8139. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8CLAMPED * Note: INT8 is -not- copy compatible, e.g. -1 would coerce to 0x00.

8140. invalidated

8141. DUK_USE_CPP_EXCEPTIONS

8142. max(incl)

8143. * Arguments objects have exotic [[DefineOwnProperty]] which updates * the internal 'map' of arguments for writes to currently mapped * arguments. More concretely, writes to mapped arguments generate * a write to a bound variable. * * The [[Put]] algorithm invokes [[DefineOwnProperty]] for existing * data properties and new properties, but not for existing accessors. * Hence, in E5 Section 10.6 ([[DefinedOwnProperty]] algorithm), we * have a Desc with 'Value' (and possibly other properties too), and * we end up in step 5.b.i.

8144. A top-level assignment is e.g. "x = y;". For these it's safe * to use the RHS as-is as the expression value, even if the RHS * is a reg-bound identifier. The RHS ('res') is right associative * so it has consumed all other assignment level operations; the * only relevant lower binding power construct is comma operator * which will ignore the expression value provided here. Usually * the top level assignment expression value is ignored, but it * is relevant for e.g. eval code.

8145. **comment:** XXX: use helper for size optimization
label: code-design

8146. 'ignoreCase'

8147. Note: env_thr != thr is quite possible and normal, so careful * with what thread is used for valstack lookup.

8148. frozen and sealed

8149. tv -> value just before prev top value; must relookup

8150. * Found existing inherited plain property. * Do an access control check, and if OK, write * new property to 'orig'.

8151. initial estimate for ASCII only codepoints

8152. Parser separator masks.

8153. just in case callback is broken and won't write 'x'

8154. **comment:** NEXTENUM needs a jump slot right after the main opcode. * We need the code emitter to reserve the slot: if there's * target shuffling, the target shuffle opcodes must happen * after the jump slot (for NEXTENUM the shuffle opcodes are * not needed if the enum is finished).
label: code-design

8155. then to a register

8156. retval to result[i]

8157. Sanity limit on max number of properties (allocated, not necessarily used). * This is somewhat arbitrary, but if we're close to 2**32 properties some * algorithms will fail (e.g. hash size selection, next prime selection). * Also, we use negative array/entry table indices to indicate 'not found', * so anything above 0x80000000 will cause trouble now.

8158. expect EOF instead of {

8159. resizing parameters

8160. value from target

8161. this is especially critical to avoid another write call in detach1()

8162. **comment:** Stripping the filename might be a good idea * (""/foo/bar/quux.js" -> logger name "quux"), * but now used verbatim.
label: code-design

8163. return the argument object

8164. new buffer of specified size

8165. * Conversion helpers

8166. %x; only 16 bits are guaranteed

8167. mix-in value for computing string hashes; should be reasonably unpredictable

8168. 'debug'

8169. keep default instance

8170. Make buffer compact, matching current written size.

8171. explicit semi follows

8172. * Packed 8-byte representation

8173. **comment:** XXX: currently no handling for non-allowed identifier characters, * e.g. a '{' in the function name.
label: code-design

8174. No support for lvalues returned from new or function call expressions. * However, these must NOT cause compile-time SyntaxErrors, but run-time * ReferenceErrors. Both left and right sides of the assignment must be * evaluated before throwing a ReferenceError. For instance: * * f() = g(); * * must result in f() being evaluated, then g() being evaluated, and * finally, a ReferenceError being thrown. See E5 Section 11.13.1.

8175. **comment:** XXX: awkward helpers
label: code-design

8176. advance, expecting current token to be a specific token; parse next token in regexp context

8177. stack prepped for func call: [... trap handler]

8178. {"_nan":true}'

8179. tagged null pointers should never occur

8180. * Heap header definition and assorted macros, including ref counting. * Access all fields through the accessor macros.

8181. * Finally, longjmp

8182. e.g. DUK_OP_POSTINCP

8183. Allow leading minus sign

8184. copy to buffer with spare to avoid Valgrind gripes from strptime

8185. throw

8186. 4: toLocaleDateString

8187. eat 'with'

8188. [... this func]

8189. setters

8190. step 4.c

8191. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)

8192. check func supports [[HasInstance]] (this is checked for every function * in the bound chain, including the final one)

8193. There is no need to NUL delimit the sscanf() call: trailing garbage is * ignored and there is always a NUL terminator which will force an error * if no error is encountered before it. It's possible that the scan * would scan further than between [js_ctx->p,p[though and we'd advance * by less than the scanned value. * * Because pointers are platform specific, a failure to scan a pointer * results in a null pointer which is a better placeholder than a missing * value or an error.

8194. **comment:** XXX: relocate
label: code-design

8195. match fail

8196. * Make a value copy of the input val. This ensures that * side effects cannot invalidate the pointer.

8197. DUK_USE_DATE_TZO_GMTIME

8198. Treat like debugger statement: `nop`
8199. Get valstack index for the func argument or throw if insane stack.
8200. **comment:** 'd3' is never NaN, so no need to normalize
 label: code-design
8201. XXX: `push_uint_string / push_u32_string`
8202. return value from `Duktape.Thread.resume()`
8203. For the case `n==1` `Node.js` doesn't seem to type check * the sole member but we do it before returning it. * For this case only the original buffer object is * returned (not a copy).
8204. end of loop (careful with `len==0`)
8205. Result is always fastint compatible.
8206. Handle a BREAK/CONTINUE opcode. Avoid using `longjmp()` for BREAK/CONTINUE * handling because it has a measurable performance impact in ordinary * environments and an extreme impact in Emscripten (GH-342).
8207. `DUK_TOK_TRY`
8208. Optional UTC conversion.
8209. 6
8210. [requested_id require require.id resolved_id last_comp Duktape modLoaded undefined fresh_require exports module]
8211. * Self test main
8212. prototype is `lex_env` before catcher created
8213. `ToInteger()` coercion; NaN -> 0, infinities are clamped to 0 and 10
8214. Keep throwing an error whenever we get here. The unusual values * are set this way because no instruction is ever executed, we just * throw an error until all try/catch/finally and other catchpoints * have been exhausted. Duktape/C code gets control at each protected * call but whenever it enters back into Duktape the RangeError gets * raised. User exec timeout check must consistently indicate a timeout * until we've fully bubbled out of Duktape.
8215. `tc1 = true, tc2 = false`
8216. no need to handle thread state book-keeping here
8217. Not found. Empty string case is handled specially above.
8218. 4 low bits
8219. **comment:** XXX: option to pretend property doesn't exist if sanity limit is * hit might be useful.
 label: code-design
8220. Use explicit steps in computation to try to ensure that * computation happens with intermediate results coerced to * double values (instead of using something more accurate). * E.g. E5.1 Section 15.9.1.11 requires use of IEEE 754 * rules (= Ecmascript '+' and '*'). * * Without 'volatile' even this approach fails on some platforms * and compiler combinations. For instance, gcc 4.8.1 on Ubuntu * 64-bit, with -m32 and without -std=c99, `test-bi-date-canceling.js` * would fail because of some optimizations when computing `tmp_time` * (MakeTime below). Adding 'volatile' to `tmp_time` solved this * particular problem (annoyingly, also adding debug prints or * running the executable under valgrind hides it).
8221. **comment:** Smaller header than for other objects, because strings are held * in string intern table which requires no link pointers. Much of * the 32-bit flags field is unused by flags, so we can stuff a 16-bit * field in there.
 label: code-design
8222. **comment:** Delete semantics are a bit tricky. The description in E5 specification * is kind of confusing, because it distinguishes between resolvability of * a reference (which is only known at runtime) seemingly at compile time * (= `SyntaxError` throwing).
 label: code-design
8223. [... enum] -> [...]
8224. * Augment an error being created using Duktape specific properties * like `_Tracedata` or `.fileName/.lineNumber`.
8225. misc constants and helper macros
8226. zero-based day
8227. 28: `setMinutes`
8228. non-object base, no offending virtual property
8229. Fixed seed value used with ROM strings.
8230. `duk_parse_stmts()` expects `curr_tok` to be set; parse in "allow regexp literal" mode with current strictness
8231. E5 Sections 11.8.1, 11.8.5; $x < y$
8232. may be < `thr->catchstack` initially
8233. complex, multicharacter conversion
8234. Not enough data to provide a full window, so "scroll" window to * start of buffer and fill up the rest.
8235. E5.1 Section 15.1.3.1: `uriReserved + '#'`
8236. **comment:** XXX: should there be an error or an automatic detach if * already attached?
 label: code-design
8237. For user code this could just return 1 (strict) always * because all Duktape/C functions are considered strict, * and strict is also the default when nothing is running. * However, Duktape may call this function internally when * the current activation is an Ecmascript function, so * this cannot be replaced by a 'return 1' without fixing * the internal call sites.
8238. "" was eaten by caller
8239. expression is left-hand-side compatible
8240. string is a reserved word (strict)
8241. **comment:** XXX: Output type could be encoded into native function 'magic' value to * save space. For setters the stridx could be encoded into 'magic'.
 label: code-design
8242. **comment:** * Manipulation of thread stacks (valstack, callstack, catchstack). * * Ideally unwinding of stacks should have no side effects, which would * then favor separate unwinding and shrink check primitives for each * stack type. A shrink check may realloc and thus have side effects. * * However, currently callstack unwinding itself has side effects, as it * needs to DECREF multiple objects, close environment records, etc. * Stacks must thus be unwound in the correct order by the caller. * * (XXX: This should be probably reworked so that there is a shared * unwind primitive which handles all stacks as requested, and knows * the proper order for unwinding.) * * Valstack entries above 'top' are always kept initialized to * "undefined unused". Callstack and catchstack entries above 'top' * are not zeroed and are left as garbage. * * Value stack handling is mostly a part of the API implementation.
 label: code-design
8243. Step 9: copy elements-to-be-deleted into the result array
8244. 0x30-0x3f
8245. 2 bits for heap type
8246. `setjmp` catchpoint setup
8247. **comment:** not sure whether or not needed; Thursday
 label: requirement
8248. emergency mode: try extra hard
8249. Set object 'length'.
8250. byte index limit for element access, exclusive
8251. We could rely on max temp/const checks: if they don't exceed BC * limit, nothing here can either (just asserts would be enough). * Currently we check for the limits, which provides additional * protection against creating invalid bytecode due to compiler * bugs.
8252. require initializer for var/const
8253. `DUK_TOK_FALSE`
8254. fall-through control flow patchup; note that `pc_prevstmt` may be * < 0 (i.e. no case clauses), in which case this is a no-op.
8255. [... eval "eval" eval_input level]
8256. * `ToObject()` (E5 Section 9.9) * * ==> implemented in the API.
8257. `DUK_USE_MARKANDSWEEP_FINALIZER_TORTURE`
8258. **comment:** * Handle integers in 32-bit range (that is, $[-(2^{32}-1), 2^{32}-1]$) * specially, as they're very likely for embedded programs. This * is now done for all radix values. We must be careful not to use * the fast path when special formatting (e.g. forced exponential) * is in force. * * XXX: could save space by supporting

radix 10 only and using * sprintf "%lu" for the fast path and for exponent formatting.

label: code-design

8259. !(p < r)

8260. Maximum exponent value when parsing numbers. This is not strictly * compliant as there should be no upper limit, but as we parse the * exponent without a bigint, impose some limit.

8261. num_stack_args

8262. **comment:** Executor interrupt counter check, used to implement breakpoints, * debugging interface, execution timeouts, etc. The counter is heap * specific but is maintained in the current thread to make the check * as fast as possible. The counter is copied back to the heap struct * whenever a thread switch occurs by the DUK_HEAP_SWITCH_THREAD() macro.

label: code-design

8263. complete 'sub atom'

8264. * Intern key via the valstack to ensure reachability behaves * properly. We must avoid longjmp's here so use non-checked * primitives. * * Note: duk_check_stack() potentially reallocs the valstack, * invalidating any duk_tval pointers to valstack. Callers * must be careful.

8265. round out to 8 bytes

8266. for convenience

8267. buffer limits

8268. Handle TypedArray vs. Node.js Buffer arg differences

8269. DUK_TOK_WHILE

8270. offset/value order different from Node.js

8271. * Other cases (function declaration, anonymous function expression, * strict direct eval code). The "outer" environment will be whatever * the caller gave us.

8272. * Setup environment record properties based on the template and * its flags. * * If DUK_HOBJECT_HAS_NEWENV(fun_temp) is true, the environment * records represent identifiers "outside" the function; the * "inner" environment records are created on demand. Otherwise, * the environment records are those that will be directly used * (e.g. for declarations). * * _Lexenv is always set; _Varenv defaults to _Lexenv if missing, * so _Varenv is only set if _Lexenv != _Varenv. * * This is relatively complex, see doc/identifier-handling.rst.

8273. * Detach actual buffer from dynamic buffer in valstack, and * pop it from the stack. * * XXX: the buffer object is certainly not reachable at this point, * so it would be nice to free it forcibly even with only * mark-and-sweep enabled. Not a big issue though.

8274. continue matched this label -- we can only continue if this is the empty * label, for which duplication is allowed, and thus there is hope of * finding a match deeper in the label stack.

8275. Plus sign must be accepted for positive exponents * (e.g. '1.5e+2'). This clause catches NULs.

8276. DUK_TOK_VOID

8277. DUK_USE_PC2LINE

8278. DUK_HOBJECT_FLAG_STRICT varies

8279. [... errval errhandler]

8280. **comment:** XXX: Valstack, callstack, and catchstack are currently assumed * to have non-NULL pointers. Relaxing this would not lead to big * benefits (except perhaps for terminated threads).

label: code-design

8281. serialize the wrapper with empty string key

8282. * Free a heap object. * * Free heap object and its internal (non-heap) pointers. Assumes that * caller has removed the object from heap allocated list or the string * intern table, and any weak references (which strings may have) have * been already dealt with.

8283. BITWISE EXPRESSIONS

8284. **comment:** * Copy some internal properties directly * * The properties will be writable and configurable, but not enumerable.

label: code-design

8285. Valstack resize flags

8286. **comment:** XXX: Hex encoded, length limited buffer summary here?

label: code-design

8287. finish up pending jump and split for last alternative

8288. nop

8289. * Local declarations.

8290. stack[0] = object (this) * stack[1] = ToUInt32(length) * stack[2] = elem at index 0 (retval)

8291. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers). A prototype loop must not cause * an error to be thrown here; duk_hobject_hasprop_raw() will ignore a * prototype loop silently and indicate that the property doesn't exist.

8292. **comment:** XXX: coerce to regs? it might be better for enumeration use, where the * same PROP ivalue is used multiple times. Or perhaps coerce PROP further * there?

label: code-design

8293. x > y --> y < x

8294. * Found existing accessor property (own or inherited). * Call setter with 'this' set to orig, and value as the only argument. * Setter calls are OK even for ROM objects. * * Note: no exotic arguments object behavior, because [[Put]] never * calls [[DefineOwnProperty]] (E5 Section 8.12.5, step 5.b).

8295. resume: [...] initial_func] (currently actually: [initial_func])

8296. Lookup 'key' from arguments internal 'map', and leave replacement value * on stack top if mapped (and return non-zero). * Used in E5 Section 10.6 algorithm for [[GetOwnProperty]] (used by [[Get]]).

8297. proto can also be NULL here (allowed explicitly)

8298. * split()

8299. Validate that the whole slice [0,length[is contained in the underlying * buffer. Caller must ensure 'buf' != NULL.

8300. could also just pop?

8301. Parser part indices.

8302. Compiler SyntaxError (or other error) gets the primary blame. * Currently no flag to prevent blaming.

8303. We don't check the zero padding bytes here right now * (that they're actually zero). This seems to be common * behavior for base-64 decoders.

8304. An object may be in heap_allocated list with a zero * refcount if it has just been finalized and is waiting * to be collected by the next cycle.

8305. * Local prototypes

8306. low-level property functions

8307. helpers for defineProperty() and defineProperties()

8308. w/o refcounts

8309. Skip dvalues to EOM.

8310. all native functions have NEWENV

8311. +1, right after inserted jump

8312. only accept lowercase 'utf8' now.

8313. **comment:** XXX: duplicates should be eliminated here

label: code-design

8314. **comment:** XXX: block removal API primitive

label: code-design

8315. LOGICAL EXPRESSIONS

8316. %lf, %ld etc

8317. * Switch is pretty complicated because of several conflicting concerns: * * - Want to generate code without an intermediate representation, * i.e., in one go * * - Case selectors are expressions, not values, and may thus e.g. throw * exceptions (which causes evaluation order concerns) * * - Evaluation semantics of case selectors and default clause need to be * carefully implemented to provide correct behavior even with case value * side effects * * - Fall through case and default clauses; avoiding dead JUMPs if case * ends with an unconditional jump (a break or a continue) * * - The same case value may occur multiple times, but evaluation rules * only process the first match before switching to a "propagation" mode * where case values are no longer evaluated * * See E5 Section 12.11. Also see doc/compiler.rst for compilation * discussion.

8318. 8: getFullYear
8319. * Debug connection peek and flush primitives
8320. 'do'
8321. duk_xdef_prop() will define an own property without any array * special behaviors. We'll need to set the array length explicitly * in the end. For arrays with elisions, the compiler will emit an * explicit SETALEN which will update the length.
8322. exponential notation forced
8323. NB: 'val' may be invalidated here because put_value may realloc valstack, * caller beware.
8324. Here we'd have the option to normalize -0 to +0.
8325. **comment:** XXX: share helper from lexer; duk_lexer.c / hexval().
label: code-design
8326. **comment:** * Create an internal enumerator object E, which has its keys ordered * to match desired enumeration ordering. Also initialize internal control * properties for enumeration. * * Note: if an array was used to hold enumeration keys instead, an array * scan would be needed to eliminate duplicates found in the prototype chain.
label: code-design
8327. **comment:** This is essentially the 'scale' algorithm, with recursion removed. * Note that 'k' is either correct immediately, or will move in one * direction in the loop. There's no need to do the low/high checks * on every round (like the Scheme algorithm does). * * The scheme algorithm finds 'k' and updates 's' simultaneously, * while the logical algorithm finds 'k' with 's' having its initial * value, after which 's' is updated separately (see the Burger-Dybvig * paper, Section 3.1, steps 2 and 3). * * The case where m+ == m- (almost always) is optimized for, because * it reduces the bigint operations considerably and almost always * applies. The scale loop only needs to work with m+, so this works.
label: code-design
8328. A -> flags * BC -> regCatch; base register for two registers used both during * trycatch setup and when catch is triggered * * If DUK_BC_TRYCATCH_FLAG_CATCH_BINDING set: * regCatch + 0: catch binding variable name (string). * Automatic declarative environment is established for * the duration of the 'catch' clause. * * If DUK_BC_TRYCATCH_FLAG_WITH_BINDING set: * regCatch + 0: with 'target value', which is coerced to * an object and then used as a bindind object for an * environment record. The binding is initialized here, for * the 'try' clause. * * Note that a TRYCATCH generated for a 'with' statement has no * catch or finally parts.
8329. default: false
8330. **comment:** fill new_h with u32 0xff = UNUSED
label: code-design
8331. A non-extensible object cannot gain any more properties, * so this is a good time to compact.
8332. if direct eval, calling activation must exist
8333. **comment:** XXX: because of the final check below (that the literal is not * followed by a digit), this could maybe be simplified, if we bail * out early from a leading zero (and if there are no periods etc). * Maybe too complex.
label: code-design
8334. * Create escaped RegExp source (E5 Section 15.10.3). * * The current approach is to special case the empty RegExp * (" -> '(?:')") and otherwise replace unescaped '/' characters * with '\' regardless of where they occur in the regexp. * * Note that normalization does not seem to be necessary for * RegExp literals (e.g. '/foo/') because to be acceptable as * a RegExp literal, the text between forward slashes must * already match the escaping requirements (e.g. must not contain * unescaped forward slashes or be empty). Escaping IS needed * for expressions like 'new RegExp("...","")' however. * Currently, we re-escape in either case. * * Also note that we process the source here in UTF-8 encoded * form. This is correct, because any non-ASCII characters are * passed through without change.
8335. Value stack is used to ensure reachability of constants and * inner functions being loaded. Require enough space to handle * large functions correctly.
8336. **comment:** Automatic defaulting of logger name from caller. This would * work poorly with tail calls, but constructor calls are currently * never tail calls, so tail calls are not an issue now.
label: code-design
8337. Fast check for extending array: check whether or not a slow density check is required.
8338. **comment:** XXX: using duk_put_prop_index() would cause obscure error cases when Array.prototype * has write-protected array index named properties. This was seen as DoubleErrors * in e.g. some test262 test cases. Using duk_xdef_prop_index() is better but heavier. * The best fix is to fill in the tracedata directly into the array part. There are * no side effect concerns if the array part is allocated directly and only INCREFs * happen after that.
label: code-design
8339. filename may be NULL in which case file/line is not recorded
8340. unsigned
8341. **comment:** XXX: function flag to make this automatic?
label: requirement
8342. Must have array part and no conflicting exotic behaviors. * Doesn't need to have array special behavior, e.g. Arguments * object has array part.
8343. Breakpoint entries above the used area are left as garbage.
8344. allow_source_elem
8345. JSON parsing code is allowed to read [p_start,p_end]: p_end is * valid and points to the string NUL terminator (which is always * guaranteed for duk_hstrings).
8346. fall through if overflow etc
8347. inner function numbering
8348. **comment:** XXX: could write output in chunks with fewer ensure calls, * but relative benefit would be small here.
label: code-design
8349. Assumes that saved[0] and saved[1] are always * set by regexp bytecode (if not, char_end_offset * will be zero). Also assumes clen reflects the * correct char length.
8350. **comment:** XXX: V8 throws a TypeError for negative values. Would it * be more useful to interpret negative offsets here from the * end of the buffer too?
label: code-design
8351. function needs shuffle registers
8352. The compiler should never emit DUK_OP_REGEXP if there is no * regexp support.
8353. invalid padding
8354. Note: here we must be wary of the fact that a pointer may be * valid and be a NULL.
8355. **comment:** Here we can choose either to end parsing and ignore * whatever follows, or to continue parsing in case * multiple (possibly padded) base64 strings have been * concatenated. Currently, keep on parsing.
label: code-design
8356. [requested_id require require.id resolved_id last_comp]
8357. advance to get one step of lookup
8358. Write signed 32-bit integer.
8359. * Canonicalize() abstract operation needed for canonicalization of individual * codepoints during regexp compilation and execution, see E5 Section 15.10.2.8. * Note that codepoints are canonicalized one character at a time, so no context * specific rules can apply. Locale specific rules can apply, though.
8360. This should not be necessary because no-one should tamper with the * regexp bytecode, but is prudent to avoid potential segfaults if that * were to happen for some reason.
8361. Needs lookahead
8362. * Churn refzero_list until empty
8363. Note: 'q_instr' is still used below
8364. duk_unicode_caseconv_uc[]
8365. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. * * XXX: this is now an overkill for many fast paths. Rework this * to be faster (although switching to a valstack discipline might * be a better solution overall).
label: code-design
8366. failed, resize and try again
8367. extend with undefined
8368. **comment:** * Traceback handling * * The unified helper decodes the traceback and produces various requested * outputs. It should be optimized for size, and may leave garbage on stack, * only the topmost return value matters. For instance, traceback separator * and decoded strings are pushed even when looking for filename

only. ** NOTE: although _Tracedata is an internal property, user code can currently * write to the array (or replace it with something other than an array). * The code below must tolerate arbitrary _Tracedata. It can throw errors * etc, but cannot cause a segfault or memory unsafe behavior.

label: code-design

8369. Error; error value is in heap->j.value1.

8370. -> [... key val toJSON val key]

8371. callstack

8372. read 3 bytes into 't', padded by zero

8373. XXX: remove this feature entirely? it would only matter for * emergency GC. Disable for lowest memory builds.

8374. * Computed values (e.g. INFINITY)

8375. DUK_TOK_DIV_EQ

8376. already set

8377. input 'd' in Windows UTC, 100ns units

8378. Keep current size

8379. 50x heap size

8380. Dynamic buffer with 'curr_alloc' pointing to a dynamic area allocated using * heap allocation primitives. Also used for external buffers when low memory * options are not used.

8381. [name this]

8382. [... holder name val enum obj_key new_elem]

8383. patched later

8384. [... regexp_object]

8385. **comment:** * Note: the description in E5 Section 15.10.2.6 has a typo, it * contains 'A' twice and lacks 'a'; the intent is [0-9a-zA-Z].

label: documentation

8386. This was the last term, and q_last was * updated to match this term at loop top.

8387. x <- b^y; use t1 and t2 as temps

8388. prev_token.start_offset points to the closing brace here; when skipping * we're going to reparse the closing brace to ensure semicolon insertion * etc work as expected.

8389. more initializers

8390. ref.holder is global object, holder is the object with the * conflicting property.

8391. essentially tracks digit position of lowest 'f' digit

8392. **comment:** Prevent any side effects on the string table and the caller provided * str/blen arguments while interning is in progress. For example, if * the caller provided str/blen from a dynamic buffer, a finalizer might * resize that dynamic buffer, invalidating the call arguments.

label: code-design

8393. binding power "levels" (see doc/compiler.rst)

8394. Enter message processing loop for sending Status notifys and * to finish a pending detach.

8395. result reg

8396. strings are only tracked by stringtable

8397. to avoid relookups

8398. abandoning requires a props allocation resize and * 'rechecks' the valstack, invalidating any existing * valstack value pointers!

8399. If object has a .toJSON() property, we can't be certain * that it wouldn't mutate any value arbitrarily, so bail * out of the fast path. * * If an object is a Proxy we also can't avoid side effects * so abandon.

8400. !'

8401. **comment:** Unlike in duk_hex_encode() 'dst' is not necessarily aligned by 2. * For platforms where unaligned accesses are not allowed, shift 'dst' * ahead by 1 byte to get alignment and then DUK_MEMMOVE() the result * in place. The faster encoding loop makes up the difference. * There's always space for one extra byte because a terminator always * follows the hex data and that's been accounted for by the caller.

label: code-design

8402. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.

8403. **comment:** * 'nregs' registers are allocated on function entry, at most 'nargs' * are initialized to arguments, and the rest to undefined. Arguments * above 'nregs' are not mapped to registers. All registers in the * active stack range must be initialized because they are GC reachable. * 'nargs' is needed so that if the function is given more than 'nargs' * arguments, the additional arguments do not 'clobber' registers * beyond 'nregs' which must be consistently initialized to undefined. * * Usually there is no need to know which registers are mapped to * local variables. Registers may be allocated to variable in any * way (even including gaps). However, a register-variable mapping * must be the same for the duration of the function execution and * the register cannot be used for anything else. * * When looking up variables by name, the '_Varmap' map is used. * When an activation closes, registers mapped to arguments are * copied into the environment record based on the same map. The * reverse map (from register to variable) is not currently needed * at run time, except for debugging, so it is not maintained.

label: code-design

8404. Slice starting point is beyond current length.

8405. ... target is extensible

8406. duk_unicode_support.c

8407. **comment:** XXX: easier check with less code?

label: code-design

8408. Used when precision is undefined; also used for NaN (-> "NaN"), * and +/- infinity (-> "Infinity", "-Infinity").

8409. x <- x * y, use t as temp

8410. [... v1 v2 str] -> [... str v2]

8411. Zero 'count' is also allowed to make call sites easier.

8412. **comment:** XXX: size optimize

label: code-design

8413. != 0 => post-update for array 'length' (used when key is an array index)

8414. **comment:** has inner functions which may slow access (XXX: this can be optimized by looking at the inner functions)

label: code-design

8415. **comment:** XXX: need to determine LHS type, and check that it is LHS compatible

label: code-design

8416. false

8417. * Built-in strings

8418. * Clear (reachable) flags of finalize_list * * We could mostly do in the sweep phase when we move objects from the * heap into the finalize_list. However, if a finalizer run is skipped * during a mark-and-sweep, the objects on the finalize_list will be marked * reachable during the next mark-and-sweep. Since they're already on the * finalize_list, no-one will be clearing their REACHABLE flag so we do it * here. (This now overlaps with the sweep handling in a harmless way.)

8419. * Variant 3

8420. * On second pass, skip the function.

8421. Note: key issue here is to re-lookup the base pointer on every attempt. * The pointer being reallocated may change after every mark-and-sweep.

8422. -> [... key val toJSON val]

8423. **comment:** XXX: It would be nice to build the string directly but ToUint16() * coercion is needed so a generic helper would not be very * helpful (perhaps coerce the value stack first here and then * build a string from a duk_tval number sequence in one go?).

label: code-design

8424. Note: getprop may invoke any getter and invalidate any * duk_tval pointers, so this must be done first.

8425. [obj trap_result res_arr propname]

8426. split1: prefer direct execution (no jump)

8427. in resumer's context

8428. Conceptually we'd extract the plain underlying buffer * or its slice and then do a type mask check below to * see if we should reject it. Do the mask check here * instead to avoid making a copy of the buffer slice.

8429. 'try'

8430. -> [in_val]

8431. **comment:** Currently all built-in native functions are strict. * This doesn't matter for many functions, but e.g. * String.prototype.charAt (and other string functions) * rely on being strict so that their 'this' binding is * not automatically coerced.

label: code-design

8432. **comment:** * Helper for handling an Ecmascript-to-Ecmascript call or an Ecmascript * function (initial) Duktape.Thread.resume(). * * Compared to normal calls handled by duk_handle_call(), there are a * bunch of differences: * * - the call is never protected * - there is no C recursion depth increase (hence an "ignore recursion * limit" flag is not applicable) * - instead of making the call, this helper just performs the thread * setup and returns; the bytecode executor then restarts execution * internally * - ecmascript functions are never 'vararg' functions (they access * varargs through the 'arguments' object) * * The callstack of the target contains an earlier Ecmascript call in case * of an Ecmascript-to-Ecmascript call (whose idx_retval is updated), or * is empty in case of an initial Duktape.Thread.resume(). * * The first thing to do here is to figure out whether an ecma-to-ecma * call is actually possible. It's not always the case if the target is * a bound function; the final function may be native. In that case, * return an error so caller can fall back to a normal call path.

label: code-design

8433. Note: 0xff != DUK_BC_B_MAX

8434. * Unions and structs for self tests

8435. Note: masking is done for 'i' to deal with negative numbers correctly

8436. * For property layout 1, tweak e_size to ensure that the whole entry * part (key + val + flags) is a suitable multiple for alignment * (platform specific). * * Property layout 2 does not require this tweaking and is preferred * on low RAM platforms requiring alignment.

8437. don't run finalizers; queue finalizable objects back to heap_allocated

8438. val1 = count

8439. * Function declaration, function expression, or (non-standard) * function statement. * * The E5 specification only allows function declarations at * the top level (in "source elements"). An ExpressionStatement * is explicitly not allowed to begin with a "function" keyword * (E5 Section 12.4). Hence any non-error semantics for such * non-top-level statements are non-standard. Duktape semantics * for function statements are modelled after V8, see * test-dev-func-decl-outside-top.js.

8440. Get minimum array part growth for a certain size.

8441. **comment:** XXX: very slow - better to bulk allocate a gap, and copy * from args_array directly (we know it has a compact array * part, etc).

label: code-design

8442. 'enumerate'

8443. DUK_TOK_RSHIFT

8444. roughly 256 bytes

8445. **comment:** For now, just use duk_safe_call() to wrap duk_new(). We can't * simply use a protected duk_handle_call() because there's post * processing which might throw. It should be possible to ensure * the post processing never throws (except in internal errors and * out of memory etc which are always allowed) and then remove this * wrapper.

label: code-design

8446. object is now reachable

8447. flags: "im"

8448. Copy trap result keys into the enumerator object.

8449. * Mark-and-sweep flags * * These are separate from heap level flags now but could be merged. * The heap structure only contains a 'base mark-and-sweep flags' * field and the GC caller can impose further flags.

8450. Success path.

8451. * If entering a 'catch' block which requires an automatic * catch variable binding, create the lexical environment. * * The binding is mutable (= writable) but not deletable. * Step 4 for the catch production in E5 Section 12.14; * no value is given for CreateMutableBinding 'D' argument, * which implies the binding is not deletable.

8452. t1 <- (* r B)

8453. DUK_BUFOBJ_UINT8ARRAY

8454. Accept any duk_hbufferobject, though we're only normally * called for Duktape.Buffer values.

8455. Lenient: allow function declarations outside top level in * non-strict mode but reject them in strict mode.

8456. assertions: env must be closed in the same thread as where it runs

8457. EOF (-1) will be cast to an unsigned value first * and then re-cast for the switch. In any case, it * will match the default case (syntax error).

8458. refzero not running -> must be empty

8459. **comment:** XXX: differentiate null pointer from empty string?

label: code-design

8460. Normal object which doesn't get automatically coerced to a * primitive value. Functions are checked for specially. The * primitive value coercions for Number, String, Pointer, and * Boolean can't result in functions so suffices to check here.

8461. DUK_USE_MARK_AND_SWEEP

8462. * Coercion and fast path processing

8463. y == +Infinity

8464. caller required to know

8465. * Function name (if any) * * We don't check for prohibited names here, because we don't * yet know whether the function will be strict. Function body * parsing handles this retroactively. * * For function expressions and declarations function name must * be an Identifier (excludes reserved words). For setter/getter * it is aPropertyName which allows reserved words and also * strings and numbers (e.g. "{ get 1() { ... } }").

8466. **comment:** XXX: There are no format strings in duk_config.h yet, could add * one for formatting duk_int64_t. For now, assumes "%lld" and that * "long long" type exists. Could also rely on C99 directly but that * won't work for older MSVC.

label: code-design

8467. 'toLocaleString'

8468. * Manipulate value stack so that exactly 'num_stack_rets' return * values are at 'idx_rebase' in every case, assuming there are * 'rc' return values on top of stack. * * This is a bit tricky, because the called C function operates in * the same activation record and may have e.g. popped the stack * empty (below idx_rebase).

8469. Exotic behaviors are only enabled for arguments objects * which have a parameter map (see E5 Section 10.6 main * algorithm, step 12). * * In particular, a non-strict arguments object with no * mapped formals does *NOT* get exotic behavior, even * for e.g. "caller" property. This seems counterintuitive * but seems to be the case.

8470. buffer is automatically zeroed

8471. num_args

8472. DUK_OP_EXTRA, sub-operation in A

8473. insert the required matches (qmin) by copying the atom

8474. DUK_USE_INTEGER_BE

8475. out_clamped

8476. adv = 2 default OK

8477. There is no need to decref anything else than 'env': if 'env' * becomes unreachable, refzero will handle decref'ing its prototype.

8478. '-0'

8479. module table remains registered to modLoaded, minimize its size

8480. * Detach handling

8481. write back

8482. [... filename (temp) func]

8483. **comment:** It's not strictly necessary to track the current size, but * it is useful for writing robust native code.

label: code-design

8484. * Second pass parsing.

8485. **comment:** Note: cannot portably debug print a function pointer, hence 'func' not printed!

label: code-design

8486. * Arbitrary byteswap for potentially unaligned values * * Used to byteswap pointers e.g. in debugger code.

8487. **comment:** XXX: There's some overlap with duk_js_closure() here, but * seems difficult to share code. Ensure that the final function * looks the same as created by duk_js_closure().**label:** code-design8488. **comment:** It is important not to call this if the last byte read was an EOM. * Reading ahead in this scenario would cause unnecessary blocking if * another message is not available.**label:** code-design

8489. DUK_USE_EXEC_TIMEOUT_CHECK

8490. * Misc defines

8491. E5.1 Section 15.1.3.4: uriUnescaped

8492. Combined size of separators already overflows

8493. -> [... res]

8494. [... this name message]

8495. "/" and not in regexp mode

8496. DUK_USE_BYTECODE_DUMP_SUPPORT

8497. **comment:** may indirectly slow access through a direct eval**label:** code-design

8498. no incref needed

8499. variable binding register if register-bound (otherwise < 0)

8500. How much to advance before next loop; note that next loop * will advance by 1 anyway, so -1 from the total escape * length (e.g. len("\uXXXXX") - 1 = 6 - 1). As a default, * 1 is good.

8501. **comment:** 'sce' points to the wrong entry here, but is no longer used**label:** code-design

8502. Note: this check relies on the fact that even a zero-size string * has a non-NUL pointer.

8503. [... varname val this] (because throw_flag == 1, always resolved)

8504. No need to reinit setjmp() catchpoint, as call handling * will store and restore our state.

8505. embed: duk_hbuffer ptr

8506. t >= 0 always true, unsigned

8507. Maximum expansion per input byte is 6: * - invalid UTF-8 byte causes "\uXXXX" to be emitted (6/1 = 6). * - 2-byte UTF-8 encodes as "\uXXXX" (6/2 = 3). * - 4-byte UTF-8 encodes as "\Uxxxxxxxx" (10/4 = 2.5).

8508. Shared helper to implement Object.getPrototypeOf and the ES6 * Object.prototype.__proto__ getter. * * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype._proto_

8509. xxx -> DUK_HBUFFEROBJECT_ELEM_INT16

8510. vararg function, careful arg handling (e.g. thisArg may not be present)

8511. implicit this value always undefined for * declarative environment records.

8512. **comment:** * IsCallable() (E5 Section 9.11) * * XXX: API equivalent is a separate implementation now, and this has * currently no callers.**label:** code-design

8513. Extract 'top' bits of curval; note that the extracted bits do not need * to be cleared, we just ignore them on next round.

8514. duk_unicode_ids_noabmp[]

8515. Return value of 'sz' or more indicates output was (potentially) * truncated.

8516. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers).

8517. Encoding/decoding flags

8518. Shared helper to implement ES6 Object.setPrototypeOf and * Object.prototype.__proto__ setter. * * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype._proto_ * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.setprototypeof

8519. no need to write 'out_bufdata'

8520. new_used / size <= 1 / DIV <=> new_used <= size / DIV

8521. Register the module table early to modLoaded[] so that we can * support circular references even in modSearch(). If an error * is thrown, we'll delete the reference.

8522. label handling

8523. Expression

8524. **comment:** This is performance critical because it appears in every DECREF.**label:** code-design

8525. code_idx = entry_top + 0

8526. -> [... lval rval]

8527. * Standard algorithm succeeded without errors, check for exotic post-behaviors. * * Arguments exotic behavior in E5 Section 10.6 occurs after the standard * [[DefineOwnProperty]] has completed successfully. * * Array exotic behavior in E5 Section 15.4.5.1 is implemented partly * prior to the default [[DefineOwnProperty]], but: * - for an array index key (e.g. "10") the final 'length' update occurs here * - for 'length' key the element deletion and 'length' update occurs here

8528. * All defined array-indexed properties are in the array part * (we assume the array part is comprehensive), and all array * entries are writable, configurable, and enumerable. Thus, * nothing can prevent array entries from being deleted.

8529. **comment:** * Helpers for creating and querying pc2line debug data, which * converts a bytecode program counter to a source line number. * * The run-time pc2line data is bit-packed, and documented in: * * doc/function-objects.rst**label:** documentation

8530. Represent a null pointer as 'null' to be consistent with * the JX format variant. Native '%p' format for a NULL * pointer may be e.g. '(nil)'.

8531. eat 'throw'

8532. always grow the array, no sparse / abandon support here

8533. leave formals on stack for later use

8534. since index non-negative

8535. tzoffset

8536. duk_has_prop() popped the second 'name'

8537. varname

8538. * E5 Section 15.10.2.6 "IsWordChar" abstract operation. Assume * x < 0 for characters read outside the string.

8539. -----XXXX

8540. stack top: new time value, return 1 to allow tail calls

8541. basereg

8542. E5 Section 15.4.5.1, step 3, steps a - i are implemented here, j - n at the end

8543. jump for 'catch' case

8544. Module object containing module.exports, etc

8545. 'prototype' property for all built-in objects (which have it) has attributes: * [[Writable]] = false, * [[Enumerable]] = false, * [[Configurable]] = false

8546. stack[0] = start * stack[1] = deleteCount * stack[2...nargs-1] = items * stack[nargs] = ToObject(this) -3 * stack[nargs+1] = ToUint32(length) -2 * stack[nargs+2] = result array -1

8547. -> [... escaped_source bytecode]

8548. * Write to array part? * * Note: array abandoning requires a property resize which uses * 'rechecks' valstack for temporaries and may cause any existing * valstack pointers to be invalidated. To protect against this, * tv_obj, tv_key, and tv_val are copies of the original inputs.

8549. copy existing entries as is

8550. 5: toLocaleTimeString

8551. mark-and-sweep: children not processed

8552. Catches EOF of JSON input.

8553. radix

8554. **comment:** XXX: when optimizing for guaranteed property slots, use a guaranteed * slot for internal value; this call can then access it directly.

label: code-design

8555. no wrapping

8556. **comment:** * Create and throw an error (originating from Duktape internally) * * Push an error object on top of the stack, possibly throw augmenting * the error, and finally longjmp. * * If an error occurs while we're dealing with the current error, we might * enter an infinite recursion loop. This is prevented by detecting a * "double fault" through the heap->handling_error flag; the recursion * then stops at the second level.

label: code-design

8557. if property key begins with underscore, encode it with * forced quotes (e.g. "_Foo") to distinguish it from encoded * internal properties (e.g. \xffBar -> _Bar).

8558. object or array

8559. no need to check now: both success and error are OK

8560. Note: allow backtracking from p == ptr_end

8561. Relookup if relocated

8562. module.filename for .fileName, default to resolved ID if * not present.

8563. -> [... toLocaleString ToObject(val)]

8564. DUK_TOK_LBRACKET already eaten, current token is right after that

8565. **comment:** * Array needs to grow, but we don't want it becoming too sparse. * If it were to become sparse, abandon array part, moving all * array entries into the entries part (for good). * * Since we don't keep track of actual density (used vs. size) of * the array part, we need to estimate somehow. The check is made * in two parts: * * - Check whether the resize need is small compared to the * current size (relatively); if so, resize without further * checking (essentially we assume that the original part is * "dense" so that the result would be dense enough). * * - Otherwise, compute the resize using an actual density * measurement based on counting the used array entries.

label: code-design

8566. Restore stack top after unbalanced code paths.

8567. position of highest digit changed

8568. * duk_heap allocation and freeing.

8569. Careful with borrow condition: * - If borrow not subtracted: 0x12345678 - 0 - 0xffffffff = 0x12345679 (> 0x12345678) * - If borrow subtracted: 0x12345678 - 1 - 0xffffffff = 0x12345678 (= 0x12345678)

8570. no statement value (unlike function expression)

8571. Three digit octal escape, digits validated.

8572. **comment:** * Addition operator is different from other arithmetic * operations in that it also provides string concatenation. * Hence it is implemented separately. * * There is a fast path for number addition. Other cases go * through potentially multiple coercions as described in the * E5 specification. It may be possible to reduce the number * of coercions, but this must be done carefully to preserve * the exact semantics. * * E5 Section 11.6.1. * * Custom types also have special behavior implemented here.

label: code-design

8573. Quite approximate but should be useful for little and big endian.

8574. For X <op>= Y we need to evaluate the pre-op * value of X before evaluating the RHS: the RHS * can change X, but when we do <op> we must use * the pre-op value.

8575. Count is incremented before DUK__DRAGON4_OUTPUT_PREINC() call * on purpose, which is taken into account by the macro.

8576. NUL also comes here. Comparison order matters, 0x20 * is most common whitespace.

8577. retrval for success path

8578. * Wrapping duk_safe_call() will mangle the stack, just return stack top

8579. * Node.js Buffer.prototype: toJSON()

8580. We'll need to interrupt early so recompute the init * counter to reflect the number of bytecode instructions * executed so that step counts for e.g. debugger rate * limiting are accurate.

8581. 'fileName'

8582. * Convert duk_compiler_func to a function template and add it * to the parent function table.

8583. parse args starting from "next temp"

8584. This seems like a good overall approach. Fast path for ASCII in 4 byte * blocks.

8585. * Mark-and-sweep garbage collection.

8586. **comment:** * Object compaction (emergency only). * * Object compaction is a separate step after sweeping, as there is * more free memory for it to work with. Also, currently compaction * may insert new objects into the heap allocated list and the string * table which we don't want to do during a sweep (the reachability * flags of such objects would be incorrect). The objects inserted * are currently: * * - a temporary duk_hbuffer for a new properties allocation * - if array part is abandoned, string keys are interned * * The object insertions go to the front of the list, so they do not * cause an infinite loop (they are not compacted).

label: code-design

8587. **comment:** unnecessary div for last byte

label: code-design

8588. buffer values are coerced first to string here

8589. sanity limit for function pointer size

8590. [sep ToObject(this) len sep str0 ... str(count-1)]

8591. DUK_TOK_YIELD

8592. **comment:** XXX: is valstack top best place for argument?

label: code-design

8593. reset bytecode buffer but keep current size; pass 2 will * require same amount or more.

8594. duk_hobject_run_finalizer() sets

8595. [source template closure]

8596. Successful return: restore jmpbuf and return to caller.

8597. negative -> no atom matched on previous round

8598. [... name]

8599. Note: may be NULL if first call

8600. unresolvable, no stack changes

8601. **comment:** The DataView .buffer property is ordinarily set to the argument * which is an ArrayBuffer. We accept any duk_hbufferobject as * an argument and .buffer will be set to the argument regardless * of what it is. This may be a bit confusing if the argument * is e.g. a DataView or another TypedArray view. * * XXX: Copy .buffer property from a DataView/TypedArray argument? * Create a fresh ArrayBuffer for Duktape.Buffer and Node.js Buffer * arguments? See: test-bug-dataview-buffer-prop.js.

label: code-design

8602. DUK_USE_DATE_NOW_TIME

8603. XXX: valstack handling is awkward. Add a valstack helper which * avoids dup():ing; valstack_copy(src, dst)?

8604. On entry, item at idx_func is a bound, non-lightweight function, * but we don't rely on that below.

8605. * Reference counting implementation.

8606. use strict equality instead of non-strict equality

8607. This assert depends on the input parts representing time inside * the Ecmascript range.

8608. controls for minimum array part growth

8609. -> [... key]

8610. * Returns non-zero if a key and/or value was enumerated, and: * * [enum] -> [key] (get_value == 0) * [enum] -> [key value] (get_value == 1) * * Returns zero without pushing anything on the stack otherwise.

8611. allow basic ASCII whitespace

8612. Heap allocated: return heap pointer which is NOT useful * for the caller, except for debugging.

8613. activation is a direct eval call

8614. current lex_env of the activation (created for catcher)

8615. **comment:** Figure out all active breakpoints. A breakpoint is * considered active if the current function's fileName * matches the breakpoint's fileName, AND there is no * inner function that has matching line numbers * (otherwise a breakpoint would be triggered both * inside and outside of the inner function which would * be confusing). Example: * * function foo() { * print('foo'); * function bar() { <-. breakpoints in these * print('bar'); | lines should not affect * } <- foo() execution * bar(); * } * * We need a few things that are only available when * debugger support is enabled: (1) a line range for * each function, and (2) access to the function template to access the inner functions (and their * line ranges). * * It's important to have a narrow match for active * breakpoints so that we don't enter checked execution * when that's not necessary. For instance, if we're * running inside a certain function and there's * breakpoint outside in (after the call site), we * don't want to slow down execution of the function.

label: code-design

8616. start byte offset of token in lexer input

8617. * Note: lj.value1 is 'value', lj.value2 is 'resume'. * This differs from YIELD.

8618. curr_token = get/set name

8619. Initial T' has been checked and eaten by caller.

8620. Thread may have changed, e.g. YIELD converted to THROW.

8621. Lookup to encode one byte directly into 2 characters: * * def genhextab(bswap): * for i in xrange(256): * t = chr(i).encode('hex') * if bswap: * t = t[1] + t[0] * print('0x' + t.encode('hex') + 'U') * print('big endian'); genhextab(False) * print('little endian'); genhextab(True)

8622. * 64-bit arithmetic * * There are some platforms/compilers where 64-bit types are available * but don't work correctly. Test for known cases.

8623. be paranoid for 32-bit time values (even avoiding negative ones)

8624. * Node.js Buffer.byteLength()

8625. emit instruction; could return PC but that's not needed in the majority * of cases.

8626. Fast write calls which assume you control the spare beforehand. * Multibyte write variants exist and use a temporary write pointer * because byte writes alias with anything: with a stored pointer * explicit pointer load/stores get generated (e.g. gcc -Os).

8627. All the flags fit in 16 bits, so will fit into duk_bool_t.

8628. flags: "g"

8629. For cross-checking during development: ensure dispatch count * matches cumulative interrupt counter init value sums.

8630. valstack should not need changes

8631. end of _Varmap

8632. **comment:** Certain boxed types are required to go through * automatic unboxing. Rely on internal value being * sane (to avoid infinite recursion).

label: code-design

8633. [-0x80000000,0x7fffffff]

8634. DUK_HOBJECT_FLAG_EXOTIC_STRINGOBJ varies

8635. 'Arguments'

8636. [key getter this] -> [key retval]

8637. Retry allocation after mark-and-sweep for this * many times. A single mark-and-sweep round is * not guaranteed to free all unreferenced memory * because of finalization (in fact, ANY number of * rounds is strictly not enough).

8638. only functions can have arguments

8639. E5 Section 9.3.1

8640. * GETVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is GetValue'd] * 8.7.1 GetValue (V) * 8.12.1 [[GetOwnProperty]] (P) * 8.12.2 [[GetProperty]] (P) * 8.12.3 [[Get]] (P) * * If 'throw' is true, always leaves two values on top of stack: [val this]. * * If 'throw' is false, returns 0 if identifier cannot be resolved, and the * stack will be unaffected in this case. If identifier is resolved, returns * 1 and leaves [val this] on top of stack. * * Note: the 'strict' flag of a reference returned by GetIdentifierReference * is ignored by GetValue. Hence we don't take a 'strict' parameter. * * The 'throw' flag is needed for implementing 'typeof' for an unreferenced * identifier. An unreferenced identifier in other contexts generates a * ReferenceError.

8641. **comment:** XXX: top of stack must contain value, which helper doesn't touch, * rework to use tv_val directly?

label: code-design

8642. * Executor interrupt handling * * The handler is called whenever the interrupt countdown reaches zero * (or below). The handler must perform whatever checks are activated, * e.g. check for cumulative step count to impose an execution step * limit or check for breakpoints or other debugger interaction. * * When the actions are done, the handler must reinvoke the interrupt * init and counter values. The 'init' value must indicate how many * bytecode instructions are executed before the next interrupt. The * counter must interface with the bytecode executor loop. Concretely, * the new init value is normally one higher than the new counter value. * For instance, to execute exactly one bytecode instruction the init * value is set to 1 and the counter to 0. If an error is thrown by the * interrupt handler, the counters are set to the same value (e.g. both * to 0 to cause an interrupt when the next bytecode instruction is about * to be executed after error handling). * * Maintaining the init/counter value properly is important for accurate * behavior. For instance, executor step limit needs a cumulative step * count which is simply computed as a sum of 'init' values. This must * work accurately even when single stepping.

8643. 0xd0...0xdf

8644. tv1 is -below- valstack_bottom

8645. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT32

8646. [obj Properties]

8647. * First attempt

8648. not found

8649. Loop check.

8650. XXX: just assert non-NULL values here and make caller arguments * do the defaulting to the default implementations (smaller code)?

8651. reserved words: additional future reserved words in strict mode

8652. * String table resize check. * * Note: this may silently (and safely) fail if GC is caused by an * allocation call in stringtable resize_hash(). Resize_hash() * will prevent a recursive call to itself by setting the * DUK_MS_FLAG_NO_STRINGTABLE_RESIZE in heap->mark_and_sweep_base_flags.

8653. Sync and NULL early.

8654. 12 to 21

8655. **comment:** XXX: we should never shrink here; when we error out later, we'd * need to potentially grow the value stack in error unwind which could * cause another error.

label: code-design

8656. -> pushes fixed buffer

8657. **comment:** XXX: improve check; check against nregs, not against top

label: code-design

8658. byte sizes will match

8659. max # of values initialized in one MPUTARR set

8660. not reached

8661. -> [... this]

8662. * Local defines and forward declarations.

8663. **comment:** prev value must be unused, no decref

label: code-design

8664. Calling duk_debugger_cooperate() while Duktape is being * called into is not supported. This is not a 100% check * but prevents any damage in most cases.

8665. Note: careful with indices like '-x'; if 'x' is zero, it refers to bottom

8666. x >= y -> not (x < y)

8667. yes, must set array length explicitly

8668. * Post resize assertions.

8669. Note: for dstlen=0, dst may be NULL

8670. !DUK_USE_TRACEBACKS

8671. -> [func funcname env funcname]

8672. Constants for duk_lexer_ctx.buf.

8673. Difference is in 100ns units, convert to milliseconds w/o fractions
8674. * "IdentifierPart" production check.
8675. prev exists and is not a letter
8676. Note: no recursion issue, we can trust 'map' to behave
8677. if fails, e_size will be zero = not an issue, except performance-wise
8678. **comment:** XXX: use a NULL error handler for the finalizer call?
 label: code-design
8679. **comment:** workaround: max nbits = 24 now
 label: code-design
8680. DUK_USE_HEX_FASTPATH
8681. DUK_UTIL_H_INCLUDED
8682. 'Infinity'
8683. We know _Varmap only has own properties so walk property * table directly. We also know _Varmap is dense and all * values are numbers; assert for these. GC and finalizers * shouldn't affect _Varmap so side effects should be fine.
8684. debugger state, only relevant when attached
8685. highest value of temp_reg (temp_max - 1 is highest used reg)
8686. * Init compiler and lexer contexts
8687. Input values are signed 48-bit so we can detect overflow * reliably from high bits or just a comparison.
8688. pre/post opcode values have constraints,
8689. allocated heap objects
8690. re-read to avoid spill / fetch
8691. LRU: move our entry to first
8692. argument/function declaration shadows 'arguments'
8693. Proxy target
8694. unconditional block
8695. **comment:** XXX: Using a string return value forces a string intern which is * not always necessary. As a rough performance measure, hex encode * time for tests/perf/test-hex-encode.js dropped from ~35s to ~15s * without string coercion. Change to returning a buffer and let the * caller coerce to string if necessary?
 label: code-design
8696. * Node.js Buffer.prototype.copy()
8697. DUK_VERBOSE_ERRORS
8698. skip '}'
8699. XXX: additional conditions when to close variables? we don't want to do it * unless the environment may have "escaped" (referenced in a function closure). * With delayed environments, the existence is probably good enough of a check.
8700. When debugger is enabled, we need to recheck the activation * status after returning. This is now handled by call handling * and heap->dbg_force_restart.
8701. 'byteLength'
8702. * Parsing implementation. * * JSON lexer is now separate from duk_lexer.c because there are numerous * small differences making it difficult to share the lexer. *
 * The parser here works with raw bytes directly; this works because all * JSON delimiters are ASCII characters. Invalid xUTF-8 encoded values * inside strings will be passed on without normalization; this is not a * compliance concern because compliant inputs will always be valid * CESU-8 encodings.
8703. * Detect iteration statements; if encountered, establish an * empty label.
8704. skip ')'
8705. * Poppers
8706. DUK_HOBJECT_H_INCLUDED
8707. always at least the header
8708. A -> target register * B -> bytecode (also contains flags) * C -> escaped source
8709. Fast path, decode as is.
8710. * "/" followed by something in regexp mode. See E5 Section 7.8.5. * * RegExp parsing is a bit complex. First, the regexp body is delimited * by forward slashes, but the body may also contain forward slashes as * part of an escape sequence or inside a character class (delimited by * square brackets). A mini state machine is used to implement these. * * Further, an early (parse time) error must be thrown if the regexp * would cause a run-time error when used in the expression new RegExp(...). * Parsing here simply extracts the (candidate) regexp, and also accepts * invalid regular expressions (which are delimited properly). The caller * (compiler) must perform final validation and regexp compilation. * * RegExp first char may not be '/' (single line comment) or '*' (multi- * line comment). These have already been checked above, so there is no * need below for special handling of the first regexp character as in * the E5 productions. * * About unicode escapes within regexp literals: * * E5 Section 7.8.5 grammar does NOT accept \uHHHH escapes. * However, Section 6 states that regexps accept the escapes, * see paragraph starting with "In string literals...". * The regexp grammar, which sees the decoded regexp literal * (after lexical parsing) DOES have a \uHHHH unicode escape. * So, for instance: * * \u1234/* * should first be parsed by the lexical grammar as: * * '\u' RegularExpressionBackslashSequence * '1' RegularExpressionNonTerminator * '2' RegularExpressionNonTerminator * '3' RegularExpressionNonTerminator * '4' RegularExpressionNonTerminator * * and the escape itself is then parsed by the regexp engine. * This is the current implementation. * * Minor spec inconsistency: * * E5 Section 7.8.5 RegularExpressionBackslashSequence is: * * \ RegularExpressionNonTerminator * * while Section A.1 RegularExpressionBackslashSequence is: * * \ NonTerminator * * The latter is not normative and a typo. *
8711. 'break'
8712. in yielder's context
8713. * E5 Section 15.4.5.1, steps 3.k - 3.n. The order at the end combines * the error case 3.l.iii and the success case 3.m-3.n. * * Note: 'length' is always in entries part, so no array abandon issues for * 'writable' update.
8714. * Avoid a GC if GC is already running. This can happen at a late * stage in a GC when we try to e.g. resize the stringtable * or compact objects.
8715. Reject attempt to change virtual properties: not part of the * standard algorithm, applies currently to e.g. virtual index * properties of buffer objects (which are virtual but writable). * (Cannot "force" modification of a virtual property.)
8716. MakeDay
8717. Push 'this' binding, check that it is a Date object; then push the * internal time value. At the end, stack is: [... this timeval]. * Returns the time value. Local time adjustment is done if requested.
8718. Cannot use duk_to_string() on the buffer because it is usually * larger than 'len'. Also, 'buf' is usually a stack buffer.
8719. non-leap year: sunday, monday, ...
8720. activation has tail called one or more times
8721. extended types: custom encoding
8722. reg
8723. -> [... enum key val]
8724. EOF
8725. XXX: optimize value stack operation
8726. * Proxy object handling
8727. * Then decode the builtins init data (see genbuiltins.py) to * init objects
8728. mark as complex (capture handling)
8729. string is a reserved word (non-strict)
8730. s <- b^(1-e) * 2
8731. * Object inspection commands: GetHeapObjInfo, GetObjPropDesc, * GetObjPropDescRange
8732. * Unreachable object about to be swept. Finalize target refcounts * (objects which the unreachable object points to) without doing * refzero processing. Recursive decrefs are also prevented when * refzero processing is disabled. * * Value cannot be a finalizable object, as they have been made * temporarily reachable for this round.
8733. Map DUK_FLX_xxx to byte size.
8734. Note that we currently parse -bytes-, not codepoints. * All non-ASCII extended UTF-8 will encode to bytes >= 0x80, * so they'll simply pass through (valid UTF-8 or not).

8735. * Get/require
8736. key, getter and setter, now reachable through object
8737. **comment:** * Finalize objects in the finalization work list. Finalized * objects are queued back to heap_allocated with FINALIZED set. ** Since finalizers may cause arbitrary side effects, they are * prevented during string table and object property allocation * resizing using the DUK_MS_FLAG_NO_FINALIZERS flag in * heap->mark_and_sweep_base_flags. In this case the objects * remain in the finalization work list after mark-and-sweep * exits and they may be finalized on the next pass. ** Finalization currently happens inside "MARKANDSWEET_RUNNING" * protection (no mark-and-sweep may be triggered by the * finalizers). As a side effect: ** 1) an out-of-memory error inside a finalizer will not * cause a mark-and-sweep and may cause the finalizer * to fail unnecessarily ** 2) any temporary objects whose refcount decreases to zero * during finalization will not be put into refzero_list; * they can only be collected by another mark-and-sweep ** This is not optimal, but since the sweep for this phase has * already happened, this is probably good enough for now.
- label:** code-design
8738. * A Dragon4 number-to-string variant, based on: ** Guy L. Steele Jr., Jon L. White: "How to Print Floating-Point Numbers * Accurately" ** Robert G. Burger, R. Kent Dybvig: "Printing Floating-Point Numbers * Quickly and Accurately" ** The current algorithm is based on Figure 1 of the Burger-Dybvig paper, * i.e. the base implementation without logarithm estimation speedups * (these would increase code footprint considerably). Fixed-format output * does not follow the suggestions in the paper; instead, we generate an * extra digit and round-with-carry. ** The same algorithm is used for number parsing (with b=10 and B=2) * by generating one extra digit and doing rounding manually. ** See doc/number-conversion.rst for limitations.
8739. holder will be set to the target object, not the actual object * where the property was found (see duk_get_identifier_reference()).
8740. 'Array'
8741. * INITSET/INITGET are only used to initialize object literal keys. * The compiler ensures that there cannot be a previous data property * of the same name. It also ensures that setter and getter can only * be initialized once (or not at all).
8742. extended utf-8 not allowed for URIs
8743. * E5 Section 15.7.2.1 requires that the constructed object * must have the original Number.prototype as its internal * prototype. However, since Number.prototype is non-writable * and non-configurable, this doesn't have to be enforced here: * The default object (bound to 'this') is OK, though we have * to change its class. ** Internal value set to ToNumber(arg) or +0; if no arg given, * ToNumber(undefined) = NaN, so special treatment is needed * (above). String internal value is immutable.
8744. bytecode limits
8745. guaranteed to find an empty slot
8746. Because 'day since epoch' can be negative and is used to compute weekday * using a modulo operation, add this multiple of 7 to avoid negative values * when year is below 1970 epoch. Ecmascript time values are restricted to * +/- 100 million days from epoch, so this adder fits nicely into 32 bits. * Round to a multiple of 7 (= floor(100000000 / 7) * 7) and add margin.
8747. Allow trailing garbage (e.g. treat "123foo" as "123")
8748. **comment:** XXX: Are steps 6 and 7 in E5 Section 15.11.4.4 duplicated by * accident or are they actually needed? The first ToString() * could conceivably return 'undefined'.
- label:** code-design
8749. **comment:** Size sanity check. Should not be necessary because caller is * required to check this, but we don't want to cause a segfault * if the size wraps either in duk_size_t computation or when * storing the size in a 16-bit field.
- label:** code-design
8750. steps 3.h and 3.i
8751. Note: toJSON() is a generic function which works even if 'this' * is not a Date. The sole argument is ignored.
8752. module (argument)
8753. valstack space that suffices for all local calls, including recursion * of other than Duktape calls (getters etc)
8754. current paren level allows 'in' token
8755. **comment:** XXX: fast-int-to-double
- label:** code-design
8756. * ArrayBuffer.isView()
8757. Magically bound variable cannot be an accessor. However, * there may be an accessor property (or a plain property) in * place with magic behavior removed. This happens e.g. when * a magic property is redefined with defineProperty(). * Cannot assert for "not accessor" here.
8758. * We could clear the book-keeping variables for the topmost activation, * but don't do so now.
8759. * Debugging enabled
8760. * E5 Section 7.4, allow SourceCharacter (which is any 16-bit * code point).
8761. **comment:** XXX: duk_set_length
- label:** code-design
8762. **comment:** XXX: which lists should participate? to be finalized?
- label:** code-design
8763. XXXXXX-- -----
8764. * Bitstream decoder.
8765. * The handling here is a bit tricky. If a previous '|' has been processed, * we have a pending split1 and a pending jump (for a previous match). These * need to be back-patched carefully. See docs for a detailed example.
8766. Step 16: update length; note that the final length may be above 32 bit range * (but we checked above that this isn't the case here)
8767. **comment:** XXX: 'internal error' is a bit of a misnomer
- label:** code-design
8768. **comment:** In compatible mode and standard JSON mode, output * something useful for non-BMP characters. This won't * roundtrip but will still be more or less readable and * more useful than an error.
- label:** code-design
8769. A debugger forced interrupt check is not needed here, as * problematic safe calls are not caused by side effects.
8770. Real world behavior for map(): trailing non-existent * elements don't invoke the user callback, but are still * counted towards result 'length'.
8771. Ensure argument name is not a reserved word in current * (final) strictness. Formal argument parsing may not * catch reserved names if strictness changes during * parsing. ** We only need to do this in strict mode because non-strict * keyword are always detected in formal argument parsing.
8772. retval (sub-atom char length) unused, tainted as complex above
8773. no pre-checks now, assume a previous yield() has left things in * tip-top shape (longjmp handler will assert for these).
8774. * Peephole optimizer for finished bytecode. ** Does not remove opcodes; currently only straightens out unconditional * jump chains which are generated by several control structures.
8775. should not happen
8776. Avoid doing an actual write callback with length == 0, * because that's reserved for a write flush.
8777. DUK_TOK_REGEXP
8778. * Peephole optimize JUMP chains.
8779. * Execution timeout check
8780. Recursive value reviver, implements the Walk() algorithm. No C recursion * check is done here because the initial parsing step will already ensure * there is a reasonable limit on C recursion depth and hence object depth.
8781. type to represent a reg/const reference during compilation
8782. **comment:** Indent helper. Calling code relies on js_ctx->recursion_depth also being * directly related to indent depth.
- label:** code-design
8783. DUK_TOK_PUBLIC
8784. E5 Sections 11.8.4, 11.8.5; x >= y --> not (x < y)
8785. 'get'
8786. **comment:** function name (borrowed reference), ends up in _name
- label:** code-design
8787. repl_value
8788. quotes

8789. 'undefined' already on stack top
8790. 'res' is used for "left", and 'tmp' for "right"
8791. def_value
8792. Delete elements required by a smaller length, taking into account * potentially non-configurable elements. Returns non-zero if all * elements could be deleted, and zero if all or some elements could * not be deleted. Also writes final "target length" to 'out_result_len'. * This is the length value that should go into the 'length' property * (must be set by the caller). Never throws an error.
8793. * duk_handle_call_protected() and duk_handle_call_unprotected(): * call into a Duktape/C or an Ecmascript function from any state. * * Input stack (thr): * * [func this arg1 ... argN] * * Output stack (thr): * * [retval] (DUK_EXEC_SUCCESS) * [errobj] (DUK_EXEC_ERROR (normal error), protected call) * * Even when executing a protected call an error may be thrown in rare cases * such as an insane num_stack_args argument. If there is no catchpoint for * such errors, the fatal error handler is called. * * The error handling path should be error free, even for out-of-memory * errors, to ensure safe sandboxing. (As of Duktape 1.4.0 this is not * yet the case, see XXX notes below.)
8794. **comment:** XXX: skip null filename?
 label: code-design
8795. avoid multiply
8796. New require() function for module, updated resolution base
8797. **comment:** * Function pointers * * Printing function pointers is non-portable, so we do that by hex printing * bytes from memory.
 label: code-design
8798. "-Infinity", '-' has been eaten
8799. make absolute
8800. 'act' already set above
8801. DUK_BUFOBJ_UINT8CLAMPEDARRAY
8802. **comment:** XXX: turkish / azeri, lowercase rules
 label: code-design
8803. **comment:** Coerce like a string. This makes sense because addition also treats * buffers like strings.
 label: code-design
8804. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
8805. no comma
8806. strict outer context
8807. jump for 'finally' case or end (if no finally)
8808. * Structs
8809. DUK_L0 -> '\ char * DUK_L1 ... DUK_L5 -> more lookup
8810. from next pc
8811. If the value happens to be 0xFFFFFFFF, it's not a valid array index * but will then match DUK_NO_ARRAY_INDEX.
8812. use 'left' as a temp
8813. JSON
8814. binary arithmetic; DUK_OP_ADD, DUK_OP_EQ, other binary ops
8815. Innermost fast path processes 4 valid base-64 characters at a time * but bails out on whitespace, padding chars ('=') and invalid chars. * Once the slow path segment has been processed, we return to the * inner fast path again. This handles e.g. base64 with newlines * reasonably well because the majority of a line is in the fast path.
8816. Key and value indices are either (-2, -1) or (-1, -2). Given idx_key, * idx_val is always (idx_key ^ 0x01).
8817. * Pointer built-ins
8818. Zero length string is not accepted without quotes
8819. [key] -> []
8820. No activation, no variable access. Could also pretend * we're in the global program context and read stuff off * the global object.
8821. current function being compiled (embedded instead of pointer for more compact access)
8822. Note: src_data may be NULL if input is a zero-size * dynamic buffer.
8823. Note: 'this' may be bound to any value, not just an object
8824. **comment:** XXX: Not needed for now, so not implemented. Note that * function pointers may have different size/layout than * a void pointer.
 label: requirement
8825. out of spec, don't care
8826. **comment:** XXX: idx_val would fit into 16 bits, but using duk_small_uint_t * might not generate better code due to casting.
 label: code-design
8827. used by e.g. duk_regexp_executor.c, string built-ins
8828. **comment:** XXX: could also copy from template, but there's no way to have any * other value here now (used code has no access to the template).
 label: code-design
8829. Share yield longjmp handler.
8830. curr_token slot2 (matches 'lex' slot2_idx)
8831. Error message doesn't matter: the error is ignored anyway.
8832. Have a calling activation, check for direct eval (otherwise * assume indirect eval.
8833. **comment:** Compact but lots of churn.
 label: code-design
8834. Output shuffling: only one output register is realistically possible. * * (Zero would normally be an OK marker value: if the target register * was zero, it would never be shuffled. But with DUK_USE_SHUFFLE_TORTURE * this is no longer true, so use -1 as a marker instead.)
8835. **comment:** actually used
 label: code-design
8836. rule match
8837. * Helper structs
8838. terminating conditions
8839. see below
8840. allow automatic semicolon even without lineterm (compatibility)
8841. Outside any activation -> put to global.
8842. +0 = break, +1 = continue
8843. try locale specific formatter; if it refuses to format the * string, fall back to an ISO 8601 formatted value in local * time.
8844. -> [val arr]
8845. * Byte matching for back-references would be OK in case- * sensitive matching. In case-insensitive matching we need * to canonicalize characters, so back-reference matching needs * to be done with codepoints instead. So, we just decode * everything normally here, too. * * Note: back-reference index which is 0 or higher than * NCapturingParens (= number of capturing parens in the * -entire- regexp) is a compile time error. However, a * backreference referring to a valid capture which has * not matched anything always succeeds! See E5 Section * 15.10.2.9, step 5, sub-step 3.
8846. this is needed for regexp mode
8847. jump to next case clause
8848. pop plain buffer, now reachable through h_bufres
8849. * PUTPROP: Ecmascript property write. * * Unlike Ecmascript primitive which returns nothing, returns 1 to indicate * success and 0 to indicate failure (assuming throw is not set). * * This is an extremely tricky function. Some examples: * * * Currently a decref may trigger a GC, which may compact an object's * property allocation. Consequently, any entry indices (e_idx) will * be potentially invalidated by a decref. * * * Exotic behaviors (strings, arrays, arguments object) require, * among other things: * * - Preprocessing before and postprocessing after an actual property * write. For example, array index write requires pre-checking the * array 'length' property for access control, and may require an * array 'length' update after the actual write has succeeded (but * not if it fails). * * - Deletion of multiple entries, as a result of array 'length' write. * * * Input values are taken as pointers which may point to the valstack. * If valstack is resized because of the

put (this may happen at least * when the array part is abandoned), the pointers can be invalidated. * (We currently make a copy of all of the input values to avoid issues.)

8850. add AD*2^16

8851. Proxy handler

8852. **comment:** * Tailcall handling ** Although the callstack entry is reused, we need to explicitly unwind * the current activation (or simulate an unwind). In particular, the * current activation must be closed, otherwise something like * test-bug-reduce-judofyr.js results. Also catchstack needs be unwound * because there may be non-error-catching label entries in valid tail calls.

label: code-design

8853. **comment:** Should not be required because the code below always sets both high * and low parts, but at least gcc-4.4.5 fails to deduce this correctly * (perhaps because the low part is set (seemingly) conditionally in a * loop), so this is here to avoid the bogus warning.

label: code-design

8854. Impose a maximum string length for now. Restricted artificially to * ensure adding a heap header length won't overflow size_t. The limit * should be synchronized with DUK_HBUFFER_MAX_BYTelen. ** E5.1 makes provisions to support strings longer than 4G characters. * This limit should be eliminated on 64-bit platforms (and increased * closer to maximum support on 32-bit platforms).

8855. relookup exports from module.exports in case it was changed by modSearch

8856. * Indexed read/write helpers (also used from outside this file)

8857. adv = 2 - 1 default OK

8858. **comment:** XXX: the key in 'key in obj' is string coerced before we're called * (which is the required behavior in E5/E5.1/E6) so the key is a string * here already.

label: code-design

8859. **comment:** XXX: side effect handling is quite awkward here

label: code-design

8860. act->func

8861. **comment:** Set stack top within currently allocated range, but don't reallocate. * This is performance critical especially for call handling, so whenever * changing, profile and look at generated code.

label: code-design

8862. **comment:** Neutered checks not necessary here: neutered buffers have * zero 'length' so we'll effectively skip them.

label: code-design

8863. Call setup checks callability.

8864. **comment:** Flags for intermediate value coercions. A flag for using a forced reg * is not needed, the forced_reg argument suffices and generates better * code (it is checked as it is used).

label: code-design

8865. **comment:** Non-ASCII slow path (range-by-range linear comparison), very slow

label: code-design

8866. Contract, either: * - Push value on stack and return 1 * - Don't push anything on stack and return 0

8867. 0: not IdentifierStart or IdentifierPart * 1: IdentifierStart and IdentifierPart * -1: IdentifierPart only

8868. DUK_BUFOBJ_UINT32ARRAY

8869. * Restore 'caller' property for non-strict callee functions.

8870. ($\geq e 0$) AND (not ($= f (\text{expt } b (- p 1))$)) * * be <- (expt b e) == b^e * r <- (* f be 2) == 2 * f * b^e [if b==2 -> f * b^(e+1)] * s <- 2 * m+ <- be == b^e * m- <- be == b^e * k <- 0 * B <- B * low_ok <- round * high_ok <- round

8871. A -> unused (reserved for flags, for consistency with DUK_OP_CALL) * B -> target register and start reg: constructor, arg1, ..., argN * (for DUK_OP_NEWI, 'b' is indirect) * C -> num args (N)

8872. 0x60...0x6f

8873. ≥ 0

8874. mimic semantics for strings

8875. Parse a single statement. * * Creates a label site (with an empty label) automatically for iteration * statements. Also "peels off" any label statements for explicit labels.

8876. DUK_FLD_8BIT

8877. [key]

8878. normal key/value

8879. mm <- b^e

8880. steps 4.a and 4.b are tricky

8881. The external string struct is defined even when the feature is inactive.

8882. fixed arg count

8883. A -> flags * B -> return value reg/const * C -> currently unused

8884. **comment:** Because there are quite many call sites, pack error code (require at most * 8-bit) into a single argument.

label: code-design

8885. Raw helper to extract internal information / statistics about a value. * The return values are version specific and must not expose anything * that would lead to security issues (e.g. exposing compiled function * 'data' buffer might be an issue). Currently only counts and sizes and * such are given so there should not be a security impact.

8886. -> [...] closure template newobj]

8887. XXX: This causes recursion up to inner function depth * which is normally not an issue, e.g. mark-and-sweep uses * a recursion limiter to avoid C stack issues. Avoiding * this would mean some sort of a work list or just refusing * to serialize deep functions.

8888. * Size-optimized pc->line mapping.

8889. return as is

8890. value1 -> resume value, value2 -> resume thread, iserror -> error/normal

8891. function: create binding for func name (function templates only, used for named function expressions)

8892. The low 8 bits map directly to duk_hobject.h DUK_PROPDESC_FLAG_xxx. * The remaining flags are specific to the debugger.

8893. **comment:** XXX: copy from caller?

label: code-design

8894. always for register bindings

8895. **comment:** XXX: optimize: allocate an array part to the necessary size (upwards * estimate) and fill in the values directly into the array part; finally * update 'length'.

label: code-design

8896. heap->dbg_udata: keep

8897. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

8898. IdentifierStart production with Letter, ASCII, and non-BMP excluded

8899. **comment:** * Custom formatter for debug printing, allowing Duktape specific data * structures (such as tagged values and heap objects) to be printed with * a nice format string. Because debug printing should not affect execution * state, formatting here must be independent of execution (see implications * below) and must not allocate memory. ** Custom format tags begin with a '%!' to safely distinguish them from * standard format tags. The following conversions are supported: * * %!T tagged value (duk_tval *) * %!O heap object (duk_heapobj *) * %!I decoded bytecode instruction * %!C bytecode instruction opcode name (arg is long) * * Everything is serialized in a JSON-like manner. The default depth is one * level, internal prototype is not followed, and internal properties are not * serialized. The following modifiers change this behavior: * * @ print pointers * # print binary representations (where applicable) * d deep traversals of own properties (not prototype) * p follow prototype chain (useless without 'd') * i include internal properties (other than prototype) * x hexdump buffers * h heavy formatting ** For instance, the following serializes objects recursively, but does not * follow the prototype chain nor print internal properties: "%!dO". ** Notes: * * Standard snprintf return value semantics seem to vary. This * implementation returns the number of bytes it actually wrote * (excluding the null terminator). If retval == buffer size, * output was truncated (except for corner cases). *** Output format is intentionally different from EcmaScript * formatting requirements, as formatting here serves debugging * of internals. *** Depth checking (and updating) is done in each type printer * separately, to allow them to call each other

freely. *** Some pathological structures might take ages to print (e.g. * self recursion with 100 properties pointing to the object * itself). To guard against these, each printer also checks * whether the output buffer is full; if so, early exit. *** Reference loops are detected using a loop stack.

label: code-design

8900. dummy value used as marker

8901. never here, but fall through

8902. **comment:** * duk_handle_safe_call(): make a "C protected call" within the * current activation. ** The allowed thread states for making a call are the same as for * duk_handle_call_xxx(). * Error handling is similar to duk_handle_call_xxx(); errors may be thrown * (and result in a fatal error) for insane arguments.

label: code-design

8903. must never longjmp

8904. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside object 'a_size'.

8905. Must set 'length' explicitly when using duk_xdef_prop_xxx() to * set the values.

8906. heapdr size and additional allocation size, followed by * type specific stuff (with varying value count)

8907. stack top contains 'true'

8908. callbacks and udata; dbg_read_cb != NULL is used to indicate attached state

8909. \x0ffHandler

8910. Note: string is a terminal heap object, so no depth check here

8911. 0x80-0x8f

8912. * Internal helper for defining an accessor property, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes. This is called * very rarely, so the implementation first sets a value to undefined * and then changes the entry to an accessor (this is to save code space).

8913. Debug print which visualizes the qsort partitioning process.

8914. XXX: this placeholder is not always correct, but use for now. * It will fail in corner cases; see test-dev-func-cons-args.js.

8915. r <- (2 * f) * b'e

8916. unreferenced with some options

8917. always terminates led()

8918. (reference is valid as long activation exists)

8919. don't report __FILE__ / __LINE__ as fileName/lineNumber

8920. **comment:** Note: don't bail out early, we must read all the ranges from * bytecode. Another option is to skip them efficiently after * breaking out of here. Prefer smallest code.

label: code-design

8921. Pause for all step types: step into, step over, step out. * This is the only place explicitly handling a step out.

8922. awkward; we assume there is space for this

8923. Peek ahead in the stream one byte.

8924. * Breakpoint management

8925. * Compact an object

8926. slot B is both a target and a source (used by extraops like DUK_EXTRAOP_INSTOF

8927. alias for above

8928. handle comma and closing brace

8929. shrink case; leave some spare

8930. strictness is not inherited, intentional

8931. only Array.prototype matches

8932. Parse a function-like expression, assuming that 'comp_ctx->curr_func' is * correctly set up. Assumes that curr_token is just after 'function' (or * 'set'/'get' etc).

8933. function: always register bound

8934. **comment:** XXX: fast primitive to set a bunch of values to UNDEFINED

label: code-design

8935. Buffer size needed for duk_bi_date_format_timeval(). * Accurate value is 32 + 1 for NUL termination: * >>> len('+123456-01-23T12:34:56.123+12:34') * 32 *
Include additional space to be safe.

8936. We want to compare the slice/view areas of the arguments. * If either slice/view is invalid (underlying buffer is shorter) * ensure equals() is false, but otherwise the only thing that * matters is to be memory safe.

8937. This seems reasonable overall.

8938. Loop prevention

8939. 'throw'

8940. we're running inside the caller's activation, so no change in call/catch stack or valstack bottom

8941. Assume that the native representation never contains a closing * parenthesis.

8942. varargs marker

8943. entry level reached

8944. Current strictness flag: affects API calls.

8945. Because value stack init policy is 'undefined above top', * we don't need to write, just assert.

8946. '*'

8947. call is from executor, so we know we have a jmpbuf

8948. Yield/resume book-keeping.

8949. finally part will catch

8950. Quite lenient, e.g. allow empty as zero, but don't allow trailing * garbage.

8951. IEEE requires that NaNs compare false

8952. Load inner functions to value stack, but don't yet copy to buffer.

8953. [start end]

8954. **comment:** Try to make do with a stack buffer to avoid allocating a temporary buffer. * This works 99% of the time which is quite nice.

label: code-design

8955. This testcase fails when Emscripten-generated code runs on Firefox. * It's not an issue because the failure should only affect packed * duk_tval representation, which is not used with Emscripten.

8956. **comment:** XXX: keys is an internal object with all keys to be processed * in its (gapless) array part. Because nobody can touch the keys * object, we could iterate its array part directly (keeping in mind * that it can be reallocated).

label: code-design

8957. Return value is like write(), number of bytes written. * The return value matters because of code like: * "off += buf.copy(...)".

8958. Exponent without digits (e.g. "1e" or "1e+"). If trailing garbage is * allowed, ignore exponent part as garbage (= parse as "1", i.e. exp 0).

8959. DUK_TOK_THROW

8960. **comment:** Note: not a typo, "object" is returned for a null value

label: documentation

8961. atom_char_length, atom_start_offset, atom_start_offset reflect the * atom matched on the previous loop. If a quantifier is encountered * on this loop, these are needed to handle the quantifier correctly. * new_atom_char_length etc are for the atom parsed on this round; * they're written to atom_char_length etc at the end of the round.

8962. Original idiom used, minimal code size.

8963. **comment:** 'fun' is quite rarely used, so no local for it

label: code-design

8964. **comment:** XXX: fastint

label: code-design

8965. Note: in integer arithmetic, (x / 4) is same as floor(x / 4) for non-negative * values, but is incorrect for negative ones.

8966. normal

8967. Careful with wrapping (left shifting idx would be unsafe).

8968. **comment:** XXX: for negative input offsets, 'offset' will be a large * positive value so the result here is confusing.
label: code-design

8969. no overflow

8970. "+11:22:0"

8971. borrowed: function being executed; for bound function calls, this is the final, real function, NULL for lightfuncs

8972. **comment:** * Augment error (throw time), unless alloc/double error
label: code-design

8973. -> [... ToObject(this) ToUint32(length) arg[i]]

8974. The specification has quite awkward order of coercion and * checks for toPrecision(). The operations below are a bit * reordered, within constraints of observable side effects.

8975. caller

8976. match followed by capture(s)

8977. return input buffer, converted to a Duktape.Buffer object * if called as a constructor (no change if called as a * function).

8978. We could use a switch-case for the class number but it turns out * a small if-else ladder on class masks is better. The if-ladder * should be in order of relevancy.

8979. * New length is smaller than old length, need to delete properties above * the new length. * * If array part exists, this is straightforward: array entries cannot * be non-configurable so this is guaranteed to work. * * If array part does not exist, array-indexed values are scattered * in the entry part, and some may not be configurable (preventing length * from becoming lower than their index + 1). To handle the algorithm * in E5 Section 15.4.5.1, step 1 correctly, we scan the entire property * set twice.

8980. 'ptr' is evaluated both as LHS and RHS.

8981. * EcmaScript [[Class]]

8982. **comment:** Quick reject of too large or too small exponents. This check * would be incorrect for zero (e.g. "0e1000" is zero, not Infinity) * so zero check must be above.
label: code-design

8983. finalize_list will always be processed completely

8984. **comment:** XXX: exposed duk_debug_read_pointer
label: code-design

8985. -> [... key val replacer holder]

8986. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside bufferobject length.

8987. open scope information, for compiled functions only

8988. Handle empty separator case: it will always match, and always * triggers the check in step 13.c.iii initially. Note that we * must skip to either end of string or start of first codepoint, * skipping over any continuation bytes! * * Don't allow an empty string to match at the end of the input.

8989. temp reg handling

8990. * Parse function body

8991. x <- y - z

8992. 'Pointer'

8993. relookup if changed

8994. **comment:** * Augment an error at creation time with _Tracedata/fileName/lineNumber * and allow a user error handler (if defined) to process/replace the error. * The error to be augmented is at the stack top. * * thr: thread containing the error value * thr_callstack: thread which should be used for generating callstack etc. * c_filename: C __FILE__ related to the error * c_line: C __LINE__ related to the error * noblame_fileline: if true, don't fileName/line as error source, otherwise use traceback * (needed because user code filename/line are reported but internal ones * are not) * * XXX: rename noblame_fileline to flags field; combine it to some existing * field (there are only a few call sites so this may not be worth it).
label: code-design

8995. throw_flag

8996. * Else, must be one of: * - ExpressionStatement, possibly a directive (String) * - LabelledStatement (Identifier followed by ':') * * Expressions beginning with 'function' keyword are covered by a case * above (such expressions are not allowed in standard E5 anyway). * Also expressions starting with '{' are interpreted as block * statements. See E5 Section 12.4. * * Directive detection is tricky; see E5 Section 14.1 on directive * prologue. A directive is an expression statement with a single * string literal and an explicit or automatic semicolon. Escape * characters are significant and no parens etc are allowed: * * 'use strict'; // valid 'use strict' directive * 'use\u0020strict'; // valid directive, not a 'use strict' directive * ('use strict'); // not a valid directive * * The expression is determined to consist of a single string literal * based on duk_expr_nud() and duk_expr_led() call counts. The string literal * of a 'use strict' directive is determined to lack any escapes based * num_escapes count from the lexer. Note that other directives may be * allowed to contain escapes, so a directive with escapes does not * terminate a directive prologue. * * We rely on the fact that the expression parser will not emit any * code for a single token expression. However, it will generate an * intermediate value which we will then successfully ignore. * * A similar approach is used for labels.

8997. The act->prev_caller should only be set if the entry for 'caller' * exists (as it is only set in that case, and the property is not * configurable), but handle all the cases anyway.

8998. break/continue without label

8999. **comment:** XXX: the best typing needs to be validated by perf measurement: * e.g. using a small type which is the cast to a larger duk_idx_t * may be slower than declaring the variable as a duk_idx_t in the * first place.
label: code-design

9000. **comment:** XXX: conversion errors should not propagate outwards. * Perhaps values need to be coerced individually?
label: code-design

9001. XXX: here we need to know if 'left' is left-hand-side compatible. * That information is no longer available from current expr parsing * state; it would need to be carried into the 'left' ivalue or by * some other means.

9002. non-global regexp: lastIndex never updated on match

9003. Note: undefined from Section 11.8.5 always results in false * return (see e.g. Section 11.8.3) - hence special treatment here.

9004. If there's no catch block, rc_varname will be 0 and duk_patch_tryCatch() * will replace the LDCONST with a NOP. For any actual constant (including * constant 0) the DUK__CONST_MARKER flag will be set in rc_varname.

9005. exposed because lexer needs these too

9006. empty match (may happen with empty separator) -> bump and continue

9007. As with all inspection code, we rely on the debug client providing * a valid, non-stale pointer: there's no portable way to safely * validate the pointer here.

9008. Set up curr_pc for opcode dispatch.

9009. [... pattern flags escaped_source]

9010. **comment:** The 'magic' field allows an opaque 16-bit field to be accessed by the * Duktape/C function. This allows, for instance, the same native function * to be used for a set of very similar functions, with the 'magic' field * providing the necessary non-argument flags / values to guide the behavior * of the native function. The value is signed on purpose: it is easier to * convert a signed value to unsigned (simply AND with 0xffff) than vice * versa. * * Note: cannot place nargs/magic into the heaphdr flags, because * duk_hobject takes almost all flags already (and needs the spare).
label: code-design

9011. Get minimum entry part growth for a certain size.

9012. valstack index of start of args (arg1) (relative to entry valstack_bottom)

9013. [ToObject(this) item1 ... itemN arr item(i) item(i)[j]]

9014. thr->heap->lj.value2 is 'thread', will be wiped out at the end

9015. break/continue with label (label cannot be a reserved word, production is 'Identifier'

9016. property access

9017. Coerce all finite parts with ToInteger(). ToInteger() must not * be called for NaN/Infinity because it will convert e.g. NaN to * zero. If ToInteger() has already been called, this has no side * effects and is idempotent. * * Don't read dparts[DUK_DATE_IDX_WEEKDAY]; it will cause Valgrind * issues if the value is uninitialized.

9018. internal property functions

9019. Report thrown value to client coerced to string
9020. **comment:** ArrayBuffer argument is handled specially above; the rest of the * argument variants are handled by shared code below.
 label: code-design
9021. DUK_USE_USER_INITJS
9022. Range limited to [0, 0x7fffffff] range, i.e. range that can be * represented with duk_int32_t. Use this when the method doesn't * handle the full 32-bit unsigned range correctly.
9023. 110x xxxx 10xx xxxx
9024. Sanity limits for stack sizes.
9025. **comment:** Linear scan: more likely because most objects are small. * This is an important fast path. ** XXX: this might be worth inlining for property lookups.
 label: code-design
9026. XXX: will need a force flag if garbage collection is triggered * explicitly during paused state.
9027. Array length is larger than 'asize'. This shouldn't * happen in practice. Bail out just in case.
9028. No net refcount change.
9029. * Assert context is valid: non-NULL pointer, fields look sane. ** This is used by public API call entrypoints to catch invalid 'ctx' pointers * as early as possible; invalid 'ctx' pointers cause very odd and difficult to * diagnose behavior so it's worth checking even when the check is not 100%.
9030. * Setup value stack: clamp to 'nargs', fill up to 'nregs'
9031. This should not happen because DUK_TAG_OBJECT case checks * for this already, but check just in case.
9032. result array is already at the top of stack
9033. args go here as a comma expression in parens
9034. [...] source? filename? &comp_args] (depends on flags)
9035. no extra padding
9036. Start position (inclusive) and end position (exclusive)
9037. tentative, checked later
9038. slot A is a source (default: target)
9039. indexed by recursion_depth
9040. * Fixed buffer helper useful for debugging, requires no allocation * which is critical for debugging.
9041. Order of unwinding is important
9042. * Final sigma context specific rule. This is a rather tricky * rule and this handling is probably not 100% correct now. * The rule is not locale/language specific so it is supported.
9043. tracks maximum initialized index + 1
9044. **comment:** first register that is a temporary (below: variables)
 label: code-design
9045. A -> register of target object * B -> first register of value data (start_index, value1, value2, ..., valueN) * C -> number of key/value pairs (N)
9046. Just skip, leaving zeroes in the result.
9047. [...] arr]
9048. Note: successive characters could be joined into string matches * but this is not trivial (consider e.g. '/xyz+/); see docs for * more discussion.
9049. Coerce an duk_alue to a register or constant; result register may * be a temp or a bound register. ** The duk_alue argument ('x') is converted into a regconst as a * side effect.
9050. bytes in source
9051. no voluntary gc
9052. A -> result reg * B -> object reg * C -> key reg/const
9053. An object may have FINALIZED here if it was finalized by mark-and-sweep * on a previous run and refcount then decreased to zero. We won't run the * finalizer again here.
9054. **comment:** XXX: allow object to be a const, e.g. in 'foo'.toString()? * On the other hand, DUK_REGCONSTP() is slower and generates * more code.
 label: code-design
9055. refcount
9056. regexp res_obj is at index 4
9057. stack[0] = callback fn * stack[1] = initialValue * stack[2] = object (coerced this) * stack[3] = length (not needed, but not popped above) * stack[4] = accumulator
9058. **comment:** Node.js Buffer variable width integer field. We don't really * care about speed here, so aim for shortest algorithm.
 label: code-design
9059. The spec algorithm first does "R = ToString(separator)" before checking * whether separator is undefined. Since this is side effect free, we can * skip the ToString() here.
9060. ']'
9061. NULL with zero length represents an empty string; NULL with higher * length is also now treated like an empty string although it is * a bit dubious. This is unlike duk_push_string() which pushes a * 'null' if the input string is a NULL.
9062. -> [func funcname env funcname func]
9063. true, because v[1] has at least one bit set
9064. bp to use when parsing a top level Expression
9065. Caller will finish the marking process if we hit a recursion limit.
9066. y
9067. **comment:** * E5 Section 7.6: ** IdentifierStart: * UnicodeLetter * \$ * _ * \ UnicodeEscapeSequence ** IdentifierStart production has one multi-character production: ** \ UnicodeEscapeSequence ** The '\ character is -not- matched by this function. Rather, the caller * should decode the escape and then call this function to check whether the * decoded character is acceptable (see discussion in E5 Section 7.6). ** The "UnicodeLetter" alternative of the production allows letters * from various Unicode categories. These can be extracted with the * "src/extract_chars.py" script. ** Because the result has hundreds of Unicode codepoint ranges, matching * for any values >= 0x80 are done using a very slow range-by-range scan * and a packed range format. ** The ASCII portion (codepoints 0x00 ... 0x7f) is fast-patched below because * it matters the most. The ASCII related ranges of IdentifierStart are: ** 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z'] * 0x0024 ['\$'] * 0x005f['_']
 label: code-design
9068. Must be an object, otherwise TypeError (E5.1 Section 8.10.5, step 1).
9069. TypeError if fails
9070. * Entries part
9071. * Delayed env creation check
9072. insert 'undefined' values at idx_rcbase to get the * return values to idx_rebase
9073. xxx -> DUK_HBUFFEROBJECT_ELEM_INT8
9074. guarantees entry_callstack_top - 1 >= 0
9075. no code needs to be emitted, the regs already have values
9076. magic: 0=getter call, 1=Object.getPrototypeOf
9077. PC is unsigned. If caller does PC arithmetic and gets a negative result, * it will map to a large PC which is out of bounds and causes a zero to be * returned.
9078. * Debug connection write primitives
9079. **comment:** XXX: join these ops (multiply-accumulate), but only if * code footprint decreases.
 label: code-design
9080. module.name for .name, default to last component if * not present.
9081. * Exposed debug macros: debugging enabled
9082. Object.defineProperty() calls [[DefineOwnProperty]] with Throw=true
9083. 0xe0-0xef
9084. compact the prototype
9085. XXX: default priority for infix operators is duk__expr_lbp(tok) -> get it here?
9086. read header

9087. prop index in 'hash part', < 0 if not there
9088. This is based on Rhino EquivalentYear() algorithm: *
<https://github.com/mozilla/rhino/blob/f99cc11d616f0cdda2c42bde72b3484df6182947/src/org.mozilla/javascript/NativeDate.java>
9089. **comment:** Clamping to zero makes the API more robust to calling code * calculation errors.
label: code-design
9090. Surround with parentheses like in JX, ensures NULL pointer * is distinguishable from null value ("(null)" vs "null").
9091. Attempt to write 'stack', 'fileName', 'lineNumber' works as if * user code called Object.defineProperty() to create an overriding * own property. This allows user code to overwrite .fileName etc * intuitively as e.g. "err.fileName = 'dummy'" as one might expect. * See <https://github.com/svaarala/duktape/issues/387>.
9092. object is a native function (duk_hnativefunction)
9093. [...] error func fileName lineNumber]
9094. 21 bits
9095. **comment:** XXX: thread selection for mark-and-sweep is currently a hack. * If we don't have a thread, the entire mark-and-sweep is now * skipped (although we could just skip finalizations).
label: code-design
9096. embed: duk_hstring ptr
9097. enumeration
9098. * Free memory
9099. Maximum value check ensures 'nbytes' won't wrap below.
9100. Use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to an existing one).
9101. currently, even for Array.prototype
9102. unsigned intentionally
9103. DUK_TOK_SNEQ
9104. only outer_lex_env matters, as functions always get a new * variable declaration environment.
9105. No need to assert, buffer size maximum is 0xffff.
9106. default is explicit index read/write copy
9107. not enumerable
9108. * Exposed data
9109. avoid problems if p == h->prototype
9110. DUK_USE_PROVIDE_DEFAULT_ALLOC_FUNCTIONS
9111. Output shuffle needed after main operation
9112. For native calls must be NULL so we don't sync back
9113. **comment:** XXX: very basic optimization -> duk_get_prop_stridx_top
label: code-design
9114. omitted
9115. 'toGMTString'
9116. * Object related * * Note: seal() and freeze() are accessible through Ecmascript bindings, * and are not exposed through the API.
9117. * ISO 8601 subset parser.
9118. Internal keys are prefixed with 0xFF in the stringtable * (which makes them invalid UTF-8 on purpose).
9119. without explicit non-BMP support, assume non-BMP characters * are always accepted as identifier characters.
9120. [res]
9121. dash is already 0
9122. Object flag. There are currently 26 flag bits available. Make sure * this stays in sync with debugger object inspection code.
9123. This may only happen if built-ins are being "torn down". * This behavior is out of specification scope.
9124. Not possible because array object 'length' is present * from its creation and cannot be deleted, and is thus * caught as an existing property above.
9125. Note: strictness is not inherited from the current Duktape/C * context. Otherwise it would not be possible to compile * non-strict code inside a Duktape/C activation (which is * always strict now). See tests/api/test-eval-strictness.c * for discussion.
9126. A -> flags * B -> base register for call (base -> func, base+1 -> this, base+2 -> arg1 ... base+2+N-1 -> argN) * (for DUK_OP_CALLI, 'b' is indirect) * C -> nargs
9127. * Wrapper for jmp_buf. * * This is used because jmp_buf is an array type for backward compatibility. * Wrapping jmp_buf in a struct makes pointer references, sizeof, etc, * behave more intuitively. * * http://en.wikipedia.org/wiki/Setjmp.h#Member_types
9128. * This is a bit tricky to implement portably. The result depends * on the timestamp (specifically, DST depends on the timestamp). * If e.g. UNIX APIs are used, they'll have portability issues with * very small and very large years. * * Current approach: * * - Stay within portable UNIX limits by using equivalent year mapping. * Avoid year 1970 and 2038 as some conversions start to fail, at * least on some platforms. Avoiding 1970 means that there are * currently DST discrepancies for 1970. * * - Create a UTC and local time breakdowns from 't'. Then create * a time_t using gmtime() and localtime() and compute the time * difference between the two. * * Equivalent year mapping (E5 Section 15.9.1.8): * * If the host environment provides functionality for determining * daylight saving time, the implementation of ECMAScript is free * to map the year in question to an equivalent year (same * leap-year-ness and same starting week day for the year) for which * the host environment provides daylight saving time information. * The only restriction is that all equivalent years should produce * the same result. * * This approach is quite reasonable but not entirely correct, e.g. * the specification also states (E5 Section 15.9.1.8): * * The implementation of ECMAScript should not try to determine * whether the exact time was subject to daylight saving time, but * just whether daylight saving time would have been in effect if * the _current daylight saving time algorithm_ had been used at the * time. This avoids complications such as taking into account the * years that the locale observed daylight saving time year round. * * Since we rely on the platform APIs for conversions between local * time and UTC, we can't guarantee the above. Rather, if the platform * has historical DST rules they will be applied. This seems to be the * general preferred direction in Ecmascript standardization (or at least * implementations) anyway, and even the equivalent year mapping should * be disabled if the platform is known to handle DST properly for the * full Ecmascript range. * * The following has useful discussion and links: * * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
9129. keep val_highest
9130. **comment:** XXX: limit to quoted strings only, to save keys from being cluttered?
label: code-design
9131. DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT
9132. can't get value, may be accessor
9133. Relookup and initialize dispatch loop variables. Debugger check.
9134. * Longjmp state, contains the information needed to perform a longjmp. * Longjmp related values are written to value1, value2, and iserror.
9135. leave result on stack top
9136. Resolved, normalized absolute module ID
9137. marker; not actual tagged value
9138. re-lookup curr char on first round
9139. [...] formals arguments map mappedNames]
9140. borrowed: full duk_tval for function being executed; for lightfuncs
9141. Shared handler to minimize parser size. Cause will be * hidden, unfortunately, but we'll have an offset which * is often quite enough.
9142. Skip dvalue.
9143. **comment:** XXX: inefficient; block remove primitive
label: code-design
9144. If duk__value_toplain_raw() allocates a temp, forget it and * restore next temp state.
9145. DUK_HOBJECT_FLAG_CONSTRUCTABLE varies
9146. [...] pattern flags]
9147. custom; implies DUK_HOBJECT_IS_THREAD
9148. [thisArg arg1 ... argN]
9149. for array and object literals
9150. validation of the regexp is caller's responsibility
9151. regnum is sane

9152. * To avoid creating a heavy intermediate value for the list of ranges, * only the start token ('[or '[^') is parsed here. The regexp * compiler parses the ranges itself.

9153. **comment:** XXX: Currently function source code is not stored, as it is not * required by the standard. Source code should not be stored by * default (user should enable it explicitly), and the source should * probably be compressed with a trivial text compressor; average * compression of 20-30% is quite easy to achieve even with a trivial * compressor (RLE + backwards lookup). ** Debugging needs source code to be useful: sometimes input code is * not found in files as it may be generated and then eval()'d, given * by dynamic C code, etc. ** Other issues: * - Need tokenizer indices for start and end to substring * - Always normalize function declaration part? * - If we keep _Formals, only need to store body

label: code-design

9154. Preinc/predc for var-by-name, slow path.

9155. tm_isdst is both an input and an output to mktime(), use 0 to * avoid DST handling in mktime(): * - <https://github.com/svaarala/duktape/issues/406> * - <http://stackoverflow.com/questions/8558919/mktime-and-tm-isdst>

9156. Small variant; roughly 150 bytes smaller than the fast variant.

9157. -> [... closure template env funcname]

9158. * Free the heap. ** Frees heap-related non-heap-tracked allocations such as the * string intern table; then frees the heap allocated objects; * and finally frees the heap structure itself. Reference counts * and GC markers are ignored (and not updated) in this process, * and finalizers won't be called. ** The heap pointer and heap object pointers must not be used * after this call.

9159. rescue no longer supported

9160. res->mark_and_sweep_trigger_counter == 0 -> now causes immediate GC; which is OK

9161. _Pc2line

9162. Sanity checks for string and token defines

9163. number of escapes and line continuations (for directive prologue)

9164. MakeTime

9165. **comment:** XXX: this is possible without resorting to the value stack

label: code-design

9166. ignored

9167. NULL for lightfunc

9168. lookup shorthands (note: assume context variable is named 'lex_ctx')

9169. Other callbacks are optional.

9170. truncated in case pass 3 needed

9171. With constants and inner functions on value stack, we can now * atomically finish the function 'data' buffer, bump refcounts, * etc. ** Here we take advantage of the value stack being just a duk_tval * array: we can just memcpy() the constants as long as we incref * them afterwards.

9172. DUK_USE_NONSTD_FUNC_SOURCE_PROPERTY

9173. Leave 'getter' on stack

9174. Called as a function, pattern has [[Class]] "RegExp" and * flags is undefined -> return object as is.

9175. Q...f

9176. x < y

9177. set exotic behavior only after we're done

9178. **comment:** Default function to write a formatted log line. Writes to stderr, * appending a newline to the log line. ** The argument is a buffer whose visible size contains the log message. * This function should avoid coercing the buffer to a string to avoid * string table traffic.

label: code-design

9179. Current size is about 152 bytes.

9180. **comment:** These base values are never used, but if the compiler doesn't know * that DUK_ERROR() won't return, these are needed to silence warnings. * On the other hand, scan-build will warn about the values not being * used, so add a DUK_UNREF.

label: code-design

9181. hash size relative to entries size: for value X, approx. hash_prime(e_size + e_size / X)

9182. while repl

9183. Note: we don't parse back exponent notation for anything else * than radix 10, so this is not an ambiguous check (e.g. hex * exponent values may have 'e' either as a significand digit * or as an exponent separator). ** If the exponent separator occurs twice, 'e' will be interpreted * as a digit (= 14) and will be rejected as an invalid decimal * digit.

9184. Setup state. Initial ivalue is 'undefined'.

9185. 'object'

9186. If we processed any debug messages breakpoints may have * changed; restart execution to re-check active breakpoints.

9187. 1 1 1 <32 bits>

9188. **comment:** Leading / trailing whitespace is sometimes accepted and * sometimes not. After white space trimming, all valid input * characters are pure ASCII.

label: code-design

9189. patch in range count later

9190. * If tracebacks are disabled, 'fileName' and 'lineNumber' are added * as plain own properties. Since Error.prototype has accessors of * the same name, we need to define own properties directly (cannot * just use e.g. duk_put_prop_stridx). Existing properties are not * overwritten in case they already exist.

9191. 0: toString

9192. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small.

label: code-design

9193. UTF-8 encoded bytes escaped as %xx%xx%xx... -> 3 * nbytes. * Codepoint range is restricted so this is a slightly too large * but doesn't matter.

9194. lowest mantissa for this exponent

9195. * Traceback handling when tracebacks disabled. ** The fileName / lineNumber stubs are now necessary because built-in * data will include the accessor properties in Error.prototype. If those * are removed for builds without tracebacks, these can also be removed. * 'stack' should still be present and produce a ToString() equivalent: * this is useful for user code which prints a stacktrace and expects to * see something useful. A normal stacktrace also begins with a ToString() * of the error so this makes sense.

9196. all we need to know

9197. ASSIGNMENT EXPRESSION

9198. IdentityEscape

9199. Must decrease recursion depth before returning.

9200. note: mixing len into seed improves hashing when skipping

9201. For join(), nargs is 1. For toLocaleString(), nargs is 0 and * setting the top essentially pushes an undefined to the stack, * thus defaulting to a comma separator.

9202. Return with final function pushed on stack top.

9203. Note: 'e' and 'E' are also accepted here.

9204. %c, passed concretely as int

9205. thread resumed another thread (active but not running)

9206. or a non-catching entry

9207. next char

9208. -> [... key val]

9209. add F

9210. **comment:** return a specific NaN (although not strictly necessary)

label: code-design

9211. * Variant 1

9212. global regexp: lastIndex updated on match

9213. **comment:** pre-check how many atom copies we're willing to make (atom_copies not needed below)

label: requirement

9214. Current size (not counting a dynamic buffer's "spare").

9215. +2 = catcher value, catcher lj_type

9216. parsing in "directive prologue", recognize directives
9217. non-strict: non-deletable, writable
9218. DUK_USE_64BIT_OPS
9219. simulate alloc failure on every realloc (except when mark-and-sweep is running)
9220. 7
9221. **comment:** alloc temp just in case, to update max temp
label: code-design
9222. * An array must have a 'length' property (E5 Section 15.4.5.2). * The special array behavior flag must only be enabled once the * length property has been added. *
* The internal property must be a number (and preferably a * fastint if fastint support is enabled).
9223. * Create a hstring and insert into the heap. The created object * is directly garbage collectable with reference count zero. * * The caller must place the interned string into the stringtable * immediately (without chance of a longjmp); otherwise the string * is lost.
9224. Note: this is correct even for default clause statements: * they participate in 'fall-through' behavior even if the * default clause is in the middle.
9225. exposed, used by e.g. duk.bi.date.c
9226. * Parsing of ints and floats
9227. flags: "gi"
9228. For TypedArrays 'undefined' return value is specified * by ES6 (matches V8).
9229. * Return target object
9230. side effects
9231. [... result]
9232. only objects have finalizers
9233. interpret e.g. '0x' and '0xg' as a NaN (= parse error)
9234. **comment:** * Get and call the finalizer. All of this must be wrapped * in a protected call, because even getting the finalizer * may trigger an error (getter may throw one, for instance).
label: code-design
9235. **comment:** * String/JSON conversions * * Human readable conversions are now basically ISO 8601 with a space * (instead of 'T') as the date/time separator. This is a good baseline * and is platform independent. * * A shared native helper to provide many conversions. Magic value contains * a set of flags. The helper provides: * * toString() * toDateString() * toTimeString() * toLocaleString() * toLocaleDateString() * toLocaleTimeString() * toUTCString() * toISOString() * * Notes: * * - Date.prototype.toGMTString() and Date.prototype.toUTCString() are * required to be the same Ecmascript function object (!), so it is * omitted from here. * * - Date.prototype.toUTCString(): E5.1 specification does not require a * specific format, but result should be human readable. The * specification suggests using ISO 8601 format with a space (instead of 'T') separator if a more human readable format is not available. * * - Date.prototype.toISOString(): unlike other conversion functions, * toISOString() requires a RangeError for invalid date values.
label: code-design
9236. * Thread support.
9237. bytecode start offset of the atom parsed in this loop * (allows quantifiers to copy the atom bytecode)
9238. temp copy, write back for next loop
9239. char loop
9240. basic platform types
9241. we currently assume virtual properties are not configurable (as none of them are)
9242. Stage 1: find highest preventing non-configurable entry (if any). * When forcing, ignore non-configurability.
9243. **comment:** XXX: keep property attributes or tweak them here? * Properties will now be non-configurable even when they're * normally configurable for the global object.
label: code-design
9244. address
9245. * Property already exists. Steps 5-6 detect whether any changes need * to be made.
9246. may be NULL
9247. * Parse Ecmascript source InputElementDiv or InputElementRegExp * (E5 Section 7), skipping whitespace, comments, and line terminators. * * Possible results are: * (1) a token * (2) a line terminator (skipped) * (3) a comment (skipped) * (4) EOF * * White space is automatically skipped from the current position (but * not after the input element). If input has already ended, returns * DUK_TOK_EOF indefinitely. If a parse error occurs, uses an DUK_ERROR() * macro call (and hence a longjmp through current heap longjmp context). * Comments and line terminator tokens are automatically skipped. * * The input element being matched is determined by regexp_mode; if set, * parses a InputElementRegExp, otherwise a InputElementDiv. The * difference between these are handling of productions starting with a * forward slash. * * If strict_mode is set, recognizes additional future reserved words * specific to strict mode, and refuses to parse octal literals. * * The matching strategy below is to (currently) use a six character * lookup window to quickly determine which production is the -longest- * matching one, and then parse that. The top-level if-else clauses * match the first character, and the code blocks for each clause * handle -all- alternatives for that first character. Ecmascript * specification uses the "longest match wins" semantics, so the order * of the if-clauses matters. * * Misc notes: * * * Ecmascript numeric literals do not accept a sign character. * Consequently e.g. "-1.0" is parsed as two tokens: a negative * sign and a positive numeric literal. The compiler performs * the negation during compilation, so this has no adverse impact. * * * There is no token for "undefined": it is just a value available * from the global object (or simply established by doing a reference * to an undefined value). * * * Some contexts want Identifier tokens, which are IdentifierNames * excluding reserved words, while some contexts want IdentifierNames * directly. In the latter case e.g. "while" is interpreted as an * identifier name, not a DUK_TOK WHILE token. The solution here is * to provide both token types: DUK_TOK WHILE goes to 't' while * DUK_TOK_IDENTIFIER goes to 't_noes', and 'slot1' always contains * the identifier / keyword name. * * * Directive prologue needs to identify string literals such as * "use strict" and 'use strict', which are sensitive to line * continuations and escape sequences. For instance, "use\0020strict" * is a valid directive but is distinct from "use strict". The solution * here is to decode escapes while tokenizing, but to keep track of the * number of escapes. Directive detection can then check that the * number of escapes is zero. * * * Multi-line comments with one or more internal LineTerminator are * treated like a line terminator to comply with automatic semicolon * insertion.
9248. IEEE double is approximately 16 decimal digits; print a couple extra
9249. * Resizing and hash behavior
9250. String sanitizer which escapes ASCII control characters and a few other * ASCII characters, passes Unicode as is, and replaces invalid UTF-8 with * question marks. No errors are thrown for any input string, except in out * of memory situations.
9251. **comment:** Try to optimize X <op>= Y for reg-bound * variables. Detect side-effect free RHS * narrowly by seeing whether it emits code. * If not, rewind the code emitter and overwrite * the unnecessary temp reg load.
label: code-design
9252. [obj key undefined]
9253. **comment:** XXX: can be optimized for smaller footprint esp. on 32-bit environments
label: code-design
9254. * isArray()
9255. DUK_USE_HOBJECT_HASH_PART
9256. trailing elisions?
9257. * Simple commands
9258. XXX: The TypeError is currently not applied to bound * functions because the 'strict' flag is not copied by * bind(). This may or may not be correct, the specification * only refers to the value being a "strict mode Function * object" which is ambiguous.
9259. [... global val] -> [... global]
9260. length in codepoints (must be E5 compatible)
9261. The smallest fastint is no longer 48-bit when * negated. Positive zero becomes negative zero * (cannot be represented) when negated.
9262. Decode helper. Return zero on error.
9263. XXX: specify array size, as we know it
9264. **comment:** Numbers are normalized to big (network) endian. We can * (but are not required) to use integer dvalues when there's * no loss of precision. * * XXX:
share check with other code; this check is slow but * reliable and doesn't require careful exponent/mantissa * mask tricks as in the fastint downgrade code.
label: code-design

9265. * Use static helpers which can work with math.h functions matching * the following signatures. This is not portable if any of these math * functions is actually a macro.
9266. 1111 0xxx; 4 bytes
9267. start array index of current MPUTARR set
9268. cp == -1 (EOF) never matches and causes return value 0
9269. **comment:** XXX: the handling of character range detection is a bit convoluted. * Try to simplify and make smaller.
 label: code-design
9270. DUK_OP_CALL flags in A
9271. * Shared helper for non-bound func lookup. * * Returns duk_hobject * to the final non-bound function (NULL for lightfunc).
9272. Must be >= 0 and multiple of element size.
9273. A stubbed built-in is useful for e.g. compilation torture testing with BCC.
9274. Decode a plain string consisting entirely of identifier characters. * Used to parse plain keys (e.g. "foo: 123").
9275. note that we can't reliably pop anything here
9276. [... put_value]
9277. 5
9278. ref.value and ref.this_binding invalidated here
9279. nret
9280. object is a compiled function (duk_hcompiledfunction)
9281. Push a new closure on the stack. * * Note: if fun_temp has NEWENV, i.e. a new lexical and variable declaration * is created when the function is called, only outer_lex_env matters * (outer_var_env is ignored and may or may not be same as outer_lex_env).
9282. Need a short reg/const, does not have to be a mutable temp.
9283. delete Duktape.modLoaded[resolved_id]
9284. keep current valstack_top
9285. Disabled until fixed, see above.
9286. 'message'
9287. Just transfer the refcount from act->prev_caller to tv_caller, * so no need for a refcount update. This is the expected case.
9288. Suggested step-by-step method from documentation of RtTimeToSecondsSince1970: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928(v=vs.85).aspx)
9289. * Migrate array to start of entries if requested. * * Note: from an enumeration perspective the order of entry keys matters. * Array keys should appear wherever they appeared before the array abandon * operation.
9290. 0x70...0x7f
9291. Maximum iteration count for computing UTC-to-local time offset when * creating an Ecmascript time value from local parts.
9292. **comment:** XXX: depend on available temps?
 label: code-design
9293. default case
9294. %u; only 16 bits are guaranteed
9295. MultiplicativeExpression
9296. Trim white space (= allow leading and trailing whitespace)
9297. * Call user provided module search function and build the wrapped * module source code (if necessary). The module search function * can be used to implement pure Ecmascript, pure C, and mixed * Ecmascript/C modules. * * The module search function can operate on the exports table directly * (e.g. DLL code can register values to it). It can also return a * string which is interpreted as module source code (if a non-string * is returned the module is assumed to be a pure C one). If a module * cannot be found, an error must be thrown by the user callback. * * Because Duktape.modLoaded[] already contains the module being * loaded, circular references for C modules should also work * (although expected to be quite rare).
9298. does not modify tv_x
9299. 26: setSeconds
9300. patch pending jump and split
9301. traceback depth doesn't take into account the filename/line * special handling above (intentional)
9302. DUK_USE_ERRTHROW || DUK_USE_ERRCREATE
9303. only needed by debugger for now
9304. Compute day number of the first day of a given year.
9305. ignore_loop
9306. * Begin
9307. Output #1: resolved absolute name
9308. and even number
9309. * DELPROP: Ecmascript property deletion.
9310. normal: implicit leading 1-bit
9311. may throw an error
9312. * Encoding constants, must match genbuiltins.py
9313. 'RegExp'
9314. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize?
 label: code-design
9315. <<
9316. **comment:** XXX: double evaluation of DUK_HCOMPILEDFUNCTION_GET_DATA()
 label: code-design
9317. ensure full memcmp() fits in while
9318. unreachable
9319. push operation normalizes NaNs
9320. getter/setter
9321. x == 0x00 (EOF) causes syntax_error
9322. XXX: range assert
9323. **comment:** XXX: move masks to js_ctx? they don't change during one * fast path invocation.
 label: code-design
9324. flags: "i"
9325. * Init stringtable: probe variant
9326. Catches both doubles and cases where only one argument is a fastint
9327. key coercion (unless already coerced above)
9328. **comment:** XXX: There's quite a bit of overlap with buffer creation handling in * duk_bi_buffer.c. Look for overlap and refactor.
 label: code-design
9329. Limit checks for bytecode byte size and line number.
9330. The necessary #includes are in place in duk_config.h.
9331. **comment:** XXX: this could be optimized
 label: code-design
9332. duk_handle_return() is guaranteed never to throw, except * for potential out-of-memory situations which will then * propagate out of the executor longjmp handler.
9333. **comment:** * XXX: if duk_handle_call() took values through indices, this could be * made much more sensible. However, duk_handle_call() needs to fudge * the 'this' and 'func' values to handle bound function chains, which * is now done "in-place", so this is not a trivial change.
 label: code-design
9334. Number serialization has a significant impact relative to * other fast path code, so careful fast path for fastints.

9335. * Compiler intermediate values ** Intermediate values describe either plain values (e.g. strings or * numbers) or binary operations which have not yet been coerced into * either a left-hand-side or right-hand-side role (e.g. object property).

9336. **comment:** Silence a few global unused warnings here.
label: code-design

9337. -> [... arr num]

9338. tzoffset seconds are dropped; 16 bits suffice for * time offset in minutes

9339. thread has yielded

9340. Lightfuncs are always strict.

9341. '\xffArgs'

9342. **comment:** XXX: for array instances we could take a shortcut here and assume * Array.prototype doesn't contain an array index property.
label: code-design

9343. PropertyName -> IdentifierName | StringLiteral | NumericLiteral

9344. -1 = top callstack entry, callstack[callstack_top - 1] * -callstack_top = bottom callstack entry, callstack[0]

9345. Further state-dependent pre-checks

9346. E5 Section 8.12.8

9347. is_decl

9348. at least one activation, ours

9349. s <- b[^](-e) * 2

9350. use p_src_base from now on

9351. 'taint' result as complex

9352. no issues with memcmp() zero size, even if broken

9353. * Disjunction struct: result of parsing a disjunction

9354. * Emit initializers in sets of maximum max_init_pairs keys. * Setter/getter is handled separately and terminates the * current set of initializer values. Corner cases such as * single value initializers do not have special handling now.

9355. * E5 Section 7.3: CR LF is detected as a single line terminator for * line numbers. Here we also detect it as a single line terminator * token.

9356. We don't need to sync back thr->ptr_curr_pc here because * the bytecode executor always has a setjmp catchpoint which * does that before errors propagate to here.

9357. Previous entry was inside visited[], nothing to do.

9358. **comment:** XXX: optimize temp reg use
label: code-design

9359.toFixed()

9360. Detect zero special case.

9361. Number and (minimum) size of bigints in the nc_ctx structure.

9362. We come here for actual aborts (like encountering .toJSON()) * but also for recursion/loop errors. Bufwriter size can be * kept because we'll probably need at least as much as we've * allocated so far.

9363. * Preliminaries: trim, sign, Infinity check ** We rely on the interned string having a NUL terminator, which will * cause a parse failure wherever it is encountered. As a result, we * don't need separate pointer checks. * * There is no special parsing for 'NaN' in the specification although * 'Infinity' (with an optional sign) is allowed in some contexts. * Some contexts allow plus/minus sign, while others only allow the * minus sign (like JSON.parse()). * * Automatic hex number detection (leading '0x' or '0X') and octal * number detection (leading '0' followed by at least one octal digit) * is done here too.

9364. duk_unicode_idp_m_ids_noabmp[]

9365. '(?:)'

9366. * Process replacer/proplist (2nd argument to JSON.stringify)

9367. may be a string constant

9368. return total chars written excluding terminator

9369. [(builtin objects) name]

9370. new_free / size <= 1 / DIV <=> new_free <= size / DIV

9371. This shouldn't happen; call sites should avoid looking up * _Finalizer "through" a Proxy, but ignore if we come here * with a Proxy to avoid finalizer re-entry.

9372. [... holder name val enum obj_key]

9373. * Create a new function object based on a "template function" which contains * compiled bytecode, constants, etc, but lacks a lexical environment. * * EcmaScript requires that each created closure is a separate object, with * its own set of editable properties. However, structured property values * (such as the formal arguments list and the variable map) are shared. * Also the bytecode, constants, and inner functions are shared. * * See E5 Section 13.2 for detailed requirements on the function objects; * there are no similar requirements for function "templates" which are an * implementation dependent internal feature. Also see function-objects.rst * for a discussion on the function instance properties provided by this * implementation. * * Notes: * * * Order of internal properties should match frequency of use, since the * properties will be linearly scanned on lookup (functions usually don't * have enough properties to warrant a hash part). * * * The created closure is independent of its template; they do share the * same 'data' buffer object, but the template object itself can be freed * even if the closure object remains reachable.

9374. E5 Section 11.13.1 (and others) step 4 never matches for prop writes -> no check

9375. exit() afterwards to satisfy "noretturn"

9376. string compare is the default (a bit oddly)

9377. %xx%xx...%xx', p points to char after first '%'

9378. **comment:** XXX: need a toplain_ignore() which will only coerce a value to a temp * register if it might have a side effect. Side-effect free values do not * need to be coerced.
label: code-design

9379. 2⁴4 -> 1/16 = 6.25% spare

9380. * reduce(), reduceRight()

9381. * Macros for accessing size fields

9382. * Resolve module identifier into canonical absolute form.

9383. leap year: sunday, monday, ...

9384. even after a detach and possible reattach

9385. tval

9386. **comment:** integer mixed endian not really used now
label: code-design

9387. only advance if not tainted

9388. tail insert: don't disturb head in case refzero is running

9389. indexOf: NaN should cause pos to be zero. * lastIndexOf: NaN should cause pos to be +Infinity * (and later be clamped to len).

9390. 0.000...

9391. leave stack unbalanced on purpose

9392. [... func this <bound args> arg1 ... argN]

9393. Bitfield for each DUK_HBUFFEROBJECT_ELEM_xxx indicating which element types * are compatible with a blind byte copy for the TypedArray set() method (also * used for TypedArray constructor). Array index is target buffer elem type, * bitfield indicates compatible source types. The types must have same byte * size and they must be coercion compatible.

9394. **comment:** Run fake finalizer. Avoid creating unnecessary garbage.
label: code-design

9395. * Table for hex encoding bytes

9396. [... buf func] -> [... func]

9397. build result as: (r << 32) + s: start with (BD + E + F)

9398. We only get here when doing non-standard JSON encoding

9399. Without heap pointer compression duk_hbuffer_dynamic and duk_hbuffer_external * have the same layout so checking for fixed vs. dynamic (or external) is enough.

9400. [target] -> [enum]

9401. Handle both full and partial slice (as long as covered).

9402. envrec: (declarative) record is closed

9403. **comment:** Log frontend shared helper, magic value indicates log level. Provides * frontend functions: trace(), debug(), info(), warn(), error(), fatal(). * This needs to have small footprint, reasonable performance, minimal * memory churn, etc.
label: code-design

9404. XXX: bump preventcount by one for the duration of this call?

9405. for

9406. 15: getUTCDay

9407. need_bytes may be zero

9408. register binding lookup is based on varmap (even in first pass)

9409. prototype: the only internal property lifted outside 'e' as it is so central

9410. 3 entries actually needed below

9411. end of input and last char has been processed

9412. [start length str]

9413. array

9414. fast paths for space and tab

9415. Significand ('f') padding.

9416. **comment:** not needed
label: requirement

9417. Regardless of whether property is found in entry or array part, * it may have arguments exotic behavior (array indices may reside * in entry part for abandoned / non-existent array parts).

9418. add a new pending match jump for latest finished alternative

9419. exclusive

9420. DUK_USE_BUILTIN_INITJS

9421. * If selftests enabled, run them as early as possible

9422. ivalue/ispec helpers

9423. * Determine the effective 'this' binding and coerce the current value * on the valstack to the effective one (in-place, at idx_this). * * The current this value in the valstack (at idx_this) represents either: * - the caller's requested 'this' binding; or * - a 'this' binding accumulated from the bound function chain * * The final 'this' binding for the target function may still be * different, and is determined as described in E5 Section 10.4.3. * * For global and eval code (E5 Sections 10.4.1 and 10.4.2), we assume * that the caller has provided the correct 'this' binding explicitly * when calling, i.e.: * * - global code: this=global object * - direct eval: this=copy from eval() caller's this binding * - other eval: this=global object * * Note: this function may cause a recursive function call with arbitrary * side effects, because ToObject() may be called.

9424. idx_this = idx_func + 1

9425. **comment:** XXX: for real world code, could just ignore array inheritance * and only look at array own properties.
label: code-design

9426. not needed, as we exit right away

9427. XXX: return type should probably be duk_size_t, or explicit checks are needed for * maximum size.

9428. relookup, may have changed

9429. leading digit + fractions

9430. Select appropriate escape format automatically, and set 'tmp' to a * value encoding both the escape format character and the nybble count: * * (nybble_count << 16) | (escape_char1) | (escape_char2)

9431. tc1 = true, tc2 = true

9432. See comments below on MakeTime why these are volatile.

9433. -> [val obj val]

9434. Store lexer position, restoring if quantifier is invalid.

9435. no need to compact since we already did that in duk_abandon_array_checked() * (regardless of whether an array part existed or not).

9436. marker; not actual tagged type

9437. **comment:** XXX: expensive check (also shared elsewhere - so add a shared internal API call?)
label: code-design

9438. For NaN/inf, the return value doesn't matter.

9439. curr char

9440. [regexp string]

9441. **comment:** NULL not needed here
label: code-design

9442. duk_pcall_prop() may itself throw an error, but we're content * in catching the obvious errors (like toLogString() throwing an * error).

9443. overflow, more than 32 bits -> not an array index

9444. special result value handling

9445. [... Logger clog res]

9446. [... put_value varname]

9447. intermediate join to avoid valstack overflow

9448. ptr before mark-and-sweep

9449. end-of-input breaks

9450. reserve a jumps slot after instr before target spilling, used for NEXTENUM

9451. a label site has been emitted by duk_parse_stmt() automatically * (it will also emit the ENDLABEL).

9452. * Variable access

9453. **comment:** function: function must not be tail called
label: code-design

9454. third arg: absolute index (to entire valstack) of idx_bottom of new activation

9455. **comment:** * HASPROP variant used internally. * * This primitive must never throw an error, callers rely on this. * In particular, don't throw an error for prototype loops; instead, * pretend like the property doesn't exist if a prototype sanity limit * is reached. * * Does not implement proxy behavior: if applied to a proxy object, * returns key existence on the proxy object itself.
label: code-design

9456. 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.

9457. !DUK_USE_HSTRING_CLEN

9458. DUK_ERR_EVAL: no macros needed

9459. DUK_USE_PREFER_SIZE

9460. Inref copies, keep originals.

9461. object is a buffer object (duk_hbufferobject) (always exotic)

9462. recheck that the property still exists

9463. jump from true part to end

9464. **comment:** XXX: fastint fast path would be very useful here
label: code-design

9465. XXX: happens e.g. when evaluating: String(Buffer.prototype).

9466. 'name'

9467. eat identifier

9468. * Init stringtable: fixed variant
9469. Copy values through direct validated reads and writes.
9470. break matches always
9471. * "length" maps to number of formals (E5 Section 13.2) for function * declarations/expressions (non-bound functions). Note that 'nargs' * is NOT necessarily equal to the number of arguments.
9472. -1 for opcode
9473. * Regexp instance check, bytecode check, input coercion. ** See E5 Section 15.10.6.
9474. **comment:** hash part size or 0 if unused
label: code-design
9475. **comment:** XXX: this duplicates functionality in duk_regex.c where a similar loop is * required anyway. We could use that BUT we need to update the regexp compiler * 'nranges' too. Work this out a bit more cleanly to save space.
label: code-design
9476. helper to insert a (non-string) heap object into heap allocated list
9477. * Internal API calls which have (stack and other) semantics similar * to the public API.
9478. retval is directly usable
9479. **comment:** * Error throwing helpers ** The goal is to provide verbose and configurable error messages. Call * sites should be clean in source code and compile to a small footprint. * Small footprint is also useful for performance because small cold paths * reduce code cache pressure. Adding macros here only makes sense if there * are enough call sites to get concrete benefits.
label: code-design
9480. digit count
9481. * Hobject property set/get functionality. ** This is very central functionality for size, performance, and compliance. * It is also rather intricate; see hobject-algorithms.rst for discussion on * the algorithms and memory-management.rst for discussion on refcounts and * side effect issues. ** Notes: ** - It might be tempting to assert "refcount nonzero" for objects * being operated on, but that's not always correct: objects with * a zero refcount may be operated on by the refcount implementation * (finalization) for instance. Hence, no refcount assertions are made. ** - Many operations (memory allocation, identifier operations, etc) * may cause arbitrary side effects (e.g. through GC and finalization). * These side effects may invalidate duk_tval pointers which point to * areas subject to reallocation (like value stack). Heap objects * themselves have stable pointers. Holding heap object pointers or * duk_tval copies is not problematic with respect to side effects; * care must be taken when holding and using argument duk_tval pointers. ** - If a finalizer is executed, it may operate on the same object * we're currently dealing with. For instance, the finalizer might * delete a certain property which has already been looked up and * confirmed to exist. Ideally finalizers would be disabled if GC * happens during property access. At the moment property table realloc * disables finalizers, and all DECREFs may cause arbitrary changes so * handle DECREF carefully. ** - The order of operations for a DECREF matters. When DECREF is executed, * the entire object graph must be consistent; note that a refzero may * lead to a mark-and-sweep through a refcount finalizer.
9482. **comment:** * Compact the closure, in most cases no properties will be added later. * Also, without this the closures end up having unused property slots * (e.g. in Duktape 0.9.0, 8 slots would be allocated and only 7 used). * A better future solution would be to allocate the closure directly * to correct size (and setup the properties directly without going * through the API).
label: code-design
9483. $2^3 \rightarrow 1/8 = 12.5\%$ min growth
9484. * Reference count updates ** Note: careful manipulation of refcounts. The top is * not updated yet, so all the activations are reachable * for mark-and-sweep (which may be triggered by decref). * However, the pointers are NULL so this is not an issue.
9485. 'hex'
9486. borrowed label name
9487. * String table algorithm: closed hashing with a probe sequence ** This is the default algorithm and works fine for environments with * minimal memory constraints.
9488. Longjmp state is kept clean in success path
9489. Longjmp callers are required to sync-and-null thr->ptr_curr_pc * before longjmp.
9490. Return invalid index; if caller uses this without checking * in another API call, the index won't map to a valid stack * entry.
9491. **comment:** XXX: faster internal way to get this
label: code-design
9492. old value is number: no refcount
9493. Using classification has smaller footprint than direct comparison.
9494. DUK_USE_ROM_GLOBAL_CLONE || DUK_USE_ROM_GLOBAL_INHERIT
9495. leading byte of match string
9496. **comment:** * Formal argument list ** We don't check for prohibited names or for duplicate argument * names here, because we don't yet know whether the function will * be strict. Function body parsing handles this retroactively.
label: code-design
9497. canonicalized by compiler
9498. [source template closure this]
9499. initial index
9500. 5 heap flags
9501. DUK_TOK_FUNCTION
9502. exponent for 'f'
9503. * Duktape includes (other than duk_features.h) ** The header files expect to be included in an order which satisfies header * dependencies correctly (the headers themselves don't include any other * includes). Forward declarations are used to break circular struct/typedef * dependencies.
9504. always 1 arg
9505. helpers
9506. ignore result
9507. DUK_JMPBUF_H_INCLUDED
9508. 'static'
9509. * sort() ** Currently qsort with random pivot. This is now really, really slow, * because there is no fast path for array parts. ** Signed indices are used because qsort() leaves and degenerate cases * may use a negative offset.
9510. extended types: compatible encoding
9511. **comment:** not allowed in strict mode, regardless of whether resolves; * in non-strict mode DELVAR handles both non-resolving and * resolving cases (the specification description is a bit confusing).
label: code-design
9512. valstack limit caller has check, prevents wrapping
9513. Note that byte order doesn't affect this test: all bytes in * 'test' will be 0xFF for two's complement.
9514. Note: number has no explicit tag (in 8-byte representation)
9515. * Calendar helpers ** Some helpers are used for getters and can operate on normalized values * which can be represented with 32-bit signed integers. Other helpers are * needed by setters and operate on un-normalized double values, must watch * out for non-finite numbers etc.
9516. Unlike year, the other parts fit into 16 bits so %d format * is portable.
9517. reg/const for case value
9518. Equivalent year mapping, used to avoid DST trouble when platform * may fail to provide reasonable DST answers for dates outside the * ordinary range (e.g. 1970-2038). An equivalent year has the same * leap-year-ness as the original year and begins on the same weekday * (Jan 1). ** The year 2038 is avoided because there seem to be problems with it * on some platforms. The year 1970 is also avoided as there were * practical problems with it; an equivalent year is used for it too, * which breaks some DST computations for 1970 right now, see e.g. * test-bi-date-tzoffset-brute-fi.js.
9519. signed integer encoding needed to work with UTF-8
9520. Here val would be potentially invalid if we didn't make * a value copy at the caller.
9521. '&'
9522. jump is inserted here (variant 3)

9523. * Closing environment records. ** The environment record MUST be closed with the thread where its activation * is. In other words (if 'env' is open): ** - 'thr' must match _env.thread * - 'func' must match _env.callee * - 'regbase' must match _env.regbase ** These are not looked up from the env to minimize code size. *
* XXX: should access the own properties directly instead of using the API

9524. resume: [... initial_func undefined(= this) resume_value]

9525. ignore reclimit, not constructor

9526. '\xffCallee'

9527. is normalized

9528. Variant for writing duk_tvals so that any heap allocated values are * written out as tagged heap pointers.

9529. Coerce like boolean

9530. mark-and-sweep marking reached a recursion limit and must use multi-pass marking

9531. [(builtin objects)]

9532. **comment:** XXX: tv_func is not actually needed
label: requirement

9533. * Source filename (or equivalent), for identifying thrown errors.

9534. '_proto_'

9535. **comment:** XXX: this is probably a useful shared helper: for a * duk_hbufferobject, get a validated buffer pointer/length.
label: code-design

9536. parent env is the prototype

9537. get varenv for varname (callee's declarative lexical environment)

9538. Use a new environment and there's an 'arguments' object. * We need to initialize it right now.

9539. The timevalue must be in valid Ecmascript range, but since a local * time offset can be applied, we need to allow a +/- 24h leeway to * the value. In other words, although the UTC time is within the * Ecmascript range, the local part values can be just outside of it.

9540. Note: can be safely scanned as bytes (undecoded)

9541. =====

9542. No activation matches, use undefined for both .fileName and * .lineNumber (matches what we do with a _Tracedata based * no-match lookup).

9543. also goes into flags

9544. UnaryExpression

9545. **comment:** XXX: return something more useful, so that caller can throw?
label: code-design

9546. Tailcalls are handled by back-patching the TAILCALL flag to the * already emitted instruction later (in return statement parser). * Since A and C have a special meaning here, they cannot be "shuffled".

9547. all virtual properties are non-configurable and non-writable

9548. [obj trap_result res_arr]

9549. **comment:** * The string conversion here incorporates all the necessary Ecmascript * semantics without attempting to be generic. nc_ctx->digits contains * nc_ctx->count digits (>= 1), with the topmost digit's 'position' * indicated by nc_ctx->k as follows: ** digits="123" count=3 k=0 --> 0.123 * digits="123" count=3 k=1 --> 1.23 * digits="123" count=3 k=5 --> 12300 * digits="123" count=3 k=-1 --> 0.0123 ** Note that the identifier names used for format selection are different * in Burger-Dybvig paper and Ecmascript specification (quite confusingly * so, because e.g. 'k' has a totally different meaning in each). See * documentation for discussion. ** Ecmascript doesn't specify any specific behavior for format selection * (e.g. when to use exponent notation) for non-base-10 numbers. ** The bigint space in the context is reused for string output, as there * is more than enough space for that (>1kB at the moment), and we avoid * allocating even more stack.
label: code-design

9550. depth check is done when printing an actual type

9551. Preinc/predic for object properties.

9552. * duk_re_range_callback for generating character class ranges. ** When ignoreCase is false, the range is simply emitted as is. * We don't, for instance, eliminate duplicates or overlapping * ranges in a character class. ** When ignoreCase is true, the range needs to be normalized through * canonicalization. Unfortunately a canonicalized version of a * continuous range is not necessarily continuous (e.g. [x-{}] is * continuous but [X-{}] is not). The current algorithm creates the * canonicalized range(s) space efficiently at the cost of compile * time execution time (see doc/regexp.rst for discussion). ** Note that the ctx->nranges is a context-wide temporary value * (this is OK because there cannot be multiple character classes * being parsed simultaneously).

9553. init function state: init valstack allocations

9554. **comment:** Currently about 7*152 = 1064 bytes. The space for these * duk_bignums is used also as a temporary buffer for generating * the final string. This is a bit awkward; a union would be * more correct.
label: code-design

9555. Check statement type based on the first token type. ** Note: expression parsing helpers expect 'curr_tok' to * contain the first token of the expression upon entry.

9556. **comment:** * The 'in' operator requires an object as its right hand side, * throwing a TypeError unconditionally if this is not the case. ** However, lightfuncs need to behave like fully fledged objects * here to be maximally transparent, so we need to handle them * here.
label: code-design

9557. * Number-to-string conversion. The semantics of these is very tightly * bound with the Ecmascript semantics required for call sites.

9558. **comment:** SCANBUILD: with suitable dmin/dmax limits 'd' is unused
label: code-design

9559. 0xff => 255 - 256 = -1; 0x80 => 128 - 256 = -128

9560. DUK_USE_PARANOI_ERROR

9561. Does not assume that jump_pc contains a DUK_OP_JUMP previously; this is intentional * to allow e.g. an INVALID opcode be overwritten with a JUMP (label management uses this).

9562. A 32-bit unsigned integer formats to at most 32 digits (the * worst case happens with radix == 2). Output the digits backwards, * and use a memmove() to get them in the right place.

9563. Without ROM objects "needs refcount update" == is heap allocated.

9564. **comment:** Note: 'act' is dangerous here because it may get invalidated at many * points, so we re-lookup it multiple times.
label: code-design

9565. **comment:** XXX: Add a flag to reject an attempt to re-attach? Otherwise * the detached callback may immediately reattach.
label: code-design

9566. side effects -> don't use tv_x, tv_y after

9567. [... enum]

9568. if gap is empty, behave as if not given at all

9569. When JX/JC not in use, the type mask above will avoid this case if needed.

9570. DUK_TOK_LT

9571. no value

9572. * HASVAR: check identifier binding from a given environment record * without traversing its parents. ** This primitive is not exposed to user code as such, but is used * internally for e.g. declaration binding instantiation. ** See E5 Sections: * 10.2.1.1 HasBinding(N) * 10.2.1.2.1 HasBinding(N) ** Note: strictness has no bearing on this check. Hence we don't take * a 'strict' parameter.

9573. hash probe sequence

9574. * Make the C call

9575. DUK_USE_NONSTD_FUNC_STMT

9576. has access to 'this' binding

9577. **comment:** Create a temporary enumerator to get the (non-duplicated) key list; * the enumerator state is initialized without being needed, but that * has little impact.
label: code-design

9578. **comment:** XXX: optimize for string inputs: no need to coerce to a buffer * which makes a copy of the input.
label: code-design

9579. initial size guess
9580. Buffer writes are often integers.
9581. env[funcname] = closure
9582. Safe to call multiple times.
9583. Default variants. Selection depends on speed/size preference. * Concretely: with gcc 4.8.1 -Os x64 the difference in final binary * is about +1kB for _FAST variants.
9584. formatted result limited
9585. [] -> [res]
9586. may be NaN
9587. 'undefined', artifact of lookup
9588. assume array part is comprehensive (contains all array indexed elements * or none of them); hence no need to check the entries part here.
9589. lJ value1: value
9590. pushes function template
9591. Identifier, i.e. don't allow reserved words
9592. **comment:** Parse an inner function, adding the function template to the current function's * function table. Return a function number to be used by the outer function. * * Avoiding O(depth^2) inner function parsing is handled here. On the first pass, * compile and register the function normally into the 'funcs' array, also recording * a lexer point (offset/line) to the closing brace of the function. On the second * pass, skip the function and return the same 'fnum' as on the first pass by using * a running counter. * * An unfortunate side effect of this is that when parsing the inner function, almost * nothing is known of the outer function, i.e. the inner function's scope. We don't * need that information at the moment, but it would allow some optimizations if it * were used.
label: code-design
9593. [arg toLogString]
9594. out_token->lineterm set by caller
9595. note: any entries above the catchstack top are garbage and not zeroed
9596. Error code also packs a tracedata related flag.
9597. enumerator must have no keys deleted
9598. Stack size increases or stays the same.
9599. known to be number; in fact an integer
9600. avail_bytes += need_bytes
9601. **comment:** XXX: this illustrates that a C catchpoint implemented using duk_safe_call() * is a bit heavy at the moment. The wrapper compiles to ~180 bytes on x64. * Alternatives would be nice.
label: code-design
9602. temp variable to pass constants and flags to shared code
9603. char length of the atom parsed in this loop
9604. -> [... fn x y]
9605. **comment:** ToBoolean() does not require any operations with side effects so * we can do it efficiently. For footprint it would be better to use * duk_js_tobool()and then push+replace to the result slot.
label: code-design
9606. -> [...]
9607. topmost activation idx_retval is considered garbage, no need to init
9608. * Delayed activation environment record initialization (for functions * with NEWENV). * * The non-delayed initialization is handled by duk_handle_call().
9609. ref.value, ref.this.binding invalidated here by getprop call
9610. * Field accessor macros
9611. [... buf]
9612. idx_base and idx_base+1 get completion value and type
9613. integer number in range
9614. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor). * * Using duk_put_prop() works incorrectly with '__proto__' * if the own property with that name has been deleted. This * does not happen normally, but a clever reviver can trigger * that, see complex reviver case in: test-bug-json-parse-__proto__.js.
label: code-design
9615. Force interrupt right away if we're paused or in "checked mode". * Step out is handled by callstack unwind.
9616. check stack before interning (avoid hanging temp)
9617. The extra (+4) is tight.
9618. Send version identification and flush right afterwards. Note that * we must write raw, unframed bytes here.
9619. DUK_TOK_STATIC
9620. No pointer compression because pointer is potentially outside of * Duktape heap.
9621. 'null'
9622. pc2line
9623. always normalized
9624. JUMP L1 omitted
9625. **comment:** writable, not configurable
label: code-design
9626. * Init/assert flags, copying them where appropriate. Some flags * (like NEWENV) are processed separately below.
9627. big endian
9628. entry_top + 3
9629. must match bytecode defines now; build autogenerate?
9630. Although NAMEBINDING is not directly needed for using * function instances, it's needed by bytecode dump/load * so copy it too.
9631. ''
9632. * Automatically generated by genbuiltins.py, do not edit!
9633. * Debug connection read primitives
9634. Note: when parsing a formal list in non-strict context, e.g. * "implements" is parsed as an identifier. When the function is * later detected to be strict, the argument list must be rechecked * against a larger set of reserved words (that of strict mode). * This is handled by duk_parse_func_body(). Here we recognize * whatever tokens are considered reserved in current strictness * (which is not always enough).
9635. **comment:** XXX: optimize with more direct internal access
label: code-design
9636. __proto__ setter returns 'undefined' on success unlike the * setPrototypeOf() call which returns the target object.
9637. 0x00-0x0f
9638. * String comparison (E5 Section 11.8.5, step 4), which * needs to compare codepoint by codepoint. * * However, UTF-8 allows us to use strcmp directly: the shared * prefix will be encoded identically (UTF-8 has unique encoding) * and the first differing character can be compared with a simple * unsigned byte comparison (which strcmp does). * * This will not work properly for non-xutf-8 strings, but this * is not an issue for compliance.
9639. DUK_TOK_BOR_EQ
9640. Argument must be a string, e.g. a buffer is not allowed.
9641. Note: val is a stable duk_tval pointer. The caller makes * a value copy into its stack frame, so 'tv_val' is not subject * to side effects here.
9642. number of formal arguments
9643. register, constant, or value
9644. Note: intentionally allow leading zeroes here, as the * actual parser will check for them.
9645. Impose a maximum buffer length for now. Restricted artificially to * ensure resize computations or adding a heap header length won't * overflow size_t and that a signed duk_int_t can hold a buffer * length. The limit should be synchronized with DUK_HSTRING_MAX_BYTELEN.
9646. h_code: held in bw_code

9647. * Forward declarations.
9648. heap pointer comparison suffices
9649. Note: 'count' is currently not adjusted by rounding (i.e. the * digits are not "chopped off". That shouldn't matter because * the digit position (absolute or relative) is passed on to the * convert-and-push function.
9650. **comment:** The variable environment for magic variable bindings needs to be * given by the caller and recorded in the arguments object. * * See E5 Section 10.6, the creation of setters/getters. * * The variable environment also provides access to the callee, so * an explicit (internal) callee property is not needed.
label: code-design
9651. All components default to 0 except day-of-month which defaults * to 1. However, because our internal day-of-month is zero-based, * it also defaults to zero here.
9652. Backtrack.
9653. **comment:** XXX: code duplication
label: code-design
9654. arg count
9655. **comment:** Switch to caller's setjmp() catcher so that if an error occurs * during error handling, it is always propagated outwards instead * of causing an infinite loop in our own handler.
label: code-design
9656. [... number] -> [... string]
9657. 'data' (and everything in it) is reachable through h_res now
9658. Fast path.
9659. called as a normal function: return new Date().toString()
9660. -----XX
9661. must be computed after realloc
9662. 'join'
9663. Get the value represented by an duk_ispec to a register or constant. * The caller can control the result by indicating whether or not: * * (1) a constant is allowed (sometimes the caller needs the result to * be in a register) * * (2) a temporary register is required (usually when caller requires * the register to be safely mutable; normally either a bound * register or a temporary register are both OK) * * (3) a forced register target needs to be used * * Bytecode may be emitted to generate the necessary value. The return * value is either a register or a constant.
9664. **comment:** XXX: FP_ZERO check can be removed, the else clause handles it * correctly (preserving sign).
label: code-design
9665. **comment:** XXX: the string shouldn't appear twice, but we now loop to the * end anyway; if fixed, add a looping assertion to ensure there * is no duplicate.
label: code-design
9666. re-lookup first char on first loop
9667. [... buf loop (proplist)]
9668. DUK_BUFOBJ_INT16ARRAY
9669. No net refcount changes.
9670. DUK_USE_DATE_FMT_STRFTIME
9671. eat the right paren
9672. **comment:** A straightforward 64-byte lookup would be faster * and cleaner, but this is shorter.
label: code-design
9673. For this to work, DATEMSK must be set, so this is not very * convenient for an embeddable interpreter.
9674. Fixed buffer; data follows struct, with proper alignment guaranteed by * struct size.
9675. **comment:** * Call handling. * * Main functions are: * * - duk_handle_call_unprotected(): unprotected call to Ecmascript or * Duktape/C function * - duk_handle_call_protected(): protected call to Ecmascript or * Duktape/C function * - duk_handle_safe_call(): make a protected C call within current * activation * - duk_handle_ecma_call_setup(): Ecmascript-to-Ecmascript calls * (not always possible), including tail calls and coroutine resume * * See 'execution.rst'. * * Note: setjmp() and local variables have a nasty interaction, * see execution.rst; non-volatile locals modified after setjmp() * call are not guaranteed to keep their value.
label: code-design
9676. -1 if disjunction is complex, char length if simple
9677. noblame_fileline
9678. Backtrack to previous slash or start of buffer.
9679. func limits
9680. This is a rare property helper; it sets the global thrower (E5 Section 13.2.3) * setter/getter into an object property. This is needed by the 'arguments' * object creation code, function instance creation code, and Function.prototype.bind().
9681. For anything other than an Error instance, we calculate the error * location directly from the current activation.
9682. **comment:** Workaround for some exotic platforms where NAN is missing * and the expression (0.0 / 0.0) does NOT result in a NaN. * Such platforms use the global 'duk_computed_nan' which must * be initialized at runtime. Use 'volatile' to ensure that * the compiler will actually do the computation and not try * to do constant folding which might result in the original * problem.
label: code-design
9683. A -> target register (A, A+1) for call setup * (for DUK_OP_CSREGI, 'a' is indirect) * B -> register containing target function (not type checked here)
9684. Return the Buffer to allow chaining: b.fill(0x11).fill(0x22, 3, 5).toString()
9685. **comment:** code_idx: not needed
label: requirement
9686. keep func->h_funcs; inner functions are not reparsed to avoid O(depth^2) parsing
9687. load factor min 25%
9688. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
9689. Note: peek cannot currently trigger a detach * so the dbg_detaching == 0 assert outside the * loop is correct.
9690. Assume interrupt init/counter are properly initialized here.
9691. * Parse a case or default clause.
9692. **comment:** XXX: when this error is caused by a nonexistent * file given to duk_peval_file() or similar, the * error message is not the best possible.
label: code-design
9693. impose a reasonable exponent limit, so that exp * doesn't need to get tracked using a bigint.
9694. marker value; in E5 2^32-1 is not a valid array index (2^32-2 is highest valid)
9695. maxval
9696. valstack space allocated especially for proxy lookup which does a * recursive property lookup
9697. Accept ASCII strings which conform to identifier requirements * as being emitted without key quotes. Since we only accept ASCII * there's no need for actual decoding: 'p' is intentionally signed * so that bytes >= 0x80 extend to negative values and are rejected * as invalid identifier codepoints.
9698. The Str(key, holder) operation. * * Stack policy: [... key] -> [...]
9699. * Bitstream encoder
9700. -> [sep ToObject(this) len str sep]
9701. Decrement PC if that was requested, this requires a PC sync.
9702. Array properties have exotic behavior but they are concrete, * so no special handling here. * * Arguments exotic behavior (E5 Section 10.6, [[GetOwnProperty]]) * is only relevant as a post-check implemented below; hence no * check here.
9703. XXX: request a "last statement is terminal" from duk_parse_stmt() and duk_parse_stmts(); * we could avoid the last RETURN if we could ensure there is no way to get here * (directly or via a jump)
9704. [... key_obj key key flags]
9705. no side effects
9706. INACTIVE: no activation, single function value on valstack
9707. Equality may be OK but >length not. Checking * this explicitly avoids some overflow cases * below.
9708. this cannot initiate a detach

9709. done so that duk__mark_heaphdr() works correctly
9710. if highest bit of a register number is set, it refers to a constant instead
9711. **comment:** Slow variants, call to a helper to reduce code size. * Can be used explicitly when size is always more important than speed.
 label: code-design
9712. DUK_USE_REFERENCE_COUNTING
9713. Caller ensures space for at least DUK_JSON_MAX_ESC_LEN.
9714. byte offset to buf
9715. * The catch variable must be updated to reflect the new allocated * register for the duration of the catch clause. We need to store * and restore the original value for the varmap entry (if any).
9716. **comment:** unused now
 label: code-design
9717. **comment:** stack[0] = callback * stack[1] = thisArg * stack[2] = object * stack[3] = ToUInt32(length) (unused, but avoid unnecessary pop) * stack[4] = result array (or undefined)
 label: code-design
9718. still in prologue
9719. heap->dbg_processing: keep on purpose to avoid debugger re-entry in detaching state
9720. fixed-format termination conditions
9721. DUK_TOK_ELSE
9722. * Add line number to a compiler error.
9723. array of function templates: [func1, offset1, line1, func2, offset2, line2] * offset/line points to closing brace to allow skipping on pass 2
9724. Ensure copy is covered by underlying buffers.
9725. longjmp state
9726. * Note: * * - Intentionally attempt (empty) match at char_offset == k_input->clen * * - Negative char_offsets have been eliminated and char_offset is duk_uint32_t
 * -> no need or use for a negative check
9727. Expression, terminates at a '
9728. can't happen when keeping current stack size
9729. Special handling for CALL/NEW/MPUTOBJ/MPUTARR shuffling. * For each, slot B identifies the first register of a range * of registers, so normal shuffling won't work. Instead, * an indirect version of the opcode is used.
9730. DUK_TOK_FOR
9731. Miscellaneous.
9732. -> [... retval]
9733. If a setter/getter is missing (undefined), the descriptor must * still have the property present with the value 'undefined'.
9734. -> [timeval this]
9735. **comment:** Clean up stack
 label: code-design
9736. **comment:** dump all allocated entries, unused entries print as 'unused', * note that these may extend beyond current 'length' and look * a bit funny.
 label: code-design
9737. unwind to 'yield' caller
9738. How much stack to require on entry to object/array decode
9739. (e.g. k=3, digits=2 -> "12X")
9740. Note: changing from writable to non-writable is OK
9741. unwinds valstack, updating refcounts
9742. For JX, expressing the whole unsigned 32-bit range matters.
9743. r <- 2 * f
9744. currently active breakpoints: NULL term, borrowed pointers
9745. conversions, coercions, comparison, etc
9746. [key val] -> [key]
9747. use stack allocated buffer to ensure reachability in errors (e.g. intern error)
9748. **comment:** * Manually optimized double-to-fastint downgrade check. * * This check has a large impact on performance, especially for fastint * slow paths, so must be changed carefully. The code should probably be * optimized for the case where the result does not fit into a fastint, * to minimize the penalty for "slow path code" dealing with fractions etc. * * At least on one tested soft float ARM platform double-to-int64 coercion * is very slow (and sometimes produces incorrect results, see self tests). * This algorithm combines a fastint compatibility check and extracting the * integer value from an IEEE double for setting the tagged fastint. For * other platforms a more naive approach might be better. * * See doc/fastint.rst for details.
 label: code-design
9749. * Misc support functions
9750. ch is a literal character here or -1 if parsed entity was * an escape such as "\s".
9751. Code emission flags, passed in the 'opcode' field. Opcode + flags * fit into 16 bits for now, so use duk_small_uint.t.
9752. array of formal argument names (-> _Formals)
9753. [... func this arg1 ... argN]
9754. * Simple atom * * If atom_char_length is zero, we'll have unbounded execution time for e.g. * /()*/x/.exec('x'). We can't just skip the match because it might have some * side effects (for instance, if we allowed captures in simple atoms, the * capture needs to happen). The simple solution below is to force the * quantifier to match at most once, since the additional matches have no effect. * * With a simple atom there can be no capture groups, so no captures need * to be reset.
9755. key encountered as a getter
9756. **comment:** XXX: typing
 label: code-design
9757. catches EOF (NUL)
9758. 'act' is no longer accessed, scanbuild fix
9759. **comment:** XXX: Array size is known before and (2 * re_ctx.nsaved) but not taken * advantage of now. The array is not compacted either, as regexp match * objects are usually short lived.
 label: code-design
9760. Ecmascript activation + Duktape.Thread.yield() activation
9761. **comment:** Presence of 'fun' is config based, there's a marginal performance * difference and the best option is architecture dependent.
 label: code-design
9762. [... RegExp]
9763. DUK_TOK_DECREMENT
9764. [... errhandler undefined errval]
9765. **comment:** The target for this LDCONST may need output shuffling, but we assume * that 'pc_ldconst' will be the LDCONST that we can patch later. This * should be the case because there's no input shuffling. (If there's * no catch clause, this LDCONST will be replaced with a NOP.)
 label: code-design
9766. no wrap assuming h_bufobj->length is valid
9767. Nominal size check.
9768. register bound variables are non-configurable -> always false
9769. -> [obj trap handler target]
9770. reasonable output estimate
9771. Note: DST adjustment is determined using UTC time. * If 'd' is NaN, tzoffset will be 0.
9772. * Character and charcode access
9773. nargs == 2 so we can pass a callstack level to eval().
9774. First pass parse is very lenient (e.g. allows '1.2.3') and extracts a * string for strict number parsing.

9775. normal valued properties
9776. '\xffLexenv'
9777. Module loading failed. Node.js will forget the module * registration so that another require() will try to load * the module again. Mimic that behavior.
9778. **comment:** Note: for an accessor without getter, falling through to * check for "caller" exotic behavior is unnecessary as * "undefined" will never activate the behavior. But it does * no harm, so we'll do it anyway.
label: code-design
9779. e.g. 12.3 -> digits="123" k=2 -> 1.23e1
9780. **comment:** XXX: there is room to use a shared helper here, many built-ins * check the 'this' type, and if it's an object, check its class, * then get its internal value, etc.
label: code-design
9781. variable map for pass 2 (identifier -> register number or null (unmapped))
9782. buffer pointer is to an externally allocated buffer
9783. 'Uint32Array'
9784. 'with'
9785. **comment:** * (Re)initialize the temporary byte buffer. May be called extra times * with little impact.
label: code-design
9786. callstack index of related activation
9787. non-standard
9788. may be safe, or non-safe depending on flags
9789. activation prevents yield (native call or "new")
9790. **comment:** XXX: expensive, but numconv now expects to see a string
label: code-design
9791. DUK_FLD_VARINT; not relevant here
9792. **comment:** * Note: use indirect realloc variant just in case mark-and-sweep * (finalizers) might resize this same buffer during garbage * collection.
label: code-design
9793. eat 'try'
9794. 1110 xxxx 10xx xxxx 10xx xxxx
9795. Note: it's nice if size is 2^N (at least for 32-bit platforms).
9796. Quite heavy assert: check valstack policy. Improper * shuffle instructions can write beyond valstack_top/end * so this check catches them in the act.
9797. Expects 'this' at top of stack on entry.
9798. The token in the switch has already been eaten here
9799. currently all labels accept a break, so no explicit check for it now
9800. [arg result]
9801. got lineterm preceding non-whitespace, non-lineterm token
9802. intentional overlap with earlier memzero
9803. Allow exponent
9804. '\xffSource'
9805. read-only object, in code section
9806. **comment:** Slice and accessor information. * * Because the underlying buffer may be dynamic, these may be * invalidated by the buffer being modified so that both offset * and length should be validated before every access. Behavior * when the underlying buffer has changed doesn't need to be clean: * virtual 'length' doesn't need to be affected, reads can return * zero/NaN, and writes can be ignored. * * Note that a data pointer cannot be precomputed because 'buf' may * be dynamic and its pointer unstable.
label: code-design
9807. Shared object part.
9808. **comment:** XXX: there are language sensitive rules for the ASCII range. * If/when language/locale support is implemented, they need to * be implemented here for the fast path. There are no context * sensitive rules for ASCII range.
label: code-design
9809. DUK_OP_RETURN flags in A
9810. DUK_TOK_ADD
9811. allocate to reg only (not const)
9812. **comment:** not strictly necessary, but avoids "uninitialized variable" warnings
label: code-design
9813. * Augment an error at throw time; allow a user error handler (if defined) * to process/replace the error. The error to be augmented is at the * stack top.
9814. **comment:** This macro works when a regconst field is 9 bits, [0,0x1ff]. Adding * DUK_LIKELY/DUK_UNLIKELY increases code footprint and doesn't seem to * improve performance on x64 (and actually harms performance in some tests).
label: code-design
9815. requested number of output digits; 0 = free-format
9816. * Shared readfield and writefield methods * * The readfield/writefield methods need support for endianness and field * types. All offsets are byte based so no offset shifting is needed.
9817. Gap in array; check for inherited property, * bail out if one exists. This should be enough * to support gappy arrays for all practical code.
9818. **comment:** XXX: could improve bufwriter handling to write multiple codepoints * with one ensure call but the relative benefit would be quite small.
label: code-design
9819. **comment:** * Delete references to given hstring from the heap string cache. * * String cache references are 'weak': they are not counted towards * reference counts, nor serve as roots for mark-and-sweep. When an * object is about to be freed, such references need to be removed.
label: code-design
9820. DUK_TOK_EOF
9821. **comment:** XXX: signalling the need to shrink check (only if unwound)
label: code-design
9822. XXX: duk_dup_unvalidated(ctx, -2) etc.
9823. fill new entries with -unused- (required, gc reachable)
9824. -> [O toString O]
9825. accept anything, expect first value (EOF will be * caught by key parsing below.
9826. unknown, ignore
9827. * Note: fmax() does not match the E5 semantics. E5 requires * that if -any- input to Math.max() is a NaN, the result is a * NaN. fmax() will return a NaN only if - both- inputs are NaN. * Same applies to fmin(). * * Note: every input value must be coerced with ToNumber(), even * if we know the result will be a NaN anyway: ToNumber() may have * side effects for which even order of evaluation matters.
9828. [... v1 v2 name filename]
9829. !DUK_USE_PANIC_HANDLER
9830. **comment:** XXX: inlined DECREF macro would be nice here: no NULL check, * refzero queueing but no refzero algorithm run (= no pointer * instability), inline code.
label: code-design
9831. expected ival
9832. **comment:** A 16-bit listlen makes sense with 16-bit heap pointers: there * won't be space for 64k strings anyway.
label: code-design
9833. * Finalize refcounts for heap elements just about to be freed. * This must be done for all objects before freeing to avoid any * stale pointer dereferences. * * Note that this must deduce the set of objects to be freed * identically to duk_sweep_heap().
9834. no sce, or sce scan not best
9835. caller must have decref'd values above new_a_size (if that is necessary)

9836. [... buf loop (proplist) (gap)]

9837. **comment:** XXX: duk_hobject_hasprop() would be correct for * non-Proxy objects too, but it is about ~20-25% * slower at present so separate code paths for * Proxy and non-Proxy now.
label: code-design

9838. Coercion may be needed, the helper handles that by pushing the * tagged values to the stack.

9839. checked by caller

9840. 'Int16Array'

9841. res->strs16[] is zeroed and zero decodes to NULL, so no NULL inits.

9842. entry part must not contain any configurable properties, or * writable properties (if is_frozen).

9843. thread; minimizes argument passing

9844. step 4.d

9845. **comment:** * The Number constructor uses ToNumber(arg) for number coercion * (coercing an undefined argument to NaN). However, if the * argument is not given at all, +0 must be used instead. To do * this, a vararg function is used.
label: code-design

9846. one operator (= assign)

9847. no need to init reg, it will be undefined on entry

9848. string objects must not created without internal value

9849. eat backslash on entry

9850. * Cached module check. * * If module has been loaded or its loading has already begun without * finishing, return the same cached value ('exports'). The value is * registered when module load starts so that circular references can * be supported to some extent.

9851. * Setup a preliminary activation and figure out nargs/nregs. * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.

9852. **comment:** XXX: Rework to use an always-inline function?
label: code-design

9853. **comment:** * Common heap header * * All heap objects share the same flags and refcount fields. Objects other * than strings also need to have a single or double linked list pointers * for insertion into the "heap allocated" list. Strings are held in the * heap-wide string table so they don't need link pointers. * * Technically, 'h_refcount' must be wide enough to guarantee that it cannot * wrap (otherwise objects might be freed incorrectly after wrapping). This * means essentially that the refcount field must be as wide as data pointers. * On 64-bit platforms this means that the refcount needs to be 64 bits even * if an 'int' is 32 bits. This is a bit unfortunate, and compromising on * this might be reasonable in the future. * * Heap header size on 32-bit platforms: 8 bytes without reference counting, * 16 bytes with reference counting.
label: code-design

9854. We're a varargs function because we need to detect whether * initialValue was given or not.

9855. cannot be strict (never mapped variables)

9856. Avoid recursive re-entry; enter when we're attached or * detaching (to finish off the pending detach).

9857. DUK_DEBUGGER_H_INCLUDED

9858. Maximum length of CommonJS module identifier to resolve. Length includes * both current module ID, requested (possibly relative) module ID, and a * slash in between.

9859. **comment:** * Property count check. This is the only point where we ensure that * we don't get more (allocated) property space that we can handle. * There aren't hard limits as such, but some algorithms fail (e.g. * finding next higher prime, selecting hash part size) if we get too * close to the 4G property limit. * * Since this works based on allocation size (not actually used size), * the limit is a bit approximate but good enough in practice.
label: code-design

9860. This loop intentionally does not ensure characters are valid * ([0-9a-fA-F]) because the hex decode call below will do that.

9861. 0 if no used entries

9862. * Basic double / byte union memory layout.

9863. copy values (no overlap even if to_ctx == from_ctx; that's not * allowed now anyway)

9864. Use match charlen instead of bytelen, just in case the input and * match codepoint encodings would have different lengths.

9865. **comment:** Count actually used array part entries and array minimum size. * NOTE: 'out_min_size' can be computed much faster by starting from the * end and breaking out early when finding first used entry, but this is * not needed now.
label: code-design

9866. if new_p == NULL, all of these pointers are NULL

9867. -> [... obj retval/error]

9868. [... holder key] -> [... holder]

9869. 'this' binding is just before current activation's bottom

9870. **comment:** * Error handling macros, assertion macro, error codes. * * There are three level of 'errors': * * 1. Ordinary errors, relative to a thread, cause a longjmp, catchable. * 2. Fatal errors, relative to a heap, cause fatal handler to be called. * 3. Panic errors, unrelated to a heap and cause a process exit. * * Panics are used by the default fatal error handler and by debug code * such as assertions. By providing a proper fatal error handler, user * code can avoid panics in non-debug builds.
label: code-design

9871. Arguments can be: [source? filename? &comp_args] so that * nargs is 1 to 3. Call site encodes the correct nargs count * directly into flags.

9872. a

9873. Allow leading zeroes (e.g. "0123" -> "123")

9874. * Default clause.

9875. d <- (quotient (* r B) s) (in range 0...B-1)

9876. **comment:** XXX: There used to be a shared log buffer here, but it was removed * when dynamic buffer spare was removed. The problem with using * bufwriter is that, without the spare, the buffer gets passed on * as an argument to the raw() call so it'd need to be resized * (reallocated) anyway. If raw() call convention is changed, this * could be made more efficient.
label: code-design

9877. Commands initiated by debug client.

9878. next loop requires a comma

9879. 0x40...0x4f

9880. 'string'

9881. match == search string, by definition

9882. **comment:** Shared handling for encode/decode argument. Fast path handling for * buffer and string values because they're the most common. In particular, * avoid creating a temporary string or buffer when possible.
label: code-design

9883. 32-bit x 11-bit = 43-bit, fits accurately into a double

9884. Opcode only emitted by compiler when debugger * support is enabled. Ignore it silently without * debugger support, in case it has been loaded * from precompiled bytecode.

9885. eat closing brace

9886. Behavior for constructor and non-constructor call is * the same except for augmenting the created error. When * called as a constructor, the caller (duk_new()) will handle * augmentation; when called as normal function, we need to do * it here.

9887. Encode into a single opcode (18 bits can encode 1-2 bytes + length indicator)

9888. defined(DUK_USE_DATE_NOW_WINDOWS) || defined(DUK_USE_DATE_TZO_WINDOWS)

9889. without finally, the second jump slot is used to jump to end of stmt

9890. slot B is a target (default: source)

9891. fastint constants etc

9892. * Unicode codepoints above U+FFFF are encoded as surrogate * pairs here. This ensures that all CESU-8 codepoints are * 16-bit values as expected in EcmaScript. The surrogate * pairs always get a 3-byte encoding (each) in CESU-8. * See: http://en.wikipedia.org/wiki/Surrogate_pair * * 20-bit codepoint, 10 bits (A and B) per surrogate pair: * * x = 0b00000000 0000AAAA AAAAABBBBBBBB * sp1 = 0b110110AA AAAAaaaa (0xd800 + ((x >> 10) & 0x3ff)) * sp2 = 0b110111BB BBBBCCCC (0xdc00 + (x & 0x3ff)) * * Encoded into CESU-8: * * sp1 -> 0b11011010 (0xe0 + ((sp1 >> 12) & 0x0f)) * -> 0b1010AAAA (0x80 +

((sp1 >> 6) & 0x3f)) * -> 0b10AAAAAA (0x80 + (sp1 & 0x3f)) * sp2 -> 0b11101101 (0xe0 + ((sp2 >> 12) & 0x0f)) * -> 0b1011BBBB (0x80 + ((sp2 >> 6) & 0x3f)) * -> 0b10BBBBBB (0x80 + (sp2 & 0x3f)) * * Note that 0x10000 must be subtracted first. The code below * avoids the sp1, sp2 temporaries which saves around 20 bytes * of code.

9893. Duktape.modLoaded

9894. We could do this operation without caller updating bw_ctx->ptr, * but by writing it back here we can share code better.

9895. Note: if encoding ends by hitting end of input, we don't check that * the encoding is valid, we just assume it is.

9896. **comment:** XXX: call method tail call?

label: code-design

9897. w.../

9898. always provideThis=true

9899. Note: only numbered indices are relevant, so arr_idx fast reject * is good (this is valid unless there are more than 4**32-1 arguments).

9900. * Thread state and calling context checks

9901. token type (with reserved word identification)

9902. **comment:** XXX: there's no explicit recursion bound here now. For the average * qsort recursion depth O(log n) that's not really necessary: e.g. for * 2**32 recursion depth would be about 32 which is OK. However, qsort * worst case recursion depth is O(n) which may be a problem.

label: code-design

9903. verified at beginning

9904. **comment:** Fall back to the initial (original) Object.toString(). We don't * currently have pointers to the built-in functions, only the top * level global objects (like "Array") so this is now done in a bit * of a hacky manner. It would be cleaner to push the (original) * function and use duk_call_method().

label: code-design

9905. XXX: duk_push_uint_string()

9906. Built-in providers

9907. **comment:** XXX: could add flags for "is valid CESU-8" (EcmaScript compatible strings), * "is valid UTF-8", "is valid extended UTF-8" (internal strings are not, * regexp bytecode is), and "contains non-BMP characters". These are not * needed right now.

label: code-design

9908. pop final_cons

9909. **comment:** When ptr == NULL, the format argument is unused.

label: code-design

9910. To convert a heap stridx to a token number, subtract * DUK_STRIDX_START_RESERVED and add DUK_TOK_START_RESERVED.

9911. ANSI C typing

9912. These pointers are at the start of the struct so that they pack * nicely. Mixing pointers and integer values is bad on some * platforms (e.g. if int is 32 bits and pointers are 64 bits).

9913. 1=little endian

9914. * "IdentifierStart" production check.

9915. * Compiler state

9916. **comment:** XXX: this is awkward as we use an internal method which doesn't handle * extensibility etc correctly. Basically we'd want to do a [[DefineOwnProperty]] * or Object.defineProperty() here.

label: code-design

9917. **comment:** * Entries part is a bit more complex

label: code-design

9918. * On first pass, perform actual parsing. Remember valstack top on entry * to restore it later, and switch to using a new function in comp_ctx.

9919. nop: insert top to top

9920. h_match is borrowed, remains reachable through match_obj

9921. 0 = uppercase, 32 = lowercase (= 'a' - 'A')

9922. catch jump

9923. 3: toLocaleString

9924. not reachable

9925. leave 'cat' as top catcher (also works if catchstack exhausted)

9926. +1 for opcode

9927. 'varname' is in stack in this else branch, leaving an unbalanced stack below, * but this doesn't matter now.

9928. Not halfway, round to nearest.

9929. remove_in_val

9930. idx_argbase

9931. Strict by default.

9932. fixed precision and zero padding would be required

9933. Ensuring (reserving) space.

9934. flags: "m"

9935. 'setPrototypeOf'

9936. return 1 to allow callers to tail call

9937. * Not found (even in global object)

9938. * Reallocate memory with garbage collection, using a callback to provide * the current allocated pointer. This variant is used when a mark-and-sweep * (e.g. finalizers) might change the original pointer.

9939. 'number'

9940. label id allocation (running counter)

9941. **comment:** * Shared error messages: declarations and macros * * Error messages are accessed through macros with fine-grained, explicit * error message distinctions. Concrete error messages are selected by the * macros and multiple macros can map to the same concrete string to save * on code footprint. This allows flexible footprint/verbosity tuning with * minimal code impact. There are a few limitations to this approach: * (1) switching between plain messages and format strings doesn't work * conveniently, and (2) conditional strings are a bit awkward to handle. * * Because format strings behave differently in the call site (they need to * be followed by format arguments), they have a special prefix (DUK_STR_FMT_ * and duk_str_fmt_). * * On some compilers using explicit shared strings is preferable; on others * it may be better to use straight literals because the compiler will combine * them anyway, and such strings won't end up unnecessarily in a symbol table.

label: code-design

9942. **comment:** XXX: how to figure correct size?

label: code-design

9943. **comment:** * See the following documentation for discussion: * * doc/execution.rst: control flow details * * Try, catch, and finally "parts" are Blocks, not Statements, so * they must always be delimited by curly braces. This is unlike e.g. * the if statement, which accepts any Statement. This eliminates any * questions of matching parts of nested try statements. The Block * parsing is implemented inline here (instead of calling out). * * Finally part has a 'let scoped' variable, which requires a few kinks * here.

label: code-design

9944. op_flags

9945. # argument registers target function wants (< 0 => never for ecma calls)

9946. Different calling convention than above used because the helper * is shared.

9947. equality not actually possible

9948. **comment:** XXX: make this leniency dependent on flags or strictness?

label: code-design

9949. [...] func this arg1 ... argN envobj]

9950. 32: setDate

9951. slot C is a target (default: source)

9952. -> [... lval rval new_rval]
9953. Sealed and frozen objects cannot gain any more properties, * so this is a good time to compact them.
9954. Parse a single variable declaration (e.g. "i" or "i=10"). A leading 'var' * has already been eaten. These is no return value in 'res', it is used only * as a temporary. * * When called from 'for-in' statement parser, the initializer expression must * not allow the 'in' token. The caller supply additional expression parsing * flags (like DUK__EXPR_FLAG_REJECT_IN) in 'expr_flags'. * * Finally, out_rc_varname and out_reg_varbind are updated to reflect where * the identifier is bound: * * If register bound: out_reg_varbind >= 0, out_rc_varname == 0 (ignore) * If not register bound: out_reg_varbind < 0, out_rc_varname >= 0 * * These allow the caller to use the variable for further assignment, e.g. * as is done in 'for-in' parsing.
9955. sanity
9956. deal with negative values
9957. Unlike for negative arguments, some call sites * want length to be clamped if it's positive.
9958. create bound function object
9959. maximum bytecode copies for {n,m} quantifiers
9960. [... lval rval]
9961. **comment:** Note: Boolean prototype's internal value property is not writable, * but duk_xdef_prop_stridx() disregards the write protection. Boolean * instances are immutable. * * String and buffer special behaviors are already enabled which is not * ideal, but a write to the internal value is not affected by them.
label: code-design
9962. use duk_double_union as duk_tval directly
9963. [... obj finalizer obj heapDestruct] -> [... obj retval]
9964. **comment:** * Assignments are right associative, allows e.g. * a = 5; * a += b = 9; // same as a += (b = 9) * -> expression value 14, a = 14, b = 9 * * Right associativity is reflected in the BP for recursion, * "-1" ensures assignment operations are allowed. * * XXX: just use DUK_BP_COMMA (i.e. no need for 2-step bp levels)?
label: code-design
9965. don't increase 'count'
9966. module.exports = exports
9967. simulate alloc failure on every alloc (except when mark-and-sweep is running)
9968. must relookup act in case of side effects
9969. **comment:** XXX: take advantage of val being unsigned, no need to mask
label: code-design
9970. status booleans
9971. identifier handling
9972. duk_unicode_caseconv_lc()
9973. DUK_TOK_PROTECTED
9974. target
9975. reserved words: future reserved words
9976. Ensure space for maximum multi-character result; estimate is overkill.
9977. * Getters. * * Implementing getters is quite easy. The internal time value is either * NaN, or represents milliseconds (without fractions) from Jan 1, 1970. * The internal time value can be converted to integer parts, and each * part will be normalized and will fit into a 32-bit signed integer. * * A shared native helper to provide all getters. Magic value contains * a set of flags and also packs the date component index argument. The * helper provides: * * getFullYear() * getUTCFullYear() * getMonth() * getUTCMonth() * getDate() * getUTCDate() * getDay() * getUTCDay() * getHours() * getUTCHours() * getMinutes() * getUTCMinutes() * getSeconds() * getUTCSeconds() * getMilliseconds() * getUTCMilliseconds() * getYear() * * Notes: * * - Date.prototype.getDate(): 'date' means day-of-month, and is * zero-based in internal calculations but public API expects it to * be one-based. * * - Date.prototype.getTime() and Date.prototype.valueOf() have identical * behavior. They have separate function objects, but share the same C * function (duk_bi_date_prototype_value_of).
9978. * Pre resize assertions.
9979. In-place unary operation.
9980. Note: expect that caller has already eaten the left paren
9981. implicit leading one
9982. **comment:** * Array index and length * * Array index: E5 Section 15.4 * Array length: E5 Section 15.4.5.1 steps 3.c - 3.d (array length write) * * The DUK_HSTRING_GET_ARRIDX_SLOW() and DUK_HSTRING_GET_ARRIDX_FAST() macros * call duk_js_to_arrayindex_string_helper().
label: code-design
9983. * Internal helpers for managing object 'length'
9984. biased exp == 0 -> denormal, exp -1022
9985. 0xdeadbeef in decimal
9986. strings up to the this length are not cached
9987. DUK_HEAPHDR_H_INCLUDED
9988. [... key_obj key val]
9989. Checking callability of the immediate target * is important, same for constructability. * Checking it for functions down the bound * function chain is not strictly necessary * because .bind() should normally reject them. * But it's good to check anyway because it's * technically possible to edit the bound function * chain via internal keys.
9990. **comment:** * Three possible element formats: * 1)PropertyName : AssignmentExpression * 2) getPropertyName () { FunctionBody } * 3) setPropertyName (PropertySetParameterList) { FunctionBody } * * PropertyName can be IdentifierName (includes reserved words), a string * literal, or a number literal. Note that IdentifierName allows 'get' and * 'set' too, so we need to look ahead to the next token to distinguish: * * { get : 1 } * * and * * { get foo() { return 1 } } * { get get({ return 1 } } // 'get' as getter propertyname * * Finally, a trailing comma is allowed. * * Key name is coerced to string at compile time (and ends up as a * string constant) even for numeric keys (e.g. "{1:foo}"). * These could be emitted using e.g. LDINT, but that seems hardly * worth the effort and would increase code size.
label: code-design
9991. keep func->h_argnames; it is fixed for all passes
9992. num entries of new func at entry
9993. One digit octal escape, digit validated.
9994. Use result value as is.
9995. step 15.a
9996. formal arg limits
9997. Use the approach described in "Remarks" of FileTimeToLocalFileTime: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx)
9998. [offset noAssert], when ftype != DUK_FLD_VARINT
9999. bufwriter for temp accumulation
10000. normal comparison; known: * - both x and y are not NaNs (but one of them can be) * - both x and y are not zero (but one of them can be) * - x and y may be denormal or infinite
10001. -1 if invalid
10002. convenience helpers
10003. * Use Object.defineProperty() helper for the actual operation.
10004. Ensure compact use of temps.
10005. * Object handling: property access and other support functions.
10006. allocated from valstack (fixed buffer)
10007. Unlike most built-ins, the internal [[PrimitiveValue]] of a Date * is mutable.
10008. DUK_TVAL_H_INCLUDED
10009. * Retry with several GC attempts. Initial attempts are made without * emergency mode; later attempts use emergency mode which minimizes * memory allocations forcibly.
10010. last component
10011. lexing (tokenization) state (contains two valstack slot indices)

10012. * Update an existing property of the base object.
10013. update length (curr points to length, and we assume it's still valid)
10014. in-place setup
10015. Currently all built-in native functions are strict. * duk_push_c_function() now sets strict flag, so * assert for it.
10016. The lo/hi indices may be crossed and hi < 0 is possible at entry.
10017. * Init compilation context
10018. add E
10019. '\xffThread'
10020. Must use memmove() because copy area may overlap (source and target * buffer may be the same, or from different slices).
10021. * Special name handling
10022. If function creation fails due to out-of-memory, the data buffer * pointer may be NULL in some cases. That's actually possible for * GC code, but shouldn't be possible here because the incomplete * function will be unwound from the value stack and never instantiated.
10023. A -> register of target object * B -> first register of key/value pair list * C -> number of key/value pairs
10024. **comment:** XXX: this is a major target for size optimization
 label: code-design
10025. 'constructor' property for all built-in objects (which have it) has attributes: * [[Writable]] = true, * [[Enumerable]] = false, * [[Configurable]] = true
10026. XXX: check num_args
10027. pop 'name'
10028. Here 'p' always points to the start of a term. * * We can also unconditionally reset q_last here: if this is * the last (non-empty) term q_last will have the right value * on loop exit.
10029. * All done, switch properties ('p') allocation to new one.
10030. * Duktape.Buffer, Node.js Buffer, and Khronos/ES6 TypedArray built-ins
10031. decode '%xx' to '%6xx' if decoded char in reserved set
10032. **comment:** XXX: Finalizer lookup should traverse the prototype chain (to allow * inherited finalizers) but should not invoke accessors or proxy object * behavior.
 At the moment this lookup will invoke proxy behavior, so * caller must ensure that this function is not called if the target is * a Proxy.
 label: code-design
10033. Maximum number of breakpoints. Only breakpoints that are set are * consulted so increasing this has no performance impact.
10034. any other printable -> as is
10035. Called by duk_hstring.h macros
10036. step type: none, step into, step over, step out
10037. **comment:** XXX: faster implementation
 label: code-design
10038. cleared before entering catch part
10039. Clamp an input byte length (already assumed to be within the nominal * duk_hbufferobject 'length') to the current dynamic buffer limits to * yield a byte length limit that's safe for memory accesses. This value * can be invalidated by any side effect because it may trigger a user * callback that resizes the underlying buffer.
10040. **comment:** XXX: ignored now
 label: code-design
10041. **comment:** * Tagged type definition (duk_tval) and accessor macros. * * Access all fields through the accessor macros, as the representation * is quite tricky. * * There are two packed type alternatives: an 8-byte representation * based on an IEEE double (preferred for compactness), and a 12-byte * representation (portability). The latter is needed also in e.g. * 64-bit environments (it usually pads to 16 bytes per value). * * Selecting the tagged type format involves many trade-offs (memory * use, size and performance of generated code, portability, etc), * see doc/types.rst for a detailed discussion (especially of how the * IEEE double format is used to pack tagged values). * * NB: because macro arguments are often expressions, macros should * avoid evaluating their argument more than once.
 label: code-design
10042. * Automatically generated by extract_caseconv.py, do not edit!
10043. * Support functions for duk_heap.
10044. DUK_HCOMPILEDFUNCTION_H_INCLUDED
10045. write_to_array_part:
10046. **comment:** * Parse variant 1 or 2. The first part expression (which differs * in the variants) has already been parsed and its code emitted. * * reg_temps + 0: unused * reg_temps + 1: unused
 label: code-design
10047. E5 Section 15.5.5.1
10048. own pointer
10049. Note: although there is no 'undefined' literal, undefined * values can occur during compilation as a result of e.g. * the 'void' operator.
10050. Note: this may cause a corner case situation where a finalizer * may see a currently reachable activation whose 'func' is NULL.
10051. # total registers target function wants on entry (< 0 => never for ecma calls)
10052. bytecode has a stable pointer
10053. * Thread builtins
10054. overwrite str1
10055. **comment:** XXX: exposed duk_debug_read_buffer
 label: code-design
10056. DUK_JS_BYTECODE_H_INCLUDED
10057. Convert a duk_tval number (caller checks) to a 32-bit index. Returns * DUK__NO_ARRAY_INDEX if the number is not whole or not a valid array * index.
10058. SCANBUILD: scan-build produces a NULL pointer dereference warning * below; it never actually triggers because holder is actually never * NULL.
10059. Set up pointers to the new property area: this is hidden behind a macro * because it is memory layout specific.
10060. no need for refcount update
10061. **comment:** Zero 'count' is also allowed to make call sites easier. * Arithmetic in bytes generates better code in GCC.
 label: code-design
10062. compiler ensures this
10063. [... old_result] -> [... result]
10064. **comment:** This shouldn't be necessary, but check just in case * to avoid any chance of overruns.
 label: code-design
10065. At the moment Buffer(<str>) will just use the string bytes as * is (ignoring encoding), so we return the string length here * unconditionally.
10066. * concat()
10067. different memory layout, alloc size, and init
10068. t2 = (* (+ r m+) B)
10069. exponent non-negative (and thus not minimum exponent)
10070. default: NULL, length 0
10071. env is closed, should be missing _Callee, _Thread, _Regbase
10072. **comment:** The E5.1 algorithm checks whether or not a decoded codepoint * is below 0x80 and perhaps may be in the "reserved" set. * This seems pointless because the single byte UTF-8 case is * handled separately, and non-shortest encodings are rejected. * So, 'cp' cannot be below 0x80 here, and thus cannot be in * the reserved set.
 label: code-design
10073. force_flag
10074. Used for minimal 'const': initializer required.
10075. * Table for base-64 encoding
10076. Note: no allocation pressure, no need to check refcounts etc
10077. args incorrect

10078. write on next loop
10079. * Helper to format a time value into caller buffer, used by logging. * 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.
10080. **comment:** XXX: the returned value is exotic in ES6, but we use a * simple object here with no prototype. Without a prototype, * [[DefaultValue]] coercion fails which is abit confusing. * No callable check/handling in the current Proxy subset.
label: code-design
10081. 2nd related value (type specific)
10082. **comment:** Extraop arithmetic opcodes must have destination same as * first source. If second source matches destination we need * a temporary register to avoid clobbering the second source. ** XXX: change calling code to avoid this situation in most cases.
label: code-design
10083. cannot grow
10084. 0=indexOf, 1=lastIndexOf
10085. bump up "allocated" reg count, just in case
10086. Strict standard behavior, ignore trailing elements for * result 'length'.
10087. For errors a string coerced result is most informative * right now, as the debug client doesn't have the capability * to traverse the error object.
10088. DUK_TOK_IF
10089. **comment:** NOTE: This is a bit fragile. It's important to ensure that * duk_debug_process_messages() never throws an error or * act->curr_pc will never be reset.
label: code-design
10090. Working with the pointer and current size.
10091. object has any exotic behavior(s)
10092. DUK_USE_STRTAB_PROBE
10093. written by a previous RESUME handling
10094. **comment:** * Macro support functions (use only macros in calling code)
label: code-design
10095. string 1 of token (borrowed, stored to ctx->slot1_idx)
10096. **comment:** XXX: more accurate?
label: code-design
10097. Note: assume array part is comprehensive, so that either * the write goes to the array part, or we've abandoned the * array above (and will not come here).
10098. level string
10099. sufficient for keeping temp reg numbers in check
10100. All other object types.
10101. **comment:** Flip highest bit of each byte which changes * the bit pattern 10xxxxxx into 00xxxxxx which * allows an easy bit mask test.
label: test
10102. mixed endian
10103. min_new_size
10104. length in bytes (not counting NUL term)
10105. No finalizers for ROM objects
10106. last element that was left in the heap
10107. compact; no need to seal because object is internal
10108. 0x90...0x9f
10109. const flag for C
10110. [...] -> [...]
10111. number of pairs in current MPUTOBJ set
10112. escaped NonEscapeCharacter
10113. Module id requested
10114. Empty searchstring always matches; cpos must be clamped here. * (If q_blen were < 0 due to clamped coercion, it would also be * caught here.)
10115. Never executed if new size is smaller.
10116. 'protected'
10117. because arg count is 1
10118. 'if'
10119. finally free the struct itself
10120. !DUK_USE_PARANOI_ERRORS
10121. Array or Array-like
10122. coerce towards zero
10123. must not be extensible
10124. **comment:** We've ensured space for one escaped input; then * bail out and recheck (this makes escape handling * quite slow but it's uncommon).
label: code-design
10125. mark-and-sweep: finalizable (on current pass)
10126. [...] replacer match [captures] match_char_offset input]
10127. DUK_OP_TRYCATCH flags in A
10128. single string token
10129. Use byte copy.
10130. Most common cases first.
10131. no point in supporting encodings of 5 or more bytes
10132. '|'
10133. use_prev_pc
10134. We need 'nbytes' even for a failed offset; return value must be * (offset + nbytes) even when write fails due to invalid offset.
10135. Return value handling.
10136. Enable DUKFUNC exotic behavior once properties are set up.
10137. duk.bi_duk_object_yield() and duk.bi_duk_object_resume() ensure all of these are met
10138. Assumes that caller has normalized NaNs, otherwise trouble ahead.
10139. 0x00: finish (not part of number) * 0x01: continue
10140. **comment:** this is relatively expensive
label: code-design
10141. num_values and temp_start reset at top of outer loop
10142. **comment:** * Panic error ** Panic errors are not relative to either a heap or a thread, and cause * DUK_PANIC() macro to be invoked. Unless a user provides DUK_USE_PANIC_HANDLER, * DUK_PANIC() calls a helper which prints out the error and causes a process * exit. ** The user can override the macro to provide custom handling. A macro is * used to allow the user to have inline panic handling if desired (without * causing a potentially risky function call). ** Panics are only used in debug code such as assertions, and by the default * fatal error handler.
label: code-design
10143. specific assert for wrapping
10144. Could also rely on native sprintf(), but it will handle * values like NaN, Infinity, -0, exponent notation etc in * a JSON-incompatible way.
10145. [arg1 ... argN this loggerLevel loggerName buffer 'raw' buffer]
10146. **comment:** Function declaration for global/eval code is emitted even * for duplicates, because of E5 Section 10.5, step 5.e of * E5.1 (special behavior for variable bound to global object). ** DECLVAR will not re-declare a variable as such, but will * update the binding value.
label: code-design
10147. bits 2...5: type
10148. As a first approximation, buffer values are coerced to strings * for addition. This means that adding two buffers currently * results in a string.
10149. must be ecmascript

10150. 'toUTCString'
10151. Note: no need to re-lookup tv, conversion is side effect free
10152. require.id of current module
10153. low 32 bits is complete
10154. time when status/peek was last done (Date-based rate limit)
10155. [obj key value desc value]
10156. 'caller'
10157. rbp_flags
10158. 'data'
10159. 'type'
10160. **comment:** XXX: shared strings
 label: code-design
10161. -> [... source]
10162. default value
10163. output 3 bytes from 't'
10164. Special flags checks. Since these strings are always * reachable and a string cannot appear twice in the string * table, there's no need to check/set these flags elsewhere. * The 'internal' flag is set by string intern code.
10165. loop iterator init and limit changed from standard algorithm
10166. enum_flags
10167. '++' or '-' in a post-increment/decrement position, * and a LineTerminator occurs between the operator and * the preceding expression. Force the previous expr * to terminate, in effect treating e.g. "a,b\n++" as * "a,b;++" (= SyntaxError).
10168. by default, use caller's environment
10169. * Note: we currently assume that the setjmp() catchpoint is * not re-entrant (longjmp() cannot be called more than once * for a single setjmp()). * * See doc/code-issues.rst for notes on variable assignment * before and after setjmp().
10170. value resides in 'valstack_idx'
10171. Verbose errors with key/value summaries (non-paranoid) or without key/value * summaries (paranoid, for some security sensitive environments), the paranoid * vs. non-paranoid distinction affects only a few specific errors.
10172. 'typeof' must handle unresolvable references without throwing * a ReferenceError (E5 Section 11.4.3). Register mapped values * will never be unresolvable so special handling is only required * when an identifier is a "slow path" one.
10173. free inner references (these exist e.g. when external * strings are enabled)
10174. The 'this' after 'sep' will get ToString() coerced by * duk_join() automatically. We don't want to do that * coercion when providing .fileName or .lineNumber (GH-254).
10175. surrogate pairs get encoded here
10176. 11: getUTCMonth
10177. **comment:** XXX: any chance of unifying this with the 'length' key handling?
 label: code-design
10178. assert just a few critical flags
10179. correction
10180. default to false
10181. * Stack slice primitives
10182. Length in elements: take into account shift, but * intentionally don't check the underlying buffer here.
10183. [key val] -> []
10184. DUK_USE_STRHASH_DENSE
10185. * Get old and new length
10186. clipped codepoint
10187. h_new_proto may be NULL
10188. -> [... errhandler retval]
10189. this is the case for normalized numbers
10190. flags: "gim"
10191. Starting from this round, use emergency mode * for mark-and-sweep.
10192. The eval code is executed within the lexical environment of a specified * activation. For now, use global object eval() function, with the eval * considered a 'direct call to eval'. * * Callstack level for debug commands only affects scope -- the callstack * as seen by, e.g. Duktape.act() will be the same regardless.
10193. * Found existing own or inherited plain property, but original * base is a primitive value.
10194. optional callstack level
10195. e.g. duk_push_string_file_raw() pushed undefined
10196. Assume that either all memory funcs are NULL or non-NUL, mixed * cases will now be unsafe.
10197. 'error'
10198. don't throw for prototype loop
10199. (< (* r 2) s)
10200. [thisArg arg1 ... argN func] (thisArg+args == nargs total)
10201. Note: not set in template (has no "prototype")
10202. unreferenced w/o asserts
10203. if (is_repl_func)
10204. precision:shortest
10205. don't resize stringtable (but may sweep it); needed during stringtable resize
10206. Write curr_pc back for the debugger.
10207. Note: duk_dec_req_stridix() backtracks one char
10208. 'length'
10209. -> [... val retval]
10210. Object property access
10211. * The specification uses RegExp [[Match]] to attempt match at specific * offsets. We don't have such a primitive, so we use an actual RegExp * and tweak lastIndex. Since the RegExp may be non-global, we use a * special variant which forces global-like behavior for matching.
10212. 3
10213. Apply timezone offset to get the main parts in UTC
10214. * Zero sign, see misc/tcc_zerosign2.c.
10215. * Boolean built-ins
10216. LeftHandSideExpression does not allow empty expression
10217. **comment:** XXX: add proper spare handling to dynamic buffer, to minimize * reallocs; currently there is no spare at all.
 label: code-design
10218. Note: escaped characters differentiate directives
10219. defaults, E5 Section 8.6.1, Table 7
10220. * Intern the temporary byte buffer into a valstack slot * (in practice, slot1 or slot2).
10221. special RegExp literal handling after IdentifierName
10222. XXX: byte offset?
10223. 'switch'
10224. * The 'duktape.h' header provides the public API, but also handles all * compiler and platform specific feature detection, Duktape feature * resolution, inclusion of system headers, etc. These have been merged * because the public API is also dependent on e.g. detecting appropriate * C types which is quite platform/compiler specific especially for a non-C99 * build. The public API is also dependent on the resolved feature set. * * Some actions taken by the merged header (such as

including system headers) * are not appropriate for building a user application. The define * DUK_COMPILING_DUKTAPE allows the merged header to skip/include some * sections depending on what is being built.

10225. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * Lightfunc detection happens here too. Note that lightweight functions * can be wrapped by (non-lightweight) bound functions so we must resolve * the bound function chain first.

10226. avoid side effects

10227. result in csreg

10228. * Other heap related defines

10229. * Start doing property attributes updates. Steps 12-13. * * Start by computing new attribute flags without writing yet. * Property type conversion is done above if necessary.

10230. strict: non-deletable, non-writable

10231. * Assuming a register binds to a variable declared within this * function (a declarative binding), the 'this' for the call * setup is always 'undefined'. E5 Section 10.2.1.1.6.

10232. This may happen when forward and backward scanning disagree * (possible for non-extended-UTF-8 strings).

10233. use fallback as 'this' value

10234. Caller has already eaten the first character ('(') which we don't need.

10235. r <- (* r B) * s <- s * m+ <- (* m+ B) * m- <- (* m- B) * k <- (- k 1)

10236. A -> object reg * B -> key reg/const * C -> value reg/const * * Note: intentional difference to register arrangement * of e.g. GETPROP; 'A' must contain a register-only value.

10237. C recursion check.

10238. b

10239. Misc

10240. catches EOF (0x00)

10241. **comment:** XXX: duk_get_length?
label: code-design

10242. XXX: if assertions enabled, walk through all valid PCs * and check line mapping.

10243. 'defineProperty'

10244. **comment:** XXX: awkward and bloated asm -- use faster internal accesses
label: code-design

10245. approximation, close enough

10246. marker for detecting internal "double faults", see duk_error_throw.c

10247. three flags

10248. No difference between raw/ensure because the buffer shrinks.

10249. * Determining which datetime components to overwrite based on * stack arguments is a bit complicated, but important to factor * out from setters themselves for compactness. * * If DUK_DATE_FLAG_TIMESETTER, maxnargs indicates setter type: * * 1 -> millisecond * 2 -> second, [millisecond] * 3 -> minute, [second], [millisecond] * 4 -> hour, [minute], [second], [millisecond] * * Else: * * 1 -> date * 2 -> month, [date] * 3 -> year, [month], [date] * * By comparing nargs and maxnargs (and flags) we know which * components to override. We rely on part index ordering.

10250. False if the object is NULL or the prototype 'p' is NULL. * In particular, false if both are NULL (don't compare equal).

10251. Start filling in the activation

10252. SCANBUILD: complains about use of uninitialized values. * The complaint is correct, but operating in undefined * values here is intentional in some cases and the caller * ignores the results.

10253. prevent duk__expr_led() by using a binding power less than anything valid

10254. borrowed reference; although 'tv1' comes from a register, * its value was loaded using LDCONST so the constant will * also exist and be reachable.

10255. [... proplist enum_obj key val]

10256. * Unix-like Date providers * * Generally useful Unix / POSIX / ANSI Date providers.

10257. Push the current 'this' binding; throw TypeError if binding is not object * coercible (CheckObjectCoercible).

10258. All strings beginning with 0xff are treated as "internal", * even strings interned by the user. This allows user code to * create internal properties too, and makes behavior consistent * in case user code happens to use a string also used by Duktape * (such as string has already been interned and has the 'internal' * flag set).

10259. 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [31 bits]

10260. shrink failure is not fatal

10261. !(l < p)

10262. valstack index of loop detection object

10263. limit is quite low: one array entry is 8 bytes, one normal entry is 4+1+8+4 = 17 bytes (with hash entry)

10264. yes -> move back to heap allocated

10265. strict flag for putvar comes from our caller (currently: fixed)

10266. catchstack limits

10267. * Manually optimized number-to-double conversion

10268. [ToObject(this) ToUint32(length) lowerValue upperValue]

10269. if a tail call: * - an Ecmascript activation must be on top of the callstack * - there cannot be any active catchstack entries

10270. [... enum_target res trap_result val true]

10271. Helper for component getter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), push a specified component as a return value to the * value stack and return 1 (caller can then tail call us).

10272. important for callers

10273. fail

10274. [sep ToObject(this) len]

10275. DUK_TOK_PERIOD

10276. Enumeration keys are checked against the enumeration target (to see * that they still exist). In the proxy enumeration case _Target will * be the proxy, and checking key existence against the proxy is not * required (or sensible, as the keys may be fully virtual).

10277. Easiest way to implement the search required by the specification * is to do a RegExp test() with lastIndex forced to zero. To avoid * side effects on the argument, "clone" the RegExp if a RegExp was * given as input. * * The global flag of the RegExp should be ignored; setting lastIndex * to zero (which happens when "cloning" the RegExp) should have an * equivalent effect.

10278. idx_reviver

10279. **comment:** Require a lot of stack to force a value stack grow/shrink. * Recursive mark-and-sweep is prevented by allocation macros * so this won't trigger another mark-and-sweep.
label: code-design

10280. signed shift

10281. Impose a string maximum length, need to handle overflow * correctly.

10282. **comment:** XXX: casts could be improved, especially for GET/SET DATA
label: code-design

10283. bytes left

10284. **comment:** XXX: these last tricks are unnecessary if the function is made * a genuine native function.
label: code-design

10285. No locale specific formatter; this is OK, we fall back * to ISO 8601.

10286. failed

10287. -> [... key]

10288. **comment:** Tested: not faster on x64

label: code-design

10289. clamp to 10 chars

10290. * "LineTerminator" production check.

10291. A lightfunc might also inherit a . toJSON() so just bail out.

10292. **comment:** Note: this is very tricky; we must never 'overshoot' the * correction downwards.

label: code-design

10293. **comment:** XXX: inefficient; block insert primitive

label: code-design

10294. * Exposed helper for setting up heap longjmp state.

10295. in non-packed representation we don't care about which NaN is used

10296. **comment:** Period followed by a digit can only start DecimalLiteral * (handled in slow path). We could jump straight into the * DecimalLiteral handling but should avoid goto to inside * a block.

label: code-design

10297. zero

10298. Handle change in value stack top. Respect value stack * initialization policy: 'undefined' above top. Note that * DECREF may cause a side effect that reallocates valstack, * so must relookup after DECREF.

10299. **comment:** XXX: tostring?

label: code-design

10300. 'func' wants stack "as is"

10301. * Debug message processing

10302. Bottom of valstack for this activation, used to reset * valstack_bottom on return; index is absolute. Note: * idx_top not needed because top is set to 'nregs' always * when returning to an Ecmascript activation.

10303. * Sweep heap

10304. Object.keys

10305. map is reachable through obj

10306. **comment:** XXX: this needs to be reworked so that we never shrink the value * stack on function entry so that we never need to grow it here. * Needing to grow here is a sandboxing issue because we need to * allocate which may cause an error in the error handling path * and thus propagate an error out of a protected call.

label: code-design

10307. **comment:** * Note: since this is an exposed API call, there should be * no way a mark-and-sweep could have a side effect on the * memory allocation behind 'ptr'; the pointer should never * be something that Duktape wants to change. ** Thus, no need to use DUK_REALLOC_INDIRECT (and we don't * have the storage location here anyway).

label: code-design

10308. required for rehash to succeed, equality not that useful

10309. DUK_TOK_CLASS

10310. **comment:** unused for label

label: code-design

10311. 'this' binding shouldn't matter here

10312. **comment:** remove old value

label: code-design

10313. **comment:** get rid of the strings early to minimize memory use before intern

label: code-design

10314. always in entry part, no need to look up parents etc

10315. * Raw intern and lookup

10316. Copy the property table verbatim; this handles attributes etc. * For ROM objects it's not necessary (or possible) to update * refcounts so leave them as is.

10317. CONDITIONAL EXPRESSION

10318. swap elements; deal with non-existent elements correctly

10319. [... closure/error]

10320. * Create a new property in the original object. ** Exotic properties need to be reconsidered here from a write * perspective (not just property attributes perspective). * However, the property does not exist in the object already, * so this limits the kind of exotic properties that apply.

10321. Initial value for highest_idx is -1 coerced to unsigned. This * is a bit odd, but (highest_idx + 1) will then wrap to 0 below * for out_min_size as intended.

10322. ignore

10323. **comment:** XXX: combine all the integer conversions: they share everything * but the helper function for coercion.

label: code-design

10324. vararg function, thisArg needs special handling

10325. Assert for value stack initialization policy

10326. **comment:** * Shallow fast path checks for accessing array elements with numeric * indices. The goal is to try to avoid coercing an array index to an * (interned) string for the most common lookups, in particular, for * standard Array objects. ** Interning is avoided but only for a very narrow set of cases: * - Object has array part, index is within array allocation, and * value is not unused (= key exists) * - Object has no interfering exotic behavior (e.g. arguments or * string object exotic behaviors interfere, array exotic * behavior does not). ** Current shortcoming: if key does not exist (even if it is within * the array allocation range) a slow path lookup with interning is * always required. This can probably be fixed so that there is a * quick fast path for non-existent elements as well, at least for * standard Array objects.

label: code-design

10327. curr_token slot1 (matches 'lex' slot1_idx)

10328. eat 'break' or 'continue'

10329. zero-based -> one-based

10330. roughly 512 bytes

10331. * indexOf() and lastIndexOf()

10332. DUK_USE_HEAPPTR16

10333. regexp literal must not follow this token

10334. qmin and qmax will be 0 or 1

10335. same as during entry

10336. Have a catch variable.

10337. Note: in strict mode the compiler should reject explicit * declaration of 'eval' or 'arguments'. However, internal * bytecode may declare 'arguments' in the function prologue. * We don't bother checking (or asserting) for these now.

10338. Object property allocation layout

10339. Node.js accepts only actual Arrays.

10340. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined]

10341. **comment:** XXX: Shrink the stacks to minimize memory usage? May not * be worth the effort because terminated threads are usually * garbage collected quite soon.

label: code-design

10342. * Actual Object.defineProperty() default algorithm.

10343. t1 = milliseconds within day (fits 32 bit) * t2 = day number from epoch (fits 32 bit, may be negative)

10344. 'class'

10345. compare: similar to string comparison but for buffer data.

10346. Note: if 'x' is zero, x->n becomes 0 here

10347. Debugger protocol version is defined in the public API header.

10348. not undefined

10349. Endianness indicator

10350. number of activation records in callstack preventing a yield
10351. DUK_TOK_IMPLEMENT
10352. 4'294'967'295
10353. XXX: switch cast?
10354. 'trace'
10355. array 'length' is always a number, as we coerce it
10356. NULL is allowed, no output
10357. * Convert duk_compiler_func to a function template
10358. Update duk_tval in-place if pointer provided and the * property is writable. If the property is not writable * (immutable binding), use duk_hobject_putprop() which * will respect mutability.
10359. Better equivalent algorithm. If the compiler is compliant, C and * Ecmascript semantics are identical for this particular comparison. * In particular, NaNs must never compare equal and zeroes must compare * equal regardless of sign. Could also use a macro, but this inlines * already nicely (no difference on gcc, for instance).
10360. Is whole and within 32 bit range. If the value happens to be 0xFFFFFFFF, * it's not a valid array index but will then match DUK_NO_ARRAY_INDEX.
10361. * PLAIN: x1 * ARITH: x1 <op> x2 * PROP: x1.x2 * VAR: x1 (name)
10362. eat closing slash
10363. * Regexp range tables
10364. Value stack intentionally mixed size here.
10365. Computation must not wrap; this limit works for 32-bit size_t: * >>> srclen = 3221225469 * >>> '%x' % ((srclen + 2) / 3 * 4) * 'fffffc'
10366. **comment:** * Special post-tweaks, for cases not covered by the init data format. * * - Set Date.prototype.toGMTString to Date.prototype.toUTCString. * toGMTString is required to have the same Function object as * toUTCString in E5 Section B.2.6. Note that while Smjs respects * this, V8 does not (the Function objects are distinct). * * - Make DoubleError non-extensible. * * - Add info about most important effective compile options to Duktape. * * - Possibly remove some properties (values or methods) which are not * desirable with current feature options but are not currently * conditional in init data.
label: code-design
10367. hobject property layout
10368. **comment:** UNUSED, intentionally empty
label: code-design
10369. double quote or backslash
10370. track utf-8 non-continuation bytes
10371. [hobject props enum(props)]
10372. 22 to 31
10373. **comment:** XXX: very similar to DUK_IVAL_ARITH - merge?
label: code-design
10374. **comment:** XXX: optional check, match_caps is zero if no regexp, * so dollar will be interpreted literally anyway.
label: code-design
10375. * Internal helper to define a property with specific flags, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes unless caller requests * that value only be updated if it doesn't already exists. * * Does not support: * - virtual properties (error if write attempted) * - getter/setter properties (error if write attempted) * - non-default (!= WEC) attributes for array entries (error if attempted) * - array abandoning: if array part exists, it is always extended * - array 'length' updating * * Stack: [... in_val] -> [] * * Used for e.g. built-in initialization and environment record * operations.
10376. -> [... enum key enum_target val]
10377. constants arbitrary, chosen for small loads
10378. reset 'allow_in' for parenthesized expression
10379. **comment:** Internal class is Object: Object.prototype.toString.call(new Buffer(0)) * prints "[object Object]".
label: code-design
10380. * The entire allocated buffer area, regardless of actual used * size, is kept zeroed in resizes for simplicity. If the buffer * is grown, zero the new part.
10381. XXX: integrate support for this into led() instead? * Similar issue as post-increment/post-decrement.
10382. E5 Sections 11.8.2, 11.8.5; x > y --> y < x
10383. if present, forces 16-byte duk_tval
10384. * 'arguments' binding is special; if a shadowing argument or * function declaration exists, an arguments object will * definitely not be needed, regardless of whether the identifier * 'arguments' is referenced inside the function body.
10385. exports (this binding)
10386. disabled
10387. **comment:** XXX: other variants like uint, u32 etc
label: code-design
10388. rely on interning, must be this string
10389. Does not allow e.g. 2**31-1, but one more would allow overflows of u32.
10390. Note: cannot be a bound function either right now, * this would be easy to relax though.
10391. Match labels starting from latest; once label_id no longer matches, we can * safely exit without checking the rest of the labels (only the topmost labels * are ever updated).
10392. The 'else' ambiguity is resolved by 'else' binding to the innermost * construct, so greedy matching is correct here.
10393. Short circuit if is safe: if act->curr_pc != NULL, 'fun' is * guaranteed to be a non-NULL Ecmascript function.
10394. read-only values 'lifted' for ease of use
10395. return value
10396. 'source'
10397. 'clog'
10398. digits
10399. * Statement value handling. * * Global code and eval code has an implicit return value * which comes from the last statement with a value * (technically a non-'empty' continuation, which is * different from an empty statement). * * Since we don't know whether a later statement will * override the value of the current statement, we need * to coerce the statement value to a register allocated * for implicit return values. In other cases we need * to coerce the statement value to a plain value to get * any side effects out (consider e.g. "foo.bar;").
10400. is_frozen
10401. **comment:** Downgrade checks are not made everywhere, so 'length' is not always * a fastint (it is a number though). This can be removed once length * is always guaranteed to be a fastint.
label: code-design
10402. if duk_uint32_t is exactly 32 bits, this is a NOP
10403. * Not found as a concrete property, check whether a String object * virtual property matches.
10404. * Function formal arguments, always bound to registers * (there's no support for shuffling them now).
10405. off >= step, and step >= 1
10406. zero-width joiner
10407. Rethrow error to calling state.
10408. ""
10409. next temporary register to allocate
10410. * For bound objects, [[HasInstance]] just calls the target function * [[HasInstance]]. If that is again a bound object, repeat until * we find a non-bound Function object.
10411. Pointer and buffer primitive values are treated like other * primitives values which have a fully fledged object counterpart: * promote to an object value. Lightfuncs are coerced with * ToObject() even they could also be returned as is.
10412. **comment:** XXX: option to fix opcode length so it lines up nicely
label: code-design

10413. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT16
10414. relookup after side effects (no side effects currently however)
10415. Note: this order matters (final value before deleting map entry must be done)
10416. 'arguments'
10417. * Fast path for bufferobject getprop/putprop
10418. XXX: shrink array allocation or entries compaction here?
10419. inline arithmetic check for constant values
10420. pre-incremented, points to first jump slot
10421. * Helpers for duk_compiler_func.
10422. Note: need to re-lookup because ToNumber() may have side effects
10423. expr_flags
10424. **comment:** XXX: exposed duk_debug_read_hbuffer
 label: code-design
10425. accept string
10426. unreferenced without assertions
10427. **comment:** XXX: macro checks for array index flag, which is unnecessary here
 label: code-design
10428. **comment:** XXX: this algorithm could be optimized quite a lot by using e.g. * a logarithm based estimator for 'k' and performing B^n multiplication * using a lookup table or using some bit-representation based exp * algorithm. Currently we just loop, with significant performance * impact for very large and very small numbers.
 label: code-design
10429. 'toString'
10430. first character has been matched
10431. 0xffff0 is -Infinity
10432. **comment:** XXX: don't want to shrink allocation here
 label: code-design
10433. -> [... error]
10434. Lightfunc coerces to a Function instance with concrete * properties. Since 'length' is virtual for Duktape/C * functions, don't need to define that. ** The result is made extensible to mimic what happens to * strings: * > Object.isExtensible(Object('foo')) * true
10435. curr_token follows 'function'
10436. * Update preventcount
10437. **comment:** These are not needed to implement quantifier capture handling, * but might be needed at some point.
 label: requirement
10438. toJSON'
10439. function executes as a constructor (called via "new")
10440. heaphdr: * - is not reachable * - is an object * - is not a finalized object * - has a finalizer
10441. DUK_ERROR_H_INCLUDED
10442. Number/string-or-buffer -> coerce string to number (e.g. "'1.5' == 1.5" -> true).
10443. XXX: There are several limitations in the current implementation for * strings with >= 0x80000000UL characters. In some cases one would need * to be able to represent the range [-0xffffffff,0xffffffff] and so on. * Generally character and byte length are assumed to fit into signed 32 * bits (< 0x80000000UL). Places with issues are not marked explicitly * below in all cases, look for signed type usage (duk_int_t etc) for * offsets/lengths.
10444. * Reset function state and perform register allocation, which creates * 'varmap' for second pass. Function prologue for variable declarations, * binding value initializations etc is emitted as a by-product. ** Strict mode restrictions for duplicate and invalid argument * names are checked here now that we know whether the function * is actually strict. See: test-dev-strict-mode-boundary.js. ** Inner functions are compiled during pass 1 and are not reset.
10445. if 1, doing a fixed format output (not free format)
10446. We want to do a straight memory copy if possible: this is * an important operation because .set() is the TypedArray * way to copy chunks of memory. However, because set() * conceptually works in terms of elements, not all views are * compatible with direct byte copying. ** If we do manage a direct copy, the "overlap issue" handled * below can just be solved using memmove() because the source * and destination element sizes are necessarily equal.
10447. borrowed; NULL if no step state (NULLed in unwind)
10448. new buffer with string contents
10449. Allow very small leniency because some compilers won't parse * exact IEEE double constants (happened in matrix testing with * Linux gcc-4.8 -m32 at least).
10450. We could fit built-in index into magic but that'd make the magic * number dependent on built-in numbering (genbuiltins.py doesn't * handle that yet). So map both class and prototype from the * element type.
10451. * Allocate a new thread. ** Leaves the built-ins array uninitialized. The caller must either * initialize a new global context or share existing built-ins from * another thread.
10452. no regexp instance should exist without a non-configurable bytecode property
10453. original input stack before nargs/nregs handling must be * intact for 'arguments' object
10454. replaces top of stack with new value if necessary
10455. 33: setUTCDate
10456. Notes: * - only numbered indices are relevant, so arr_idx fast reject is good * (this is valid unless there are more than 4**32-1 arguments). * - since variable lookup has no side effects, this can be skipped if * DUK_GETDESC_FLAG_PUSH_VALUE is not set.
10457. Automatic semicolon insertion is allowed if a token is preceded * by line terminator(s), or terminates a statement list (right curly * or EOF).
10458. eat 'for'
10459. init 'curr_token'
10460. Index clamping is a bit tricky, we must ensure that we'll only iterate * through elements that exist and that the specific requirements from E5.1 * Sections 15.4.4.14 and 15.4.4.15 are fulfilled; especially: ** - indexOf: clamp to [-len,len], negative handling -> [0,len], * if clamped result is len, for-loop bails out immediately ** - lastIndexOf: clamp to [-len-1, len-1], negative handling -> [-1, len-1], * if clamped result is -1, for-loop bails out immediately ** If fromIndex is not given, ToInteger(undefined) = 0, which is correct * for indexOf() but incorrect for lastIndexOf(). Hence special handling, * and why lastIndexOf() needs to be a vararg function.
10461. implicit attributes
10462. allow a constant to be returned
10463. prop index in 'entry part', < 0 if not there
10464. add_auto_proto
10465. [... ToObject(this) ToUint32(length) val]
10466. Load bytecode instructions.
10467. unsupported: would consume multiple args
10468. DUK_USE_DEBUGGER_THROW_NOTIFY
10469. * Same type? ** Note: since number values have no explicit tag in the 8-byte * representation, need the awkward if + switch.
10470. There's intentionally no check for * current underlying buffer length.
10471. follow parent chain
10472. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT64
10473. **comment:** XXX: this should be an assert
 label: test
10474. without explicit non-BMP support, assume non-BMP characters * are always accepted as letters.
10475. No debugger support, just pop values.
10476. The lookup byte is intentionally sign extended to (at least) * 32 bits and then ORed. This ensures that is at least 1 byte * is negative, the highest bit of 't' will be set at the end * and we don't need to check every byte.
10477. [hobject props enum(props) key desc value? getter? setter?]

10478. start match from beginning
10479. DUK_HOBJECT_FLAG_ARRAY_PART: don't care
10480. Validity checks for various fraction formats ("0.1", ".1", "1.", ".").
10481. DUK_USE_SECTION_B
10482. !DUK_USE_NONSTD_FUNC_CALLER_PROPERTY
10483. **comment:** XXX: better to get base and walk forwards?
 label: code-design
10484. 'ownKeys'
10485. **comment:** Call this.raw(msg); look up through the instance allows user to override * the raw() function in the instance or in the prototype for maximum * flexibility.
 label: code-design
10486. restore stack top
10487. DUK_USE_BASE64_FASTPATH
10488. Valstack should suffice here, required on function valstack init
10489. duk_hnativefunction specific fields.
10490. $r < (*f b 2) [if b==2 -> (*f 4)] * s <- (* (expt b (- 1 e)) 2) == b^{(1-e)} * 2 [if b==2 -> b^{(2-e)}] * m+ <- b == 2 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round$
10491. Insert bytes in the middle of the buffer from an external buffer.
10492. identifier names (E5 Section 7.6)
10493. * Set up the resolution input which is the requested ID directly * (if absolute or no current module path) or with current module * ID prepended (if relative and current module path exists). * Suppose current module is 'foo/bar' and relative path is './quux'. * The 'bar' component must be replaced so the initial input here is * 'foo/bar/././quux'.
10494. A non-Array object should not have an array part in practice. * But since it is supported internally (and perhaps used at some * point), check and abandon if that's the case.
10495. keep heap->dbg_detached_cb
10496. For tv1 == tv2, both pointing to stack top, the end result * is same as duk_pop(ctx).
10497. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.
10498. During parsing, month and day are one-based; set defaults here.
10499. **comment:** * Logger arguments are: * * magic: log level (0-5) * this: logger * stack: plain log args * * We want to minimize memory churn so a two-pass approach * is used: first pass formats arguments and computes final * string length, second pass copies strings either into a * pre-allocated and reused buffer (short messages) or into a * newly allocated fixed buffer. If the backend function plays * nice, it won't coerce the buffer to a string (and thus * intern it).
 label: code-design
10500. Non-zero refcounts should not happen because we refcount * finalize all unreachable objects which should cancel out * refcounts (even for cycles).
10501. token allows automatic semicolon insertion (eof or preceded by newline)
10502. Use special opcodes to load short strings
10503. * Add .fileName and .lineNumber to an error on the stack top.
10504. * CommonJS require() and modules support
10505. * Lexing helpers
10506. * Create built-in objects by parsing an init bitstream generated * by genbuiltins.py.
10507. explicit PropertyList
10508. Special handling for call setup instructions. The target * is expressed indirectly, but there is no output shuffling.
10509. * Create mantissa
10510. NOTE: act may be NULL if an error is thrown outside of any activation, * which may happen in the case of, e.g. syntax errors.
10511. ToNumber() for a fastint is a no-op.
10512. [key] -> [undefined] (default value)
10513. traceback depth fits into 16 bits
10514. [thisArg arg1 ... argN func boundFunc argArray]
10515. * Parser duk__advance() token eating functions
10516. **comment:** cleanup varmap from any null entries, compact it, etc; returns number * of final entries after cleanup.
 label: code-design
10517. **comment:** * Needs a save of multiple saved[] entries depending on what range * may be overwritten. Because the regexp parser does no such analysis, * we currently save the entire saved array here. Lookaheads are thus * a bit expensive. Note that the saved array is not needed for just * the lookahead sub-match, but for the matching of the entire sequel. * * The temporary save buffer is pushed on to the valstack to handle * errors correctly. Each lookahead causes a C recursion and pushes * more stuff on the value stack. If the C recursion limit is less * than the value stack spare, there is no need to check the stack. * We do so regardless, just in case.
 label: code-design
10518. **comment:** XXX: split into separate functions for each field type?
 label: code-design
10519. Same as above but for unsigned int range.
10520. comma after a value, expected
10521. duplicate key
10522. Given a day number, determine year and day-within-year.
10523. **comment:** Vararg function: must be careful to check/require arguments. * The JSON helpers accept invalid indices and treat them like * non-existent optional parameters.
 label: code-design
10524. Success path handles
10525. **comment:** XXX: getter with class check, useful in built-ins
 label: code-design
10526. already updated
10527. checking for DUK__DELETED_MARKER is not necessary here, but helper does it now
10528. match: keep saves
10529. DUK_USE_PARANOIAD_MATH
10530. 'boolean'
10531. * Debugging disabled
10532. fixed offset in valstack
10533. DUK_TOK_ENUM
10534. when key is NULL, value is garbage so no need to set
10535. **comment:** * Error during call. The error value is at heap->lj.value1. * * The very first thing we do is restore the previous setjmp catcher. * This means that any error in error handling will propagate outwards * instead of causing a setjmp() re-entry above.
 label: code-design
10536. Standard behavior for map(): trailing non-existent * elements don't invoke the user callback and are not * counted towards result 'length'.
10537. * Append a Unicode codepoint to the temporary byte buffer. Performs * CESU-8 surrogate pair encoding for codepoints above the BMP. * Existing surrogate pairs are allowed and also encoded into CESU-8.
10538. * Proxy helpers
10539. * Found; post-processing (Function and arguments objects)
10540. maps to finalizer 2nd argument
10541. DUK_USE_AVOID_PLATFORM_FUNCPTRS

10542. 'roundpos' is relative to nc_ctx->k and increases to the right * (opposite of how 'k' changes).
10543. key encountered as a plain property
10544. e.g. DUK_OP_POSTINCV
10545. Allow empty string to be interpreted as 0
10546. Math.pow(-0,y) where y<0 should be: * -Infinity if y<0 and an odd integer * - Infinity otherwise * NetBSD pow() returns -Infinity for all finite y<0. The * if clause also catches y == -Infinity (which works even * without the fix).
10547. **comment:** XXX: add temporary duk__p pointer here too; sharing
label: code-design
10548. not minimum exponent
10549. * Avoid nested calls. Concretely this happens during debugging, e.g. * when we eval() an expression. * * Also don't interrupt if we're currently doing debug processing * (which can be initiated outside the bytecode executor) as this * may cause the debugger to be called recursively. Check required * for correct operation of throw intercept and other "exotic" halting * scenarios.
10550. Because new_size > duk_count_used_probe(heap), guaranteed to work
10551. bump refcount to prevent refzero during finalizer processing
10552. * Pointers to function data area for faster access. Function * data is a buffer shared between all closures of the same * "template" function. The data buffer is always fixed (non- * dynamic, hence stable), with a layout as follows: * * constants (duk_tval) * inner functions (duk_hobject) * bytecode (duk_instr_t) * * Note: bytecode end address can be computed from 'data' buffer * size. It is not strictly necessary functionally, assuming * bytecode never jumps outside its allocated area. However, * it's a safety/robustness feature for avoiding the chance of * executing random data as bytecode due to a compiler error. * * Note: values in the data buffer must be incref'd (they will * be decref'd on release) for every compiledfunction referring * to the 'data' element.
10553. **comment:** * Advance lookup window by N characters, filling in new characters as * necessary. After returning caller is guaranteed a character window of * at least DUK_LEXER_WINDOW_SIZE characters. * * The main function duk_advance_bytes() is called at least once per every * token so it has a major lexer/compiler performance impact. There are two * variants for the main duk_advance_bytes() algorithm: a sliding window * approach which is slightly faster at the cost of larger code footprint, * and a simple copying one. * * Decoding directly from the source string would be another lexing option. * But the lookup window based approach has the advantage of hiding the * source string and its encoding effectively which gives more flexibility * going forward to e.g. support chunked streaming of source from flash. * * Decodes UTF-8/CESU-8 leniently with support for code points from U+0000 to * U+10FFFF, causing an error if the input is unparseable. Leniency means: * * * Unicode code point validation is intentionally not performed, * except to check that the codepoint does not exceed 0x10ffff. * * * In particular, surrogate pairs are allowed and not combined, which * allows source files to represent all SourceCharacters with CESU-8. * Broken surrogate pairs are allowed, as Ecmascript does not mandate * their validation. * * * Allow non-shortest UTF-8 encodings. * * Leniency here causes few security concerns because all character data is * decoded into Unicode codepoints before lexer processing, and is then * re-encoded into CESU-8. The source can be parsed as strict UTF-8 with * a compiler option. However, Ecmascript source characters include -all- * 16-bit unsigned integer codepoints, so leniency seems to be appropriate. * * Note that codepoints above the BMP are not strictly SourceCharacters, * but the lexer still accepts them as such. Before ending up in a string * or an identifier name, codepoints above BMP are converted into surrogate * pairs and then CESU-8 encoded, resulting in 16-bit Unicode data as * expected by Ecmascript. * * An alternative approach to dealing with invalid or partial sequences * would be to skip them and replace them with e.g. the Unicode replacement * character U+FFFD. This has limited utility because a replacement character * will most likely cause a parse error, unless it occurs inside a string. * Further, Ecmascript source is typically pure ASCII. * * See: * * http://en.wikipedia.org/wiki=UTF-8 * http://en.wikipedia.org/wiki/CESU-8 * http://tools.ietf.org/html/rfc3629 * http://en.wikipedia.org/wiki/UTF-8#Invalid_byte_sequences * * Future work: * * * Reject other invalid Unicode sequences (see Wikipedia entry for examples) * in strict UTF-8 mode. * * * Size optimize. An attempt to use a 16-byte lookup table for the first * byte resulted in a code increase though. * * * Is checking against maximum 0x10ffff really useful? 4-byte encoding * imposes a certain limit anyway. * * * Support chunked streaming of source code. Can be implemented either * by streaming chunks of bytes or chunks of codepoints.
label: code-design
10554. duk_push_this() + CheckObjectCoercible() + duk_to_string()
10555. pointers for scanning
10556. array part size (entirely gc reachable)
10557. Steps 12 and 13: reorganize elements to make room for itemCount elements
10558. number of tokens parsed
10559. Tear down state.
10560. Don't check for Infinity unless the context allows it. * 'Infinity' is a valid integer literal in e.g. base-36: * * parseInt('Infinity', 36) * 1461559270678
10561. zero handled by caller
10562. Don't accept relative indices now.
10563. [holder name val] -> [holder]
10564. res_obj
10565. * XUTF-8 and CESU-8 encoding/decoding
10566. 1110 xxxx 10xx xxxx 10xx xxxx [16 bits]
10567. function declaration
10568. **comment:** XXX: disable error handlers for duration of compaction?
label: code-design
10569. * PUTVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is PutValue'd] * 8.7.2 PutValue (V,W) [see especially step 3.b, undefined -> automatic global in non-strict mode] * 8.12.4 [[CanPut]] (P) * 8.12.5 [[Put]] (P) * * Note: may invalidate any valstack (or object) duk_tval pointers because * putting a value may reallocate any object or any valstack. Caller beware.
10570. '?'
10571. state == 3
10572. Maximum traversal depth for "bound function" chains.
10573. **comment:** XXX: magic for getter/setter? use duk_def_prop()?
label: code-design
10574. xxx -> DUK_HBUFFEROBJECT_ELEM_INT32
10575. [thread value]
10576. * GETPROP: Ecmascript property read.
10577. **comment:** * The __FILE__ and __LINE__ information is intentionally not used in the * creation of the error object, as it isn't useful in the tracedata. The * tracedata still contains the function which returned the negative return * code, and having the file/line of this function isn't very useful.
label: code-design
10578. object is a thread (duk_hthread)
10579. 'info'
10580. add a new pending split to the beginning of the entire disjunction
10581. duk_push_this() + CheckObjectCoercible() + duk_to_object()
10582. [... arr jsonarr(res) res] -> [... res jsonarr(arr)]
10583. this may have side effects, so re-lookup act
10584. Fall through to handle the rest.
10585. [... error lineNumber fileName]
10586. * Number built-ins
10587. **comment:** XXX: this compiles to over 500 bytes now, even without special handling * for an array part. Uses signed ints so does not handle full array range correctly.
label: code-design
10588. -> [... closure template env]
10589. Note: if the reg_temp load generated shuffling * instructions, we may need to rewind more than * one instruction, so use explicit PC computation.
10590. 6: toUTCString
10591. **comment:** note: ctx-wide temporary
label: code-design
10592. value popped by call

10593. Note: this may fail, caller should protect the call if necessary
10594. Leaves new timevalue on stack top and returns 1, which is correct * for part setters.
10595. stack[0] = compareFn * stack[1] = ToObject(this) * stack[2] = ToUint32(length)
10596. 4
10597. Boolean/any -> coerce boolean to number and try again. If boolean is * compared to a pointer, the final comparison after coercion now always * yields false (as pointer vs. number compares to false), but this is * not special cased.
10598. s <- 2 * b
10599. native functions: 149
10600. * Range parsing is done with a special lexer function which calls * us for every range parsed. This is different from how rest of * the parsing works, but avoids a heavy, arbitrary size intermediate * value type to hold the ranges. * * Another complication is the handling of character ranges when * case insensitive matching is used (see docs for discussion). * The range handler callback given to the lexer takes care of this * as well. * * Note that duplicate ranges are not eliminated when parsing character * classes, so that canonicalization of * [0-9a-fA-Fx-{}] * creates the result (note the duplicate ranges): * * [0-9A-FA-FX-Z{-{}} * where [x-{} is split as a result of canonicalization. The duplicate * ranges are not a semantics issue: they work correctly.
10601. Although we can allow non-BMP characters (they'll decode * back into surrogate pairs), we don't allow extended UTF-8 * characters; they would encode to URIs which won't decode * back because of strict UTF-8 checks in URI decoding. * (However, we could just as well allow them here.)
10602. exponent 1070
10603. no need to reset temps, as we're finished emitting code
10604. DUK_ERR_ERROR: no macros needed
10605. **comment:** XXX: assumed to fit for now
 label: code-design
10606. [... fallback retval]
10607. **comment:** XXX: Migrate bufwriter and other read/write helpers to its own header?
 label: code-design
10608. recursive call for a primitive value (guaranteed not to cause second * recursion).
10609. Getter might have arbitrary side effects, * so bail out.
10610. * The traceback format is pretty arcane in an attempt to keep it compact * and cheap to create. It may change arbitrarily from version to version. * It should be decoded/accessible through version specific accessors only. * * See doc/error-objects.rst.
10611. The internal _Target property is kept pointing to the original * enumeration target (the proxy object), so that the enumerator * 'next' operation can read property values if so requested. The * fact that the _Target is a proxy disables key existence check * during enumeration.
10612. Reject a proxy object as the handler because it would cause * potentially unbounded recursion. (ES6 has no such restriction)
10613. absolute req_digits; e.g. digits = 1 -> last digit is 0, * but add an extra digit for rounding.
10614. -> [... voidp voidp]
10615. convert duk_compiler_func into a function template, leaving the result * on top of stack.
10616. r <- r (updated above: r <- (remainder (* r B) s) * s <- s * m+ <- m+ (updated above: m+ <- (* m+ B) * m- <- m- (updated above: m- <- (* m- B) * B, low_ok, high_ok are fixed
10617. special helper for emitting u16 lists (used for character ranges for built-in char classes)
10618. [... re_obj input bc saved_buf]
10619. 'reg_varbind' is the operation result and can also * become the expression value for top level assignments * such as: "var x; x += y;".
10620. **comment:** * Three possible outcomes: * * A try or finally catcher is found => resume there. * (or) * * The error propagates to the bytecode executor entry * level (and we're in the entry thread) => rethrow * with a new longjmp(), after restoring the previous * catchpoint. * * The error is not caught in the current thread, so * the thread finishes with an error. This works like * a yielded error, except that the thread is finished * and can no longer be resumed. (There is always a * resumer in this case.) * * Note: until we hit the entry level, there can only be * Ecmascript activations.
 label: code-design
10621. * Number-to-string and string-to-number conversions. * * Slow path number-to-string and string-to-number conversion is based on * a Dragon4 variant, with fast paths for small integers. Big integer * arithmetic is needed for guaranteeing that the conversion is correct * and uses a minimum number of digits. The big number arithmetic has a * fixed maximum size and does not require dynamic allocations. * * See: doc/number-conversion.rst.
10622. Parser separator indices.
10623. Regexp
10624. k > 0 -> k was too low, and cannot be too high
10625. 'func' in the algorithm
10626. **comment:** * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written. * * Note: must re-lookup because calls above (e.g. duk_alloc_entry_checked()) * may realloc and compact properties and hence change e_idx.
 label: code-design
10627. Outside any activation -> look up from global.
10628. Decrease by 25% every round
10629. For manual testing only.
10630. **comment:** Execute the fast path in a protected call. If any error is thrown, * fall back to the slow path. This includes e.g. recursion limit * because the fast path has a smaller recursion limit (and simpler, * limited loop detection).
 label: code-design
10631. * Main heap structure
10632. * Argument exotic [[Delete]] behavior (E5 Section 10.6) is * a post-check, keeping arguments internal 'map' in sync with * any successful deletes (note that property does not need to * exist for delete to 'succeed'). * * Delete key from 'map'. Since 'map' only contains array index * keys, we can use arr_idx for a fast skip.
10633. E5 Section 9.2
10634. Backtrack utf-8 input and return a (possibly canonicalized) input character.
10635. Cause an interrupt after executing one instruction.
10636. Compiler
10637. at least errobj must be on stack
10638. 16 bits
10639. * Default panic handler
10640. thr->heap->dbg_detaching may be != 0 if a debugger write outside * the message loop caused a transport error and detach1() to run.
10641. 'modSearch'
10642. **comment:** The condition could be more narrow and check for the * copy area only, but there's no need for fine grained * behavior when the underlying buffer is misconfigured.
 label: code-design
10643. **comment:** XXX: for a memory-code tradeoff, remove 'func' and make it's access either a function * or a macro. This would make the activation 32 bytes long on 32-bit platforms again.
 label: code-design
10644. Function name is an Identifier (not IdentifierName), but we get * the raw name (not recognizing keywords) here and perform the name * checks only after pass 1.
10645. does not fit into 16 bits
10646. ensures no overflow
10647. [... input buffer]
10648. execution resumes in bytecode executor
10649. Standard JSON omits functions
10650. 7 bits
10651. * Value stack resizing. * * This resizing happens above the current "top": the value stack can be * grown or shrunk, but the "top" is not affected. The value stack cannot * be resized to a size below the current "top". * * The low level reallocation primitive must carefully recompute all value * stack pointers, and must also work if ALL pointers are NULL. The resize * is quite tricky because the valstack realloc may cause a mark-and-sweep, * which may run finalizers. Running

finalizers may resize the valstack * recursively (the same value stack we're working on). So, after realloc * returns, we know that the valstack "top" should still be the same (there * should not be live values above the "top"), but its underlying size and * pointer may have changed.

10652. * Objects have internal references. Must finalize through * the "refzero" work list.

10653. different thread

10654. number of values in current MPUTARR set

10655. steps 13-14

10656. Compute result length and validate argument buffers.

10657. DUK_BUILTINS_H_INCLUDED

10658. copy bytecode instructions one at a time

10659. **comment:** E5.1 Section 15.2.4.6, step 3.a, lookup proto once before compare. * Prototype loops should cause an error to be thrown.

label: code-design

10660. [... RegExp val] -> [... res]

10661. load factor max 75%

10662. [value offset noAssert], when ftype != DUK_FLD_VARINT

10663. * Resizing

10664. token closes expression, e.g. ')', ']'

10665. fileName

10666. * Get prototype object for an integer error code.

10667. '\ufffNext'

10668. PC/line semantics here are: * - For callstack top we're conceptually between two * opcodes and current PC indicates next line to * execute, so report that (matches Status). * - For other activations we're conceptually still * executing the instruction at PC-1, so report that * (matches error stacktrace behavior). * - See: <https://github.com/svaarala/duktape/issues/281>

10669. Don't allow negative zero as it will cause trouble with * LDINT+LDINTX. But positive zero is OK.

10670. borrow literal Infinity from builtin string

10671. **comment:** XXX: This now coerces an identifier into a GETVAR to a temp, which * causes an extra LDREG in call setup. It's sufficient to coerce to a * unary ivalue?

label: code-design

10672. prefer direct execution

10673. **comment:** Longjmp state is cleaned up by error handling

label: code-design

10674. **comment:** spaces[] must be static to allow initializer with old compilers like BCC

label: code-design

10675. * Object.prototype.hasOwnProperty() and Object.prototype.propertyIsEnumerable().

10676. Ordinary gap, undefined encodes to 'null' in * standard JSON (and no JX/JC support here now).

10677. fall-through jump to next code of next case (backpatched)

10678. already one-based

10679. **comment:** XXX: could use at least one fewer loop counters

label: code-design

10680. Create function 'data' buffer but don't attach it yet.

10681. * Function built-ins

10682. * No match, E5 Section 15.10.6.2, step 9.a.i - 9.a.ii apply, regardless * of 'global' flag of the RegExp. In particular, if lastIndex is invalid * initially, it is reset to zero.

10683. **comment:** hacky helper for String.prototype.split()

label: code-design

10684. [(builtin objects) name func]

10685. We request a tail call, but in some corner cases * call handling can decide that a tail call is * actually not possible. * See: test-bug-tailcall-preventyield-assert.c.

10686. toString() conditions

10687. Write bytecode executor's curr_pc back to topmost activation (if any).

10688. [] -> [val]

10689. **comment:** XXX: Extensibility check for target uses IsExtensible(). If we * implemented the isExtensible trap and didn't reject proxies as * proxy targets, it should be respected here.

label: code-design

10690. **comment:** Technically return value is not needed because INVLHS will * unconditionally throw a ReferenceError. Coercion is necessary * for proper semantics (consider ToNumber() called for an object). * Use DUK_EXTRAOP_UNP with a dummy register to get ToNumber().

label: code-design

10691. decl type

10692. -> [sep ToObject(this) len sep str]

10693. Shared lenient buffer length clamping helper. Indices are treated as * element indices (though output values are byte offsets) which only * really matters for TypedArray views as other buffer object have a zero * shift. Negative indices are counted from end of input slice; crossed * indices are clamped to zero length; and final indices are clamped * against input slice. Used for e.g. ArrayBuffer slice().

10694. XXX: perhaps refactor this to allow caller to specify some parameters, or * at least a 'compact' flag which skips any spare or round-up .. useful for * emergency gc.

10695. [...] enum_target res handler trap]

10696. XXX: Decrementing and restoring act->curr_pc works now, but if the * debugger message loop gains the ability to adjust the current PC * (e.g. a forced jump) restoring the PC here will break. Another * approach would be to use a state flag for the "decrement 1 from * topmost activation's PC" and take it into account whenever dealing * with PC values.

10697. Create a fresh object environment for the global scope. This is * needed so that the global scope points to the newly created RAM-based * global object.

10698. x <- (1<<y)

10699. * HASPROP: EcmaScript property existence check ("in" operator). * * Interestingly, the 'in' operator does not do any coercion of * the target object.

10700. Any 'res' will do.

10701. If neutered must return 0; length is zeroed during * neutering.

10702. XXX: assert idx_base

10703. **comment:** XXX: refactor into internal helper, duk_clone_hobject()

label: code-design

10704. * Get starting character offset for match, and initialize 'sp' based on it. * * Note: lastIndex is non-configurable so it must be present (we check the * internal class of the object above, so we know it is). User code can set * its value to an arbitrary (garbage) value though; E5 requires that lastIndex * be coerced to a number before using. The code below works even if the * property is missing: the value will then be coerced to zero. * * Note: lastIndex may be outside Uint32 range even after ToInteger() coercion. * For instance, ToInteger(+Infinity) = +Infinity. We track the match offset * as an integer, but pre-check it to be inside the 32-bit range before the loop. * If not, the check in E5 Section 15.10.6.2, step 9.a applies.

10705. avoid side effect issues

10706. * Setup call: target and 'this' binding. Three cases: * * 1. Identifier base (e.g. "foo()") * 2. Property base (e.g. "foo.bar()") * 3. Register base (e.g. "foo()0"; i.e. when a return value is a function)

10707. SHIFT EXPRESSION

10708. overflow, fall through

10709. restored if ecma-to-ecma setup fails

10710. Match labels starting from latest label because there can be duplicate empty * labels in the label set.

10711. If debugger is paused, garbage collection is disabled by default.

10712. DUK_ERR_REFERENCE_ERROR: no macros needed

10713. XXX: shared helper for duk_push_hobject_or_undefined()

10714. * Strings have no internal references but do have "weak" * references in the string cache. Also note that strings * are not on the heap_allocated list like other heap * elements.
10715. unconditionally
10716. **comment:** XXX: At the moment Duktape accesses internal keys like _Finalizer using a * normal property set/get which would allow a proxy handler to interfere with * such behavior and to get access to internal key strings. This is not a problem * as such because internal key strings can be created in other ways too (e.g. * through buffers). The best fix is to change Duktape internal lookups to * skip proxy behavior. Until that, internal property accesses bypass the * proxy and are applied to the target (as if the handler did not exist). * This has some side effects, see test-bi-proxy-internal-keys.js.
label: code-design
10717. Get local time offset (in seconds) for a certain (UTC) instant 'd'.
10718. **comment:** Can be called multiple times with no harm. Mark the transport * bad (dbg_read_cb == NULL) and clear state except for the detached * callback and the udata field. The detached callback is delayed * to the message loop so that it can be called between messages; * this avoids corner cases related to immediate debugger reattach * inside the detached callback.
label: code-design
10719. resume
10720. * Parse a function body or a function-like expression, depending * on flags.
10721. actually used, non-NULL entries
10722. Fast path check.
10723. length check
10724. Do decrefs only with safe pointers to avoid side effects * disturbing e_idx.
10725. '{"_ninf":true}'
10726. shared checks for all descriptor types
10727. There's no need to check for buffer validity status for the * target here: the property access code will do that for each * element. Moreover, if we did check the validity here, side * effects from reading the source argument might invalidate * the results anyway.
10728. * Process space (3rd argument to JSON.stringify)
10729. Difference to non-strict/strict comparison is that NaNs compare * equal and signed zero signs matter.
10730. DUK_USE_PANIC_HANDLER
10731. avoid array abandoning which interns strings
10732. traits are separate; in particular, arguments not an array
10733. duk_get_min_grow_a() is always >= 1
10734. 9: getUTCFullYear
10735. [key getter this key] -> [key retval]
10736. Source end clamped silently to available length.
10737. First character has already been eaten and checked by the caller. * We can scan until a NUL in stridx string because no built-in strings * have internal NULs.
10738. * Proxy built-in (ES6)
10739. Native function, no relevant lineNumber.
10740. * Limited functionality bigint implementation. * * Restricted to non-negative numbers with less than 32 * DUK_BI_MAX_PARTS bits, * with the caller responsible for ensuring this is never exceeded. No memory * allocation (except stack) is needed for bigint computation. Operations * have been tailored for number conversion needs. * * Argument order is "assignment order", i.e. target first, then arguments: * x <- y * z --> duk_bi_mul(x, y, z);
10741. * InitJS code - Ecmascript code evaluated from a built-in source * which provides e.g. backward compatibility. User can also provide * JS code to be evaluated at startup.
10742. 'res' must be a plain ivalue, and not register-bound variable.
10743. function helpers
10744. 'Null'
10745. internal properties
10746. custom
10747. * XXX: shrink array allocation or entries compaction here?
10748. ToInteger(lastIndex)
10749. * Encoding/decoding helpers
10750. default: NaN
10751. covers -Infinity
10752. Error instance, use augmented error data directly
10753. t1 = (+ r m+)
10754. [... func arg1 ... argN]
10755. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small; some overlap with string * handling.
label: code-design
10756. Some contexts don't allow fractions at all; this can't be a * post-check because the state ('f' and expt) would be incorrect.
10757. variable value (function, we hope, not checked here)
10758. return 'res_obj'
10759. E5 Section 10.4.2
10760. DUK_FORWDECL_H_INCLUDED
10761. avoid unary minus on unsigned
10762. Just flip the single bit.
10763. unicode code points, window[0] is always next, points to 'buffer'
10764. probe
10765. may be set to 0 by duk_debugger_attach() inside callback
10766. Easy to get wrong, so assert for it.
10767. mask out flags not actually stored
10768. 'with' target must be created first, in case we run out of memory
10769. **comment:** Slow writing of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. There's * no special sign handling when writing varints.
label: code-design
10770. Important main primitive.
10771. no throw
10772. [...] lval rval(func)
10773. **comment:** XXX: 'this' will be ToObject() coerced twice, which is incorrect * but should have no visible side effects.
label: code-design
10774. **comment:** XXX: The duk_to_number() cast followed by integer coercion * is platform specific so NaN, +/- Infinity, and out-of-bounds * values result in platform specific output now. * See: test-bi-nodejs-buffer-proto-varint-special.js
label: code-design
10775. 'lastIndex'
10776. 0xd0-0xdf
10777. * Format tag parsing. Since we don't understand all the * possible format tags allowed, we just scan for a terminating * specifier and keep track of relevant modifiers that we do * understand. See man 3 printf.
10778. pop varname
10779. **comment:** XXX: this should probably just operate on the stack top, because it * needs to push stuff on the stack anyway...
label: code-design
10780. 'advtok' indicates how much to advance and which token id to assign * at the end. This shared functionality minimizes code size. All * code paths are required to set 'advtok' to some value, so no default * init value is used. Code paths calling DUK_ERROR() never return so * they don't need to set advtok.

10781. debugger
10782. * Automatically generated by extract_chars.py, do not edit!
10783. * Copy keys and values in the entry part (compacting them at the same time).
10784. expensive init, just want to fill window
10785. 'toLogString'
10786. split2: prefer jump execution (not direct)
10787. Zero special case: fake requested number of zero digits; ensure * no sign bit is printed. Relative and absolute fixed format * require separate handling.
10788. skip line info
10789. heap destruction ongoing, finalizer rescue no longer possible
10790. E5 Section 9.1
10791. value stack indices for tracking objects
10792. two encoding attempts suffices
10793. DUK_ISPEC_XXX
10794. **comment:** XXX: append primitive
 label: code-design
10795. current lexical environment (may be NULL if delayed)
10796. explicit NULL inits
10797. [... closure template env]
10798. use a temp: decref only when valstack reachable values are correct
10799. Helper for string conversion calls: check 'this' binding, get the * internal time value, and format date and/or time in a few formats. * Return value allows tail calls.
10800. 0 or -1
10801. **comment:** * Allocate heap struct ** Use a raw call, all macros expect the heap to be initialized
 label: code-design
10802. Object.defineProperty() equivalent C binding.
10803. skip jump slots
10804. We can directly access value stack here.
10805. DUK_TOK_TYPEOF
10806. mark as complex
10807. XXX: handling of timestamps outside Windows supported range. * How does Windows deal with dates before 1600? Does windows * support all Ecmascript years (like -200000 and +200000)? * Should equivalent year mapping be used here too? If so, use * a shared helper (currently integrated into timeval-to-parts).
10808. **comment:** m+ != m- (very rarely)
 label: code-design
10809. * Duktape debugger
10810. Ecmascript activation + Duktape.Thread.resume() activation
10811. \"
10812. [... str]
10813. -> [... value]
10814. Use temporaries and update lex_ctx only when finished.
10815. mark-and-sweep not running -> must be empty
10816. **comment:** * Normal error ** Normal error is thrown with a longjmp() through the current setjmp() * catchpoint record in the duk_heap. The 'curr_thread' of the duk_heap * identifies the throwing thread. ** Error formatting is usually unnecessary. The error macros provide a * zero argument version (no formatting) and separate macros for small * argument counts. Variadic macros are not used to avoid portability * issues and avoid the need for stash-based workarounds when they're not * available. Vararg calls are avoided for non-formatted error calls * because vararg call sites are larger than normal, and there are a lot * of call sites with no formatting. ** Note that special formatting provided by debug macros is NOT available. ** The _RAW variants allow the caller to specify file and line. This makes * it easier to write checked calls which want to use the call site of the * checked function, not the error macro call inside the checked function.
 label: code-design
10817. **comment:** Fast canonicalization lookup at the cost of 128kB footprint.
 label: code-design
10818. DUK_TOK_EXPORT
10819. array of active label names
10820. * Helper for C function call negative return values.
10821. entry_top + 0
10822. * Thread switching ** To switch heap->curr_thread, use the macro below so that interrupt counters * get updated correctly. The macro allows a NULL target thread because that * happens e.g. in call handling.
10823. entry allocation updates hash part and increases the key * refcount; may need a props allocation resize but doesn't * 'recheck' the valstack.
10824. Not necessary to unwind catchstack: break/continue * handling will do it. The finally flag of 'cat' is * no longer set. The catch flag may be set, but it's * not checked by break/continue handling.
10825. Shared helper.
10826. x1 must be a string
10827. [... value? getter? setter?]
10828. **comment:** XXX: the insert here is a bit expensive if there are a lot of items. * It could also be special cased in the outermost for loop quite easily * (as the element is dup()'d anyway).
 label: code-design
10829. Gather in big endian
10830. **comment:** SCANBUILD: warning about 'thr' potentially being NULL here, * warning is incorrect because thr != NULL always here.
 label: code-design
10831. format is bit packed
10832. See test-bug-netbsd-math-pow.js: NetBSD 6.0 on x86 (at least) does not * correctly handle some cases where x=+-0. Specific fixes to these * here.
10833. parse args starting from "next temp", reg_target + 1
10834. Note: multiple threads may be simultaneously in the RUNNING * state, but not in the same "resume chain".
10835. * Named function expression, name needs to be bound * in an intermediate environment record. The "outer" * lexical/variable environment will thus be: ** a) { funcname: <func>, __prototype: outer_lex_env } * b) { funcname: <func>, __prototype: <globalenv> } (if outer_lex_env missing)
10836. * x is finite and non-zero ** -1.6 -> floor(-1.1) -> -2 * -1.5 -> floor(-1.0) -> -1 (towards +Inf) * -1.4 -> floor(-0.9) -> -1 * -0.5 -> -0.0 (special case) * -0.1 -> -0.0 (special case) * +0.1 -> +0.0 (special case) * +0.5 -> floor(+1.0) -> 1 (towards +Inf) * +1.4 -> floor(+1.9) -> 1 * +1.5 -> floor(+2.0) -> 2 (towards +Inf) * +1.6 -> floor(+2.1) -> 2
10837. * Context init
10838. **comment:** XXX: rework
 label: code-design
10839. Call stack. [0,callstack_top[is GC reachable.
10840. DUK_NUMCONV_H_INCLUDED
10841. is a function declaration (as opposed to function expression)
10842. Precomputed pointers when using 16-bit heap pointer packing.
10843. The E5.1 Section 15.4.4.4 algorithm doesn't set the length explicitly * in the end, but because we're operating with an internal value which * is known to be an array, this should be equivalent.
10844. XXX: Allow customizing the pause and notify behavior at runtime * using debugger runtime flags. For now the behavior is fixed using * config options.
10845. = IsAccessorDescriptor(Desc)
10846. 36: setFullYear
10847. **comment:** not covered, return all zeroes

label: test

10848. * Unicode range matcher * * Matches a codepoint against a packed bitstream of character ranges. * Used for slow path Unicode matching.

10849. DUK_HTHREAD_H_INCLUDED

10850. DUK_USE_ES6_PROXY

10851. Located in duk_heap.h_extra16. Subclasses of duk_hobject (like * duk_hcompiledfunction) are not free to use h_extra16 for this reason.

10852. When looking for .fileName./lineNumber, blame compilation * or C call site unless flagged not to do so.

10853. Pretend like we got EOM

10854. [regexp source bytecode]

10855. catch depth at point of definition

10856. -> [... cons target]

10857. Grow entry part allocation for one additional entry.

10858. **comment:** * Calls. * * Protected variants should avoid ever throwing an error.**label:** code-design10859. **comment:** XXX: better multiline**label:** code-design

10860. object literal key tracking flags

10861. curr is accessor -> cannot be in array part

10862. stmt has explicit/implicit semicolon terminator

10863. lexing

10864. [... result/error]

10865. catches EOF (NUL) and initial comma

10866. **comment:** If no explicit message given, put error code into message field * (as a number). This is not fully in keeping with the EcmaScript * error model because messages are supposed to be strings (Error * constructors use ToString() on their argument). However, it's * probably more useful than having a separate 'code' property.**label:** code-design

10867. The #ifdef clutter here needs to handle the three cases: * (1) JX+JC, (2) JX only, (3) JC only.

10868. currently running thread

10869. temproots

10870. Tail call check: if last opcode emitted was CALL(I), and * the context allows it, change the CALL(I) to a tail call. * This doesn't guarantee that a tail call will be allowed at * runtime, so the RETURN must still be emitted. (Duktape * 0.10.0 avoided this and simulated a RETURN if a tail call * couldn't be used at runtime; but this didn't work * correctly with a thread yield/resume, see * test-bug-tailcall-thread-yield-resume.js for discussion.) * * In addition to the last opcode being CALL, we also need to * be sure that 'rc_val' is the result register of the CALL(I). * For instance, for the expression 'return 0, (function () * { return 1; }), 2' the last opcode emitted is CALL (no * bytecode is emitted for '2') but 'rc_val' indicates * constant '2'. Similarly if '2' is replaced by a register * bound variable, no opcodes are emitted but tail call would * be incorrect. * * This is tricky and easy to get wrong. It would be best to * track enough expression metadata to check that 'rc_val' came * from that last CALL instruction. We don't have that metadata * now, so we check that 'rc_val' is a temporary register result * (not a constant or a register bound variable). There should * be no way currently for 'rc_val' to be a temporary for an * expression following the CALL instruction without emitting * some opcodes following the CALL. This proxy check is used * below. * * See: test-bug-comma-expr-gh131.js. * * The non-standard 'caller' property disables tail calls * because they pose some special cases which haven't been * fixed yet.

10871. * Helper: handle Array object 'length' write which automatically * deletes properties, see E5 Section 15.4.5.1, step 3. This is * quite tricky to get right. * * Used by duk_hobject_putprop().

10872. vararg

10873. Use 'res' as the expression value (it's side effect * free and may be a plain value, a register, or a * constant) and write it to the LHS binding too.

10874. Allow NULL 'msg'

10875. borrowed reference

10876. [... s1 s2] -> [... s1+s2]

10877. DUK_TOK_DIV

10878. **comment:** XXX: lithuanian not implemented**label:** requirement

10879. Explicit zero size check to avoid NULL 'tv1'.

10880. * Get holder object

10881. XXX: can shift() / unshift() use the same helper? * shift() is (close to?) <-> splice(0, 1) * unshift is (close to?) <-> splice(0, 0, [items])?

10882. lead zero + 'digits' fractions + 1 for rounding

10883. stack discipline consistency check

10884. * Update the interrupt counter

10885. these non-standard properties are copied for convenience

10886. still reachable

10887. * Forward declarations for all Duktape structures.

10888. [... obj key]

10889. **comment:** * Object.preventExtensions() and Object.isExtensible() (E5 Sections 15.2.3.10, 15.2.3.13) * * Not needed, implemented by macros DUK_HOBJECT_{HAS,CLEAR,SET}_EXTENSIBLE * and the Object built-in bindings.**label:** code-design

10890. * Macros for property handling

10891. val2 = min count, val1 = max count

10892. find and remove string from stringtable; caller must free the string itself

10893. Inner functions recursively.

10894. == DUK_MAX_TEMPS is OK

10895. Don't free h->resumer because it exists in the heap. * Callstack entries also contain function pointers which * are not freed for the same reason.

10896. use bigint area as a temp

10897. 'n'

10898. * Allocate initial stacks for a thread. Note that 'thr' must be reachable * as a garbage collection may be triggered by the allocation attempts. * Returns zero (without leaking memory) if init fails.

10899. step 4.b

10900. (Number.MAX_VALUE).toString(2).length == 1024, + spare

10901. * Check whether already declared. * * We need to check whether the binding exists in the environment * without walking its parents. However, we still need to check * register-bound identifiers and the prototype chain of an object * environment target object.

10902. Catch stack. [0,catchstack_top[is GC reachable.

10903. DUK_USE_DATE_TZO_WINDOWS

10904. -> [... res_obj]

10905. * E5 Section 15.10.2.6. The previous and current character * should -not- be canonicalized as they are now. However, * canonicalization does not affect the result of IsWordChar() * (which depends on Unicode characters never canonicalizing * into ASCII characters) so this does not matter.

10906. flags is unsigned

10907. 0

10908. **comment:** XXX: better place for this**label:** code-design

10909. Zero encoded pointer is required to match NULL

10910. Given a (year, month, day-within-month) triple, compute day number. * The input triple is un-normalized and may contain non-finite values.

10911. upper limit, assuming no whitespace etc

10912. Copy slice, respecting underlying buffer limits; remainder * is left as zero.

10913. **comment:** not necessary to init, disabled for faster parsing
label: code-design

10914. We already ate 'x', so backup one byte.

10915. maximum token count before error (sanity backstop)

10916. * For ASCII strings, the answer is simple.

10917. 'super'

10918. **comment:** XXX: This won't be shown in practice now * because this code is not run when builtins * are in ROM.
label: code-design

10919. 'JSON'

10920. combine left->x1 and res->x1 (right->x1, really) -> (left->x1 OP res->x1)

10921. We can't reliably pop anything here because the stack input * shape is incorrect. So we throw an error; if the caller has * no catch point for this, a fatal error will occur. Another * alternative would be to just return an error. But then the * stack would be in an unknown state which might cause some * very hard to diagnose problems later on. Also note that even * if we did not throw an error here, the underlying call handler * might STILL throw an out-of-memory error or some other internal * fatal error.

10922. marker: copy t if not changed

10923. thread has terminated

10924. * Context management

10925. Return codes for protected calls (duk_safe_call(), duk_pcall())

10926. RangeError

10927. set/clear writable

10928. DUK_USE_64BIT_OPS

10929. Ecmascript date range is 100 million days from Epoch: * > 100e6 * 24 * 60 * 60 * 1000 // 100M days in millisecs * 8640000000000000 * (= 8.64e15)

10930. * Coercion operations: in-place coercion, return coerced value where * applicable. If index is invalid, throw error. Some coercions may * throw an expected error (e.g. from a toString() or valueOf() call) * or an internal error (e.g. from out of memory).

10931. packed tval

10932. (internal) catch compilation errors

10933. * END PUBLIC API

10934. E == 0x7ff, topmost four bits of F != 0 => assume NaN

10935. APIError

10936. * BEGIN PUBLIC API

10937. timeval breakdown: internal time value NaN -> RangeError (toISOString)

10938. internal flag: external buffer

10939. enumerate internal properties (regardless of enumerability)

10940. Ecmascript E5 specification error codes

10941. * Debugger (debug protocol)

10942. lstring

10943. timeval breakdown: internal time value NaN -> zero

10944. include time part in string conversion result

10945. prefer number

10946. Note: parentheses are required so that the comma expression works in assignments.

10947. * Duktape public API for Duktape 1.5.2. * * See the API reference for documentation on call semantics. * The exposed API is inside the DUK_API_PUBLIC_H_INCLUDED * include guard. Other parts of the header are Duktape * internal and related to platform/compiler/feature detection. * * Git commit cad34ae155acb0846545ca6bf2d29f9463b22bbb (v1.5.2). * Git branch HEAD. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.

10948. internal flag value: throw if mask doesn't match

10949. set getter (given on value stack)

10950. no error if file does not exist

10951. * Memory management * * Raw functions have no side effects (cannot trigger GC).

10952. Coercion hints

10953. is_copy

10954. * Variable access

10955. weekday: 0 to 6, 0=sunday, 1=monday, etc

10956. DUK_DBLEUNION_H_INCLUDED

10957. lightweight function pointer

10958. * String manipulation

10959. getter: subtract 1900 from year when getting year part

10960. Value mask types, used by e.g. duk_get_type_mask()

10961. Ecmascript undefined

10962. file

10963. Log levels

10964. Ecmascript boolean: 0 or 1

10965. * Public API specific typedefs * * Many types are wrapped by Duktape for portability to rare platforms * where e.g. 'int' is a 16-bit type. See practical typing discussion * in Duktape web documentation.

10966. flags

10967. Assertion Error

10968. only enumerate array indices

10969. * Stack management

10970. Flags for duk_def_prop() and its variants

10971. NOTE: when writing a Date provider you only need a few specific * flags from here, the rest are internal. Avoid using anything you * don't need.

10972. * Date provider related constants * * NOTE: These are "semi public" - you should only use these if you write * your own platform specific Date provider, see doc/datetime.rst.

10973. internal: request fixed buffer result

10974. Number of value stack entries (in addition to actual call arguments) * guaranteed to be allocated on entry to a Duktape/C function.

10975. * Indexes of various types with respect to big endian (logical) layout

10976. no error (e.g. from duk_get_error_code())

10977. LICENSE.txt

10978. SyntaxError

10979. * Duktape/C function magic value

10980. internal: request dynamic buffer result

10981. * Avoid C++ name mangling

10982. **comment:** XXX: replace with TypeError?
label: code-design

10983. **comment:** XXX: to be removed?
label: code-design

10984. Git commit, describe, and branch for Duktape build. Useful for * non-official snapshot builds so that application code can easily log * which Duktape snapshot was used. Not available in the Ecmascript * environment.

10985. DUK_API_PUBLIC_H_INCLUDED

10986. Compilation flags for duk_compile() and duk_eval()

10987. timeval breakdown: convert month and day-of-month parts to one-based (default is zero-based)
10988. Duktape version, (major * 10000) + (minor * 100) + patch. Allows C code * to #ifdef against Duktape API version. The same value is also available * to Ecmascript code in Duktape.version. Unofficial development snapshots * have 99 for patch level (e.g. 0.10.99 would be a development version * after 0.10.0 but before the next official release).
10989. * Buffer
10990. set enumerable (effective if DUK_DEFPROP_HAVE_ENUMERABLE set)
10991. set configurable (effective if DUK_DEFPROP_HAVE_CONFIGURABLE set)
10992. don't walk prototype chain, only check own properties
10993. UnimplementedError
10994. (internal) omit eval result
10995. **comment:** * If no variadic macros, __FILE__ and __LINE__ are passed through globals * which is ugly and not thread safe.
 label: requirement
10996. last value is func pointer, arguments follow in parens
10997. set/clear configurable
10998. Return codes for C functions (shortcut for throwing an error)
10999. * Pop operations
11000. * Require operations: no coercion, throw error if index or type * is incorrect. No defaulting.
11001. * Bytecode load/dump
11002. compile function code (instead of global code)
11003. enumerate a proxy object itself without invoking proxy behavior
11004. **comment:** XXX: native 64-bit byteswaps when available
 label: code-design
11005. **comment:** XXX: These calls are incomplete and not usable now. They are not (yet) * part of the public API.
 label: code-design
11006. Error
11007. DUK_USE_FILE_IO
11008. ReferenceError
11009. timeval breakdown: replace year with equivalent year in the [1971,2037] range for DST calculations
11010. * Ecmascript operators
11011. args
11012. * Some defines forwarded from feature detection
11013. (internal) take strlen() of src_buffer (avoids double evaluation in macro)
11014. set/clear enumerable
11015. internal use
11016. * Error handling
11017. EvalError
11018. * Object prototype
11019. setter: perform 2-digit year fixup (00...99 -> 1900...1999)
11020. nop: no need to normalize
11021. * Compilation and evaluation
11022. or is a normalized NaN
11023. AllocError
11024. month: 0 to 11
11025. internal flag: create backing ArrayBuffer; keep in one byte
11026. additional values begin at bit 12
11027. * Constants
11028. **comment:** XXX: replace with plain Error?
 label: code-design
11029. * Object finalizer
11030. Reverse operation is the same.
11031. duk_context is now defined in duk_config.h because it may also be * referenced there by prototypes.
11032. * C++ name mangling
11033. UncaughtError
11034. Ecmascript year range: * > new Date(100e6 * 24 * 3600e3).toISOString() * '+275760-09-13T00:00:00.000Z' * > new Date(-100e6 * 24 * 3600e3).toISOString() * '-271821-04-20T00:00:00.000Z'
11035. create a new global environment
11036. * Global object
11037. DUKTAPE_H_INCLUDED
11038. E == 0x7ff, F != 0 => NaN
11039. DUK_USE_PACKED_TVAL
11040. * Logging
11041. safe variants of a few coercion operations
11042. Indicates that a native function does not have a fixed number of args, * and the argument stack should not be capped/extended at all.
11043. * Get operations: no coercion, returns default value for invalid * indices and invalid value types. * * duk_get_undefined() and duk_get_null() would be pointless and * are not included.
11044. **comment:** use locale specific formatting if available
 label: code-design
11045. External duk_config.h provides platform/compiler/OS dependent * typedefs and macros, and DUK_USE_xxx config options so that * the rest of Duktape doesn't need to do any feature detection.
11046. internal: don't care about fixed/dynamic nature
11047. * Property access * * The basic function assumes key is on stack. The _string variant takes * a C string as a property name, while the _index variant takes an array * index as a property name (e.g. 123 is equivalent to the key "123").
11048. * Debugging
11049. 64-bit byteswap, same operation independent of target endianness.
11050. * Type checks * * duk_is_none(), which would indicate whether index is outside of stack, * is not needed; duk_is_valid_index() gives the same information.
11051. sort array indices, use with DUK_ENUM_ARRAY_INDICES_ONLY
11052. force change if possible, may still fail for e.g. virtual properties
11053. Millisecond count constants.
11054. used by packed duk_tval, assumes sizeof(void *) == 4
11055. set value (given on value stack)
11056. Used to represent invalid index; if caller uses this without checking, * this index will map to a non-existent stack entry. Also used in some * API calls as a marker to denote "no value".
11057. * Module helpers: put multiple function or constant properties
11058. raw void pointer
11059. Flags for duk_push_thread_raw()
11060. * ===== * Duktape authors * ===== * * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people

have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u000e1szl\u00f3 Lang \u00f3f3 <llango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6tzte (<https://github.com/jaseg>) * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6dman * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstruktrup> * * Michael Drake (<https://github.com/ltsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn\u00e5s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.

11061. InternalError
11062. Value types, used by e.g. duk_get_type()
11063. **comment:** Duktape specific error codes (must be 8 bits at most, see duk_error.h)
label: code-design
11064. no value, e.g. invalid index
11065. * Stack manipulation (other than push/pop)
11066. Byteswap an IEEE double in the duk_double_union from host to network * order. For a big endian target this is a no-op.
11067. prefer string
11068. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2016 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
11069. external use
11070. Flags for duk_push_string_file_raw()
11071. Duktape debug protocol version used by this build.
11072. **comment:** DUK_COMPILE_xxx bits 0-2 are reserved for an internal 'nargs' argument * (the nargs value passed is direct stack arguments + 1 to account for an * internal extra argument).
label: code-design
11073. fixed or dynamic, garbage collected byte buffer
11074. * Thread management
11075. * Double NaN manipulation macros related to NaN normalization needed when * using the packed duk_tval representation. NaN normalization is necessary * to keep double values compatible with the duk_tval format. * * When packed duk_tval is used, the NaN space is used to store pointers * and other tagged values in addition to NaNs. Actual NaNs are normalized * to a specific quiet NaN. The macros below are used by the implementation * to check and normalize NaN values when they might be created. The macros * are essentially NOPs when the non-packed duk_tval representation is used. * * A FULL check is exact and checks all bits. A NOTFULL check is used by * the packed duk_tval and works correctly for all NaNs except those that * begin with 0x7ff0. Since the 'normalized NaN' values used with packed * duk_tval begin with 0x7ff8, the partial check is reliable when packed * duk_tval is used. The 0x7ff8 prefix means the normalized NaN will be a * quiet NaN regardless of its remaining lower bits. * * The ME variant below is specifically for ARM byte order, which has the * feature that while doubles have a mixed byte order (32107654), unsigned * long long values has a little endian byte order (76543210). When writing * a logical double value through a ULL pointer, the 32-bit words need to be * swapped; hence the #ifdefs below for ULL writes with DUK_USE_DOUBLE_ME. * This is not full ARM support but suffices for some environments.
11076. either not a NaN
11077. convert time value to local time
11078. not directly applicable, byte order differs from a double
11079. end 'extern "C"' wrapper
11080. * Object operations
11081. set writable (effective if DUK_DEFPROP_HAVE_WRITABLE set)
11082. UnsupportedError
11083. **comment:** Internal API call flags, used for various functions in this file. * Certain flags are used by only certain functions, but since the flags * don't overlap, a single flags value can be passed around to multiple * functions. * * The unused top bits of the flags field are also used to pass values * to helpers (duk_get_part_helper() and duk_set_part_helper()). * * (Must be in-sync with genbuiltins.py.)
label: code-design
11084. There are currently no native functions to yield/resume, due to the internal * limitations on coroutine handling. These will be added later.
11085. Concrete macros for NaN handling used by the implementation internals. * Chosen so that they match the duk_tval representation: with a packed * duk_tval, ensure NaNs are properly normalized; with a non-packed duk_tval * these are essentially NOPs.
11086. (internal) no source string on stack
11087. * Function (method) calls
11088. plain
11089. Ecmascript string: CESU-8 / extended UTF-8 encoded
11090. internal flag: don't zero allocated buffer
11091. use strict (outer) context for global, eval, or function code
11092. in non-packed representation we don't care about which NaN is used
11093. * Union for accessing double parts, also serves as packed duk_tval
11094. include date part in string conversion result
11095. internal flag: dynamic buffer
11096. Ecmascript null
11097. enumerate non-enumerable properties in addition to enumerable
11098. **comment:** XXX: replace with RangeError?
label: code-design
11099. * Push operations * * Push functions return the absolute (relative to bottom of frame) * position of the pushed value for convenience. * * Note: duk_dup() is technically a push.
11100. string
11101. * ROM pointer compression
11102. * Union to access IEEE double memory representation, indexes for double * memory representation, and some macros for double manipulation. * * Also used by packed duk_tval. Use a union for bit manipulation to * minimize aliasing issues in practice. The C99 standard does not * guarantee that this should work, but it's a very widely supported * practice for low level manipulation. * * IEEE double format summary: * * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * See http://en.wikipedia.org/wiki/Double_precision_floating-point_format. * * NaNs are represented as exponent 0x7ff and mantissa != 0. The NaN is a * signaling NaN when the highest bit of the mantissa is zero, and a quiet * NaN when the highest bit is set. * * At least three memory layouts are relevant here: * * A B C D E F G H Big endian (e.g. 68k) DUK_USE_DOUBLE_BE * H G F E D C B A Little endian (e.g. x86) DUK_USE_DOUBLE_LE * D C B A H G F E Mixed/cross endian (e.g. ARM) DUK_USE_DOUBLE_ME * * ARM is a special case: ARM

double values are in mixed/cross endian * format while ARM duk_uint64_t values are in standard little endian * format (H G F E D C B A). When a double is read as a duk_uint64_t * from memory, the register will contain the (logical) value * E F G H A B C D. This requires some special handling below. ** Indexes of various types (8-bit, 16-bit, 32-bit) in memory relative to * the logical (big endian) order: ** byte order duk_uint8_t duk_uint16_t duk_uint32_t * BE 01234567 0123 01 * LE 76543210 3210 10 * ME (ARM) 32107654 1032 01 ** Some processors may alter NaN values in a floating point load+store. * For instance, on X86 a FLD + FSTP may convert a signaling NaN to a * quiet one. This is catastrophic when NaN space is used in packed * duk_tval values. See: misc/clang_aliasing.c.

11103. * Helper macros for reading/writing memory representation parts, used * by duk_numconv.c and duk_tval.h.

11104. string conversion: use 'T' instead of '' as a separator

11105. Ecmascript number: double

11106. TypeError

11107. Support array for ROM pointer compression. Only declared when ROM * pointer compression is active.

11108. **comment:** Although extra/top could be an unsigned type here, using a signed type * makes the API more robust to calling code calculation errors or corner * cases (where caller might occasionally come up with negative values). * Negative values are treated as zero, which is better than casting them * to a large unsigned number. (This principle is used elsewhere in the * API too.)

label: code-design

11109. * Misc conversion

11110. Ecmascript object: includes objects, arrays, functions, threads

11111. URIError

11112. all doubles are considered normalized

11113. AUTHORS.rst

11114. day within month: 0 to 30

11115. setter: call is a time setter (affects hour, min, sec, ms); otherwise date setter (affects year, month, day-in-month)

11116. (internal) no filename on stack

11117. compile eval code (instead of global code)

11118. One problem with this macro is that expressions like the following fail * to compile: "(void) duk_error(...)". But because duk_error() is noreturn, * they make little sense anyway.

11119. set setter (given on value stack)

11120. prefer number, unless input is a Date, in which * case prefer string (E5 Section 8.12.8)

11121. Part indices for internal breakdowns. Part order from DUK_DATE_IDX_YEAR * to DUK_DATE_IDX_MILLISECOND matches argument ordering of Ecmascript API * calls (like Date constructor call). Some functions in duk.bi_date.c * depend on the specific ordering, so change with care. 16 bits are not * enough for all parts (year, specifically). * * (Must be in-sync with genbuiltins.py.)

11122. E == 0x7ff, F == 8 => normalized NaN

11123. Enumeration flags for duk_enum()

11124. year

11125. * Other state related functions

11126. DUK_USE_PROVIDE_DEFAULT_ALLOC_FUNCTIONS

11127. * Default allocation functions. * * Assumes behavior such as malloc allowing zero size, yielding * a NULL or a unique pointer which is a no-op for free.

11128. * Buffer

11129. maximum size check is handled by callee

11130. **comment:** Forget the previous allocation, setting size to 0 and alloc to * NULL. Caller is responsible for freeing the previous allocation. * Getting the allocation and clearing it is done in the same API * call to avoid any chance of a realloc.

label: code-design

11131. Load constants onto value stack but don't yet copy to buffer.

11132. -> [func funcname env]

11133. _Formals

11134. Setup function properties.

11135. Push function object, init flags etc. This must match * duk_js_push_closure() quite carefully.

11136. **comment:** * Bytecode dump/load * * The bytecode load primitive is more important performance-wise than the * dump primitive. * * Unlike most Duktape API calls, bytecode dump/load is not guaranteed to be * memory safe for invalid arguments - caller beware! There's little point * in trying to achieve memory safety unless bytecode instructions are also * validated which is not easy to do with indirect register references etc.

label: code-design

11137. _Varmap is dense

11138. We know _Formals is dense and all entries will be in the * array part. GC and finalizers shouldn't affect _Formals * so side effects should be fine.

11139. skip line info

11140. Bound functions don't have all properties so we'd either need to * lookup the non-bound target function or reject bound functions. * For now, bound functions are rejected.

11141. end of _Formals

11142. **comment:** XXX: There's some overlap with duk_js_closure() here, but * seems difficult to share code. Ensure that the final function * looks the same as created by duk_js_closure().

label: code-design

11143. The caller is responsible for being sure that bytecode being loaded * is valid and trusted. Invalid bytecode can cause memory unsafe * behavior directly during loading or later during bytecode execution * (instruction validation would be quite complex to implement). * * This signature check is the only sanity check for detecting * accidental invalid inputs. The initial 0xFF byte ensures no * ordinary string will be accepted by accident.

11144. buffer limits

11145. Inner functions recursively.

11146. known to be number; in fact an integer

11147. Object extra properties. * * There are some difference between function templates and functions. * For example, function templates don't have .length and nargs is * normally used to instantiate the functions.

11148. Bytecode instructions: endian conversion needed unless * platform is big endian.

11149. **comment:** Load a function from bytecode. The function object returned here must * match what is created by duk_js_push_closure() with respect to its flags, * properties, etc. * * NOTE: there are intentionally no input buffer length / bound checks. * Adding them would be easy but wouldn't ensure memory safety as untrusted * or broken bytecode is unsafe during execution unless the opcodes themselves * are validated (which is quite complex, especially for indirect opcodes).

label: code-design

11150. duk_hcompiledfunction flags; quite version specific

11151. standard prototype

11152. Load bytecode instructions.

11153. string limits

11154. no side effects

11155. We know _Varmap only has own properties so walk property * table directly. We also know _Varmap is dense and all * values are numbers; assert for these. GC and finalizers * shouldn't affect _Varmap so side effects should be fine.

11156. **comment:** known to be 32-bit

label: code-design

11157. Fixed header info.

11158. constants are strings or numbers now

11159. Create function 'data' buffer but don't attach it yet.

11160. With constants and inner functions on value stack, we can now * atomically finish the function 'data' buffer, bump refcounts, * etc. * * Here we take advantage of the value stack being just a duk_tval * array: we can just memcpy() the constants as long as we incref * them afterwards.

11161. Estimating the result size beforehand would be costly, so * start with a reasonable size and extend as needed.

11162. DUK_USE_BYTECODE_DUMP_SUPPORT
11163. ensures no overflow
11164. The function object is now reachable and refcounts are fine, * so we can pop off all the temporaries.
11165. _Varmap
11166. -> [func funcname env funcname func]
11167. XXX: This causes recursion up to inner function depth * which is normally not an issue, e.g. mark-and-sweep uses * a recursion limiter to avoid C stack issues.
 Avoiding * this would mean some sort of a work list or just refusing * to serialize deep functions.
11168. Constants: variable size encoding.
11169. Return with final function pushed on stack top.
11170. [... func buf] -> [... buf]
11171. [... buf func] -> [... func]
11172. Original function instance/template had NAMEBINDING. * Must create a lexical environment on loading to allow * recursive functions like 'function foo() { foo(); }'.
11173. **comment:** * Dump/load helpers, xxx_raw() helpers do no buffer checks
 label: code-design
11174. may be NULL if no constants or inner funcs
11175. assert just a few critical flags
11176. Load inner functions to value stack, but don't yet copy to buffer.
11177. **comment:** XXX: awkward
 label: code-design
11178. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
11179. **comment:** Array is dense and contains only strings, but ASIZE may * be larger than used part and there are UNUSED entries.
 label: code-design
11180. Important to do a fastint check so that constants are * properly read back as fastints.
11181. Value stack is used to ensure reachability of constants and * inner functions being loaded. Require enough space to handle * large functions correctly.
11182. Explicit zero size check to avoid NULL 'tv1'.
11183. func.prototype.constructor = func
11184. end of _Varmap
11185. -> [func funcname env funcname]
11186. -> [... cons target]
11187. noblame_fileline
11188. [... func this arg1 ... argN]
11189. thread
11190. * Figure out the final, non-bound constructor, to get "prototype" * property.
11191. **comment:** * Calls. * * Protected variants should avoid ever throwing an error.
 label: code-design
11192. num_stack_res
11193. class Object, extensible
11194. note that we can't reliably pop anything here
11195. [... retval]
11196. Strict by default.
11197. **comment:** XXX: merge this with duk_js_call.c, as this function implements * core semantics (or perhaps merge the two files altogether).
 label: code-design
11198. num_stack_args
11199. nrets
11200. Anything else is not constructable.
11201. Prepare value stack for a method call through an object property. * May currently throw an error e.g. when getting the property.
11202. not protected, respect reclimit, is a constructor call
11203. respect reclimit, not constructor
11204. * Augment created errors upon creation (not when they are thrown or * rethrown). __FILE__ and __LINE__ are not desirable here; the call * stack reflects the caller which is correct.
11205. **comment:** XXX: code duplication
 label: code-design
11206. not protected, respect reclimit, not constructor
11207. pop_final_cons
11208. [... fallback retval]
11209. -> [... target]
11210. For user code this could just return 1 (strict) always * because all Duktape/C functions are considered strict, * and strict is also the default when nothing is running.
 * However, Duktape may call this function internally when * the current activation is an Ecmascript function, so * this cannot be replaced by a 'return 1' without fixing * the internal call sites.
11211. make absolute
11212. use fallback as 'this' value
11213. [... fallback constructor fallback(this) arg1 ... argN]; * Note: idx_cons points to first 'fallback', not 'constructor'.
11214. **comment:** * XXX: if duk_handle_call() took values through indices, this could be * made much more sensible. However, duk_handle_call() needs to fudge * the 'this' and 'func' values to handle bound function chains, which * is now done "in-place", so this is not a trivial change.
 label: code-design
11215. * Determine whether to use the constructor return value as the created * object instance or not.
11216. Lightfuncs cannot be bound.
11217. * Call the constructor function (called in "constructor mode").
11218. [... key arg1 ... argN]
11219. Inputs: explicit arguments (nargs), +1 for key, +2 for obj_index/nargs passing. * If the value stack does not contain enough args, an error is thrown; this matches * behavior of the other protected call API functions.
11220. **comment:** * There are two [[Construct]] operations in the specification: * * - E5 Section 13.2.2: for Function objects * - E5 Section 15.3.4.5.2: for "bound" Function objects * * The chain of bound functions is resolved in Section 15.3.4.5.2, * with arguments "piling up" until the [[Construct]] internal * method is called on the final, actual Function object. Note * that the "prototype" property is looked up *only* from the * final object, *before* calling the constructor. * * Currently we follow the bound function chain here to get the * "prototype" property value from the final, non-bound function. * However, we let duk_handle_call() handle the argument "piling" * when the constructor is called. The bound function chain is * thus now processed twice. * * When constructing new Array instances, an unnecessary object is * created and discarded now: the standard [[Construct]] creates an * object, and calls the Array constructor. The Array constructor * returns an Array instance, which is used as the result value for * the "new" operation; the object created before the Array constructor * call is discarded. * * This would be easy to fix, e.g. by knowing that the Array constructor * will always create a replacement object and skip creating the fallback * object in that case. * * Note: functions called via "new" need to know they are called as a * constructor. For instance, built-in constructors behave differently * depending on how they are called.
 label: code-design
11221. [... constructor arg1 ... argN final_cons]
11222. See comments in duk_pcall().
11223. because callstack_top > 0
11224. Checking callability of the immediate target * is important, same for constructability. * Checking it for functions down the bound * function chain is not strictly necessary * because .bind() should normally reject them. * But it's good to check anyway because it's * technically possible to edit the bound function * chain via

internal keys.

11225. * Create "fallback" object to be used as the object instance, * unless the constructor returns a replacement value. * Its internal prototype needs to be set based on "prototype" * property of the constructor.

11226. call_flags

11227. Get the original arguments. Note that obj_index may be a relative * index so the stack must have the same top when we use it.

11228. must work for nargs <= 0

11229. * Duktape/C function magic

11230. fall through

11231. unreachable

11232. awkward; we assume there is space for this

11233. [... key arg1 ... argN func]

11234. func

11235. Note: -nargs alone would fail for nargs == 0, this is OK

11236. [... constructor arg1 ... argN]

11237. also stash it before constructor, * in case we need it (as the fallback value)

11238. nargs

11239. **comment:** For now, just use duk_safe_call() to wrap duk_new(). We can't * simply use a protected duk_handle_call() because there's post * processing which might throw. It should be possible to ensure * the post processing never throws (except in internal errors and * out of memory etc which are always allowed) and then remove this * wrapper.

label: code-design

11240. [... func arg1 ... argN]

11241. [... constructor arg1 ... argN final_cons fallback]

11242. duplicate key

11243. We can't reliably pop anything here because the stack input * shape is incorrect. So we throw an error; if the caller has * no catch point for this, a fatal error will occur. Another * alternative would be to just return an error. But then the * stack would be in an unknown state which might cause some * very hard to diagnose problems later on. Also note that even * if we did not throw an error here, the underlying call handler * might STILL throw an out-of-memory error or some other internal * fatal error.

11244. * Manipulate callstack for the call.

11245. **comment:** XXX: awkward; we assume there is space for this, overwrite * directly instead?

label: code-design

11246. * Must be careful to catch errors related to value stack manipulation * and property lookup, not just the call itself.

11247. -----XXX-----

11248. output 3 bytes from 't'

11249. XX==

11250. Innermost fast path processes 4 valid base-64 characters at a time * but bails out on whitespace, padding chars ('=') and invalid chars. * Once the slow path segment has been processed, we return to the * inner fast path again. This handles e.g. base64 with newlines * reasonably well because the majority of a line is in the fast path.

11251. Fixed buffer, no zeroing because we'll fill all the data.

11252. allowed ascii whitespace

11253. full 3-byte -> 4-char conversions

11254. allow basic ASCII whitespace

11255. Here we'd have the option of decoding unpadded base64 * (e.g. "xxxxyy" instead of "xxxxyy==". Currently not * accepted.

11256. Computation must not wrap, only srclen + 3 is at risk of * wrapping because after that the number gets smaller. * This limit works for 32-bit size_t: * 0x100000000
- 3 - 1 = 4294967292

11257. XXX=

11258. read 3 bytes into 't', padded by zero

11259. back to fast loop

11260. never here

11261. **comment:** * Missing bytes snip base64 example * 0 4 XXXX * 1 3 XXX= * 2 2 XX==

label: code-design

11262. Backtrack.

11263. idx_value

11264. idx_space

11265. For invalid characters the value -1 gets extended to * at least 16 bits. If either nybble is invalid, the * resulting 't' will be < 0.

11266. -----XX XXXX-----

11267. **comment:** XXX: optimize for buffer inputs: no need to coerce to a string * which causes an unnecessary interning.

label: code-design

11268. DUK_USE_HEX_FASTPATH

11269. pointer is aligned, guaranteed for fixed buffer

11270. Don't allow actual chars after equal sign.

11271. Continue parsing after padding, allows concatenated, * padded base64.

11272. case 0: nop

11273. XXXXXX--

11274. **comment:** Shared handling for encode/decode argument. Fast path handling for * buffer and string values because they're the most common. In particular, * avoid creating a temporary string or buffer when possible.

label: code-design

11275. Handle one slow path unit (or finish if we're done).

11276. idx_reviver

11277. Check if any lookup above had a negative result.

11278. * Encoding and decoding basic formats: hex, base64. * * These are in-place operations which may allow an optimized implementation. * * Base-64:
<https://tools.ietf.org/html/rfc4648#section-4>

11279. **comment:** if 'src < src_end_safe', safe to read 4 bytes

label: requirement

11280. two level break

11281. **comment:** XXX: optimize for string inputs: no need to coerce to a buffer * which makes a copy of the input.

label: code-design

11282. Don't allow mixed padding and actual chars.

11283. -----XX

11284. The lookup byte is intentionally sign extended to (at least) * 32 bits and then ORed. This ensures that is at least 1 byte * is negative, the highest bit of 't' will be set at the end * and we don't need to check every byte.

11285. invalid padding

11286. We don't check the zero padding bytes here right now * (that they're actually zero). This seems to be common * behavior for base-64 decoders.

11287. aaaaabb bbbbcccc cccccccc

11288. **comment:** Here we can choose either to end parsing and ignore * whatever follows, or to continue parsing in case * multiple (possibly padded) base64 strings have been * concatenated. Currently, keep on parsing.

label: code-design

11289. **comment:** XXX: Using a string return value forces a string intern which is * not always necessary. As a rough performance measure, hex encode * time for tests/perf/test-hex-encode.js dropped from ~35s to ~15s * without string coercion. Change to returning a buffer and let the * caller coerce to string if necessary?

label: code-design

11290. upper limit, assuming no whitespace etc

11291. checked by caller

11292. shift in zeroes

11293. idx_replacer

11294. Fast path, handle units with just actual encoding characters.

11295. **comment:** XXX: convert to fixed buffer?

label: code-design

11296. Computation must not wrap; this limit works for 32-bit size_t: * >>> srclen = 3221225469 * >>> "%x' % ((srclen + 2) / 3 * 4) * 'fffffff'c'

11297. **comment:** There may be whitespace between the equal signs.

label: code-design

11298. XXXXXX-----

11299. **comment:** Tested: not faster on x64

label: code-design

11300. **comment:** A straightforward 64-byte lookup would be faster * and cleaner, but this is shorter.

label: code-design

11301. Emit 3 bytes and backtrack if there was padding. There's * always space for the whole 3 bytes so no check needed.

11302. DUK_USE_BASE64_FASTPATH

11303. flags

11304. Note: for dstlen=0, dst may be NULL

11305. Note: for srclen=0, src may be NULL

11306. * Compilation and evaluation

11307. Arguments can be: [source? filename? &comp_args] so that * nargs is 1 to 3. Call site encodes the correct nargs count * directly into flags.

11308. Helper which can be called both directly and with duk_safe_call().

11309. [... source? func_template]

11310. [... source? filename?]

11311. add_auto_proto

11312. [... closure/error]

11313. **comment:** String length is computed here to avoid multiple evaluation * of a macro argument in the calling side.

label: code-design

11314. [... result/error]

11315. **comment:** Note: strictness is *not* inherited from the current Duktape/C. * This would be confusing because the current strictness state * depends on whether we're running inside a Duktape/C activation * (= strict mode) or outside of any activation (= non-strict mode). * See tests/api/test-eval-strictness.c for more discussion.

label: code-design

11316. [... source? filename?] (depends on flags)

11317. [... func_template]

11318. **comment:** XXX: when this error is caused by a nonexistent * file given to duk_peval_file() or similar, the * error message is not the best possible.

label: code-design

11319. e.g. duk_push_string_file_raw() pushed undefined

11320. should be first on 64-bit platforms

11321. Automatic filename: 'eval' or 'input'.

11322. [... source? filename? &comp_args] (depends on flags)

11323. explicit 'this' binding, see GH-164

11324. [... closure]

11325. **comment:** XXX: unnecessary translation of flags

label: code-design

11326. Note: strictness is not inherited from the current Duktape/C * context. Otherwise it would not be possible to compile * non-strict code inside a Duktape/C activation (which is * always strict now). See tests/api/test-eval-strictness.c * for discussion.

11327. Eval is just a wrapper now.

11328. args incorrect

11329. may be safe, or non-safe depending on flags

11330. -> [... closure]

11331. [... source? filename]

11332. Treat like debugger statement: nop

11333. [... arr jsonx(arr) res] -> [... res jsonx(arr)]

11334. **comment:** XXX: conversion errors should not propagate outwards. * Perhaps values need to be coerced individually?

label: code-design

11335. idx_value

11336. idx_space

11337. **comment:** XXX: should there be an error or an automatic detach if * already attached?

label: code-design

11338. Other callbacks are optional.

11339. nop

11340. Pause on the next opcode executed. This is always safe to do even * inside the debugger message loop: the interrupt counter will be reset * to its proper value when the message loop exits.

11341. DUK_USE_DEBUGGER_SUPPORT

11342. Start in paused state.

11343. no_block

11344. No debugger support, just pop values.

11345. idx_replacer

11346. unreachable

11347. Return non-zero (true) if we have a good reason to believe * the notify was delivered; if we're still attached at least * a transport error was not indicated by the transport write * callback. This is not a 100% guarantee of course.

11348. Calling duk_debugger_cooperate() while Duktape is being * called into is not supported. This is not a 100% check * but prevents any damage in most cases.

11349. Treat like a debugger statement: ignore when not attached.

11350. We don't duk_require_stack() here now, but rely on the caller having * enough space.

11351. * Debugging related API calls

11352. flags

11353. Send version identification and flush right afterwards. Note that * we must write raw, unframed bytes here.

11354. Can be called multiple times with no harm.

11355. [... new_glob new_env]

11356. side effects, in theory (referenced by global env)

11357. Assume that either all memory funcs are NULL or non-NULL, mixed * cases will now be unsafe.

11358. * Heap creation and destruction

11359. [... new_glob new_env new_glob new_glob]

11360. XXX: just assert non-NULL values here and make caller arguments * do the defaulting to the default implementations (smaller code)?

11361. side effects

11362. * Replace lexical environment for global scope ** Create a new object environment for the global lexical scope. * We can't just reset the _Target property of the current one, * because the lexical scope is shared by other threads with the * same (initial) built-ins.
11363. [...]
11364. * Replace global object.
11365. **comment:** XXX: better place for this
 label: code-design
11366. no prototype, updated below
11367. without refcounts
11368. **comment:** These are macros for now, but could be separate functions to reduce code * footprint (check call site count before refactoring).
 label: code-design
11369. sometimes stack and array indices need to go on the stack
11370. [key val] -> []
11371. Current convention is to use duk_size_t for value stack sizes and global indices, * and duk_idx_t for local frame indices.
11372. [val] -> []
11373. **comment:** XXX: assumed to fit for now
 label: code-design
11374. duk_push_this() + CheckObjectCoercible() + duk_to_object()
11375. only needed by debugger for now
11376. * Internal API calls which have (stack and other) semantics similar * to the public API.
11377. Raw internal valstack access macros: access is unsafe so call site * must have a guarantee that the index is valid. When that is the case, * using these macro results in faster and smaller code than duk_get_tval(). * Both 'ctx' and 'idx' are evaluated multiple times, but only for asserts.
11378. Flag ORed to err_code to indicate __FILE__ / __LINE__ is not * blamed as source of error for error fileName / lineNumber.
11379. [] -> []
11380. duk_push_sprintf constants
11381. **comment:** XXX: add fastint support?
 label: requirement
11382. This would be pointless: unexpected type and lightfunc would both return NULL
11383. duk_push_(u)int() is guaranteed to support at least (un)signed 32-bit range
11384. Get a borrowed duk_tval pointer to the current 'this' binding. Caller must * make sure there's an active callstack entry. Note that the returned pointer * is unstable with regards to side effects.
11385. DUK_API_INTERNAL_H_INCLUDED
11386. Set object 'length'.
11387. duk_push_this() + CheckObjectCoercible() + duk_to_string()
11388. Push the current 'this' binding; throw TypeError if binding is not object * coercible (CheckObjectCoercible).
11389. [] -> [val]
11390. Valstack resize flags
11391. out_clamped=NULL, RangeError if outside range
11392. **comment:** unused
 label: code-design
11393. stridx_logfunc[] must be static to allow initializer with old compilers like BCC
11394. [...] Logger clog logfunc clog(=this) msg]
11395. nargs
11396. [...] Logger clog logfunc clog]
11397. **comment:** * Logging ** Current logging primitive is a sprintf-style log which is convenient * for most C code. Another useful primitive would be to log N arguments * from value stack (like the Ecmascript binding does).
 label: code-design
11398. [...] Logger clog res]
11399. * Memory calls.
11400. NULL accepted
11401. **comment:** * Note: since this is an exposed API call, there should be * no way a mark-and-sweep could have a side effect on the * memory allocation behind 'ptr'; the pointer should never * be something that Duktape wants to change. ** Thus, no need to use DUK_REALLOC_INDIRECT (and we don't * have the storage location here anyway).
 label: code-design
11402. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property get right now.
11403. * Helpers for writing multiple properties
11404. Key and value indices are either (-2, -1) or (-1, -2). Given idx_key, * idx_val is always (idx_key ^ 0x01).
11405. remove key and value
11406. [...] obj ...]
11407. This is a rare property helper; it sets the global thrower (E5 Section 13.2.3) * setter/getter into an object property. This is needed by the 'arguments' * object creation code, function instance creation code, and Function.prototype.bind().
11408. * Object finalizer
11409. proto can also be NULL here (allowed explicitly)
11410. a value is left on stack regardless of rc
11411. pop key
11412. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property put right now (putprop protects * against it internally).
11413. "Have" flags must not be conflicting so that they would * apply to both a plain property and an accessor at the same * time.
11414. Object.defineProperty() equivalent C binding.
11415. Define own property without inheritance looks and such. This differs from * [[DefineOwnProperty]] because special behaviors (like Array 'length') are * not invoked by this method. The caller must be careful to invoke any such * behaviors if necessary.
11416. **comment:** XXX: the duk_hobject_enum.c stack APIs should be reworked
 label: code-design
11417. defprop_flags
11418. * Property handling ** The API exposes only the most common property handling functions. * The caller can invoke Ecmascript built-ins for full control (e.g. * defineProperty, getOwnPropertyDescriptor).
11419. * Object related * * Note: seal() and freeze() are accessible through Ecmascript bindings, * and are not exposed through the API.
11420. XXX: shared helper for duk_push_hobject_or_undefined()
11421. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property existence check right now.
11422. * Object handling: property access and other support functions.
11423. remove key
11424. **comment:** Careful here and with other duk_put_prop_xxx() helpers: the * target object and the property value may be in the same value * stack slot (unusual, but still conceptually clear).
 label: code-design
11425. value popped by call
11426. 1 if property found, 0 otherwise
11427. **comment:** XXX: "read only object"?
 label: code-design
11428. [...] global val] -> [...] global]
11429. Note: this may fail, caller should protect the call if necessary

11430. **comment:** XXX: direct implementation
label: requirement

11431. **comment:** Clean up stack
label: code-design

11432. [target] -> [enum]

11433. * Object prototype

11434. **comment:** XXX: these could be implemented as macros calling an internal function * directly. * XXX: same issue as with Duktape.fin: there's no way to delete the property * now (just set it to undefined).
label: code-design

11435. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property delete right now.

11436. * Shortcut for accessing global object properties

11437. **comment:** XXX: shared api error strings, and perhaps even throw code for rare cases?
label: code-design

11438. Maximum value check ensures ' nbytes' won't wrap below.

11439. DUK_BUFOBJ_UINT32ARRAY

11440. throw_flag

11441. only flag now

11442. DUK_USE_VARIADIC_MACROS

11443. E5 Section 9.1

11444. The underlying types for offset/length in duk_hbufferobject is * duk_uint_t; make sure argument values fit and that offset + length * does not wrap.

11445. * When resizing the valstack, a mark-and-sweep may be triggered for * the allocation of the new valstack. If the mark-and-sweep needs * to use our thread for something, it may cause *the same valstack* * to be resized recursively. This happens e.g. when mark-and-sweep * finalizers are called. This is taken into account carefully in * duk_resize_valstack(). * * 'new_size' is known to be <= valstack_max, which ensures that * size_t and pointer arithmetic won't wrap in duk_resize_valstack().

11446. **comment:** Function pointers do not always cast correctly to void * * (depends on memory and segmentation model for instance), * so they coerce to NULL.
label: code-design

11447. This would be pointless: we'd return NULL for both lightfuncs and * unexpected types.

11448. Lightfunc coerces to a Function instance with concrete * properties. Since 'length' is virtual for Duktape/C * functions, don't need to define that. * * The result is made extensible to mimic what happens to * strings: * > Object.isExtensible(Object('foo')) * true

11449. consequence of above

11450. Never executed if new size is smaller.

11451. * An array must have a 'length' property (E5 Section 15.4.5.2). * The special array behavior flag must only be enabled once the * length property has been added. * * The internal property must be a number (and preferably a * fastint if fastint support is enabled).

11452. **comment:** * Global state for working around missing variadic macros
label: code-design

11453. use a temp: decref only when valstack reachable values are correct

11454. Note: recursive call

11455. * Stack index validation/normalization and getting a stack duk_tval ptr. * * These are called by many API entrypoints so the implementations must be * fast and "inlined". * * There's some repetition because of this; keep the functions in sync.

11456. **comment:** XXX: There's quite a bit of overlap with buffer creation handling in * duk.bi.buffer.c. Look for overlap and refactor.
label: code-design

11457. Should match Function.prototype.toString()

11458. **comment:** XXX: combine all the integer conversions: they share everything * but the helper function for coercion.
label: code-design

11459. fatal_func should be noreturn, but noreturn declarations on function * pointers has a very spotty support apparently so it's not currently * done.

11460. **comment:** XXX: .code = err_code disabled, not sure if useful
label: code-design

11461. * Special cases like NaN and +/- Infinity are handled explicitly * because a plain C coercion from double to int handles these cases * in undesirable ways. For instance, NaN may coerce to INT_MIN * (not zero), and INT_MAX + 1 may coerce to INT_MIN (not INT_MAX). * * This double-to-int coercion differs from ToInteger() because it * has a finite range (ToInteger() allows e.g. +/- Infinity). It * also differs fromToInt32() because the INT_MIN/INT_MAX clamping * depends on the size of the int type on the platform. In particular, * on platforms with a 64-bit int type, the full range is allowed.

11462. Assert for value stack initialization policy.

11463. val is unsigned so >= 0

11464. important to do this *after* pushing, to make the thread reachable for gc

11465. !DUK_USE_PARANOI_ERRORS

11466. initial estimate based on format string

11467. no need to incref

11468. coerce towards zero

11469. * API calls related to general value stack manipulation: resizing the value * stack, pushing and popping values, type checking and reading values, * coercing values, etc. * * Also contains internal functions (such as duk_get_tval()), defined * in duk_api_internal.h, with semantics similar to the public API.

11470. Non-critical.

11471. default: false

11472. success, fixup pointers

11473. Coerce top into Object.prototype.toString() output.

11474. * Misc helpers

11475. nbytes zero size case * <-----> * [... | p | x | x | q] [... | p==q] * => [... | x | x | q] [...]

11476. **comment:** XXX: several pointer comparison issues here
label: code-design

11477. must be computed after realloc

11478. side effects

11479. number

11480. radix

11481. Popping one element is called so often that when footprint is not an issue, * compile a specialized function for it.

11482. Here we rely on duk_hstring instances always being zero * terminated even if the actual string is not.

11483. get pointer offsets for tweaking below

11484. zero size not an issue: pointers are valid

11485. DUK_BUFOBJ_INT8ARRAY

11486. advance manually

11487. out_clamped==NULL -> RangeError if outside range

11488. not reached

11489. because of valstack init policy

11490. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Returns NULL for a lightfunc.

11491. ensure value is 1 or 0 (not other non-zero)

11492. DUK_BUFOBJ_INT16ARRAY

11493. No net refcount changes.

11494. **comment:** SCANBUILD: with suitable dmin/dmax limits 'd' is unused
label: code-design

11495. object is now reachable

11496. 4 low bits
11497. * Pushers
11498. quotes
11499. XXX: perhaps refactor this to allow caller to specify some parameters, or * at least a 'compact' flag which skips any spare or round-up .. useful for * emergency gc.
11500. **comment:** XXX: string not shared because it is conditional
 label: code-design
11501. E5 Section 8.12.8
11502. 'd' and 'res' agree here
11503. XXX: direct valstack write
11504. Creation time error augmentation
11505. filename may be NULL in which case file/line is not recorded
11506. Check for maximum buffer length.
11507. Inref copies, keep originals.
11508. DUK_HOBJECT_FLAG_EXOTIC_DUKFUNC: omitted here intentionally
11509. 'undefined' already on stack top
11510. unsigned
11511. covers +Infinity
11512. **comment:** A plain buffer coerces to a Duktape.Buffer because it's the * object counterpart of the plain buffer value. But it might * still make more sense to produce an ArrayBuffer here?
 label: code-design
11513. avoid warning (unsigned)
11514. default prototype (Note: 'obj' must be reachable)
11515. everything except object stay as is
11516. flags
11517. Enable DUKFUNC exotic behavior once properties are set up.
11518. DUK_BUFOBJ_UINT8ARRAY
11519. may be NULL
11520. **comment:** No coercions or other side effects, so safe
 label: requirement
11521. Allocate a new valstack. * * Note: cannot use a plain DUK_REALLOC() because a mark-and-sweep may * invalidate the original thr->valstack base pointer inside the realloc * process. See doc/memory-management.rst.
11522. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object).
 Return value is never NULL.
11523. DUK_BUFOBJ_DATAVIEW
11524. If tv1==tv2 this is a NOP, no check is needed
11525. * Basic stack manipulation: swap, dup, insert, replace, etc
11526. No net refcount change.
11527. **comment:** XXX: shorter version for 12-byte representation?
 label: code-design
11528. special handling of fmt==NULL
11529. specific assert for wrapping
11530. Clamping only necessary for 32-bit ints.
11531. **comment:** * Function pointers * * Printing function pointers is non-portable, so we do that by hex printing * bytes from memory.
 label: code-design
11532. DUK_BUFOBJ_UINT8CLAMPEDARRAY
11533. caller required to know
11534. ANSI C typing
11535. as is
11536. Only allow Duktape.Buffer when support disabled.
11537. **comment:** XXX: Simplify this algorithm, should be possible to come up with * a shorter and faster algorithm by inspecting IEEE representation * directly.
 label: code-design
11538. String sanitizer which escapes ASCII control characters and a few other * ASCII characters, passes Unicode as is, and replaces invalid UTF-8 with * question marks. No errors are thrown for any input string, except in out * of memory situations.
11539. Explicit length is only needed if it differs from 'nargs'.
11540. DUK_BUFOBJ_FLOAT64ARRAY
11541. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object).
 Return value is NULL if value is neither an object nor a * lightfunc.
11542. * Must be very careful here, every DECREF may cause reallocation * of our valstack.
11543. inline initializer for coercers[] is not allowed by old compilers like BCC
11544. **comment:** XXX: duk_ssize_t would be useful here
 label: code-design
11545. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
11546. **comment:** XXX: better macro for DUK_TVAL_IS_UNDEFINED_OR_NULL(tv)
 label: code-design
11547. no shrink
11548. nop: insert top to top
11549. **comment:** Lightfunc name, includes Duktape/C native function pointer, which * can often be used to locate the function from a symbol table. * The name also includes the 16-bit duk_tval flags field because it * includes the magic value. Because a single native function often * provides different functionality depending on the magic value, it * seems reasonably to include it in the name. * * On the other hand, a complicated name increases string table * pressure in low memory environments (but only when function name * is accessed).
 label: code-design
11550. periods
11551. Note: here we must be wary of the fact that a pointer may be * valid and be a NULL.
11552. Nothing to initialize, strsz[] is in ROM.
11553. [... parent stash stash] -> [... parent stash]
11554. if slice not fully valid, treat as error
11555. Note: no need to re-lookup tv, conversion is side effect free
11556. Template functions are not strictly constructable (they don't * have a "prototype" property for instance), so leave the * DUK_HOBJECT_FLAG_CONSTRUCTABLE flag cleared here.
11557. DUK_BUFOBJ_ARRAYBUFFER
11558. check stack first
11559. [... retval]; popped below
11560. tv points to element just below prev top
11561. **comment:** XXX: fastint?
 label: code-design
11562. dynamic
11563. DUK_USE_FILE_IO
11564. Return invalid index; if caller uses this without checking * in another API call, the index won't map to a valid stack * entry.

11565. ignore fclose() error
11566. DUK_BUFOBJ_FLOAT32ARRAY
11567. not reachable
11568. coerced value is updated to value stack even when RangeError thrown
11569. error gets its 'name' from the prototype
11570. no side effects
11571. shrink case; leave some spare
11572. 'this' binding exists
11573. may be NULL (but only if size is 0)
11574. failed, resize and try again
11575. * Poppers
11576. -> [... func this]
11577. **comment:** not very useful, used for debugging
 label: code-design
11578. * Lightfunc
11579. Because new_size != 0, if condition doesn't need to be * (new_valstack != NULL || new_size == 0).
11580. Not halfway, round to nearest.
11581. * Number is special because it doesn't have a specific * tag in the 8-byte representation.
11582. **comment:** XXX: fast primitive to set a bunch of values to UNDEFINED
 label: code-design
11583. TypedArray views need an automatic ArrayBuffer which must be * provided as .buffer property of the view. Just create a new * ArrayBuffer sharing the same underlying buffer.
11584. make the new thread reachable
11585. **comment:** copied so that 'ap' can be reused
 label: code-design
11586. estimate is valid
11587. * Comparison
11588. DUK_USE_REFERENCE_COUNTING
11589. **comment:** XXX: repetition of stack pre-checks -> helper or macro or inline
 label: code-design
11590. Value coercion (in stack): ToInteger(), E5 Section 9.4 * API return value coercion: custom
11591. [... func]
11592. nrets
11593. Error code also packs a tracedata related flag.
11594. For tv1 == tv2, both pointing to stack top, the end result * is same as duk_pop(ctx).
11595. allow fmt==NULL
11596. thr->heap->ljjmpbuf_ptr is checked by duk_err_longjmp() so we don't * need to check that here. If the value is NULL, a panic occurs because * we can't return.
11597. DUK_BUFOBJ_NODEJS_BUFFER
11598. Stack size increases or stays the same.
11599. Exact halfway, round to even.
11600. **comment:** If no explicit message given, put error code into message field * (as a number). This is not fully in keeping with the EcmaScript * error model because messages are supposed to be strings (Error * constructors use ToString() on their argument). However, it's * probably more useful than having a separate 'code' property.
 label: code-design
11601. Note: may be triggered even if minimal new_size would not reach the limit, * plan limit accordingly (taking DUK_VALSTACK_GROW_STEP into account).
11602. We intentionally ignore the duk_safe_call() return value and only * check the output type. This way we don't also need to check that * the returned value is indeed a string in the success case.
11603. Cannot use duk_to_string() on the buffer because it is usually * larger than 'len'. Also, 'buf' is usually a stack buffer.
11604. Clamping needed if duk_int_t is 64 bits.
11605. Note: src_data may be NULL if input is a zero-size * dynamic buffer.
11606. !DUK_USE_PREFER_SIZE
11607. Error: try coercing error to string once.
11608. Note: the realloc may have triggered a mark-and-sweep which may * have resized our valstack internally. However, the mark-and-sweep * MUST NOT leave the stack bottom/top in a different state. Particular * assumptions and facts: * * - The thr->valstack pointer may be different after realloc, * and the offset between thr->valstack_end <-> thr->valstack * may have changed. * - The offset between thr->valstack_bottom <-> thr->valstack * and thr->valstack_top <-> thr->valstack MUST NOT have changed, * because mark-and-sweep must adhere to a strict stack policy. * In other words, logical bottom and top MUST NOT have changed. * - All values above the top are unreachable but are initialized * to UNDEFINED, up to the post-realloc valstack_end. * - 'old_end_offset' must be computed after realloc to be correct.
11609. **comment:** XXX: inlined DECREF macro would be nice here: no NULL check, * refzero queueing but no refzero algorithm run (= no pointer * instability), inline code.
 label: code-design
11610. **comment:** XXX: other variants like uint, u32 etc
 label: code-design
11611. rely on interning, must be this string
11612. DUK_USE_FASTINT
11613. Because value stack init policy is 'undefined above top', * we don't need to write, just assert.
11614. sanity limit for function pointer size
11615. Initial stack size satisfies the stack spare constraints so there * is no need to require stack here.
11616. * Error throwing
11617. can't resize below 'top'
11618. -> [... retval]
11619. Non-buffer value is first ToString() coerced, then converted * to a buffer (fixed buffer is used unless a dynamic buffer is * explicitly requested).
11620. shared flags for a subset of types
11621. round up roughly to next 'grow step'
11622. Custom coercion for API
11623. NULL with zero length represents an empty string; NULL with higher * length is also now treated like an empty string although it is * a bit dubious. This is unlike duk_push_string() which pushes a * 'null' if the input string is a NULL.
11624. tv -> value just before prev top value; must relookup
11625. Init newly allocated slots (only).
11626. default: NULL, length 0
11627. * instanceof
11628. Assume value stack sizes (in elements) fits into duk_idx_t.
11629. useful for debugging
11630. **comment:** Relookup in case duk_js_tointeger() ends up e.g. coercing an object.
 label: code-design
11631. valstack limit caller has check, prevents wrapping
11632. default: NaN
11633. Coercion may be needed, the helper handles that by pushing the * tagged values to the stack.

11634. Note: this check relies on the fact that even a zero-size string * has a non-NULL pointer.
11635. Negative indices are always within allocated stack but * must not go below zero index.
11636. covers -Infinity
11637. * Get/require
11638. **comment:** * Number should already be in NaN-normalized form, * but let's normalize anyway.
 label: code-design
11639. Note: number has no explicit tag (in 8-byte representation)
11640. Positive index can be higher than valstack top but must * not go above allocated stack (equality is OK).
11641. Same as above but for unsigned int range.
11642. Errors are augmented when they are created, not when they are * thrown or re-thrown. The current error handler, however, runs * just before an error is thrown.
11643. **comment:** XXX: Hex encoded, length limited buffer summary here?
 label: code-design
11644. When src_size == 0, src_data may be NULL (if source * buffer is dynamic), and dst_data may be NULL (if * target buffer is dynamic). Avoid zero-size memcpy()
 * with an invalid pointer.
11645. check stack before interning (avoid hanging temp)
11646. **comment:** Try to make do with a stack buffer to avoid allocating a temporary buffer. * This works 99% of the time which is quite nice.
 label: code-design
11647. * Stack slice primitives
11648. **comment:** unused
 label: code-design
11649. normalize NaN which may not match our canonical internal NaN
11650. * Value stack top handling
11651. check_object_coercible
11652. **comment:** Note: Boolean prototype's internal value property is not writable, * but duk_xdef_prop_stridx() disregards the write protection. Boolean * instances are immutable. * String and buffer special behaviors are already enabled which is not * ideal, but a write to the internal value is not affected by them.
 label: code-design
11653. Note: this allows creation of internal strings.
11654. Check for maximum string length
11655. E5 Section 9.2
11656. force_exponential
11657. DUK_INVALID_INDEX won't be accepted as a valid index.
11658. only needed by debugger for now
11659. Raw helper for getting a value from the stack, checking its tag. * The tag cannot be a number because numbers don't have an internal * tag in the packed representation.
11660. DUK_USE_BUFFEROBJECT_SUPPORT
11661. Heap allocated: return heap pointer which is NOT useful * for the caller, except for debugging.
11662. * Forward declarations
11663. **comment:** This is a bit clunky because it is ANSI C portable. Should perhaps * relocate to another file because this is potentially platform * dependent.
 label: code-design
11664. For tv1 == tv2, this is a no-op (no explicit check needed).
11665. require
11666. **comment:** XXX: typing
 label: code-design
11667. Use unsigned arithmetic to optimize comparison.
11668. **comment:** * Number should already be in NaN-normalized form, but let's * normalize anyway.
 label: code-design
11669. **comment:** XXX: take advantage of val being unsigned, no need to mask
 label: code-design
11670. Return value of 'sz' or more indicates output was (potentially) * truncated.
11671. DUK_BUFOBJ_INT32ARRAY
11672. nop
11673. Faster but value stack overruns are memory unsafe.
11674. E5 Section 9.4, ToInteger()
11675. Note: need to re-lookup because ToNumber() may have side effects
11676. Relookup in case coerce_func() has side effects, e.g. ends up coercing an object
11677. Buffer is kept as is, with the fixed/dynamic nature of the * buffer only changed if requested. An external buffer * is converted into a non-external dynamic buffer in a * duk_to_dynamic_buffer() call.
11678. * Type checking
11679. Double error
11680. no compact
11681. w/o refcounting
11682. [... func/retval] -> [...]
11683. Special coercion for Uint8ClampedArray.
11684. flags is unsigned
11685. Example: d=3.5, t=0.5 -> ret = (3 + 1) & 0xfe = 4 & 0xfe = 4 * Example: d=4.5, t=0.5 -> ret = (4 + 1) & 0xfe = 5 & 0xfe = 4
11686. **comment:** XXX: optimize loops
 label: code-design
11687. Handle change in value stack top. Respect value stack * initialization policy: 'undefined' above top. Note that * DECREF may cause a side effect that reallocates valstack, * so must relookup after DECREF.
11688. nbytes * <-----> * [... | p | x | x | q] * => [... | q | p | x | x]
11689. tagged null pointers should never occur
11690. no throw
11691. Care must be taken to avoid pointer wrapping in the index * validation. For instance, on a 32-bit platform with 8-byte * duk_tval the index 0x20000000UL would wrap the memory space * once.
11692. * Conversions and coercions * * The conversion/coercions are in-place operations on the value stack. * Some operations are implemented here directly, while others call a * helper in duk_js_ops.c after validating arguments.
11693. initialize built-ins - either by copying or creating new ones
11694. may throw an error
11695. **comment:** Set stack top within currently allocated range, but don't reallocate. * This is performance critical especially for call handling, so whenever * changing, profile and look at generated code.
 label: code-design
11696. **comment:** XXX: size optimize
 label: code-design
11697. **comment:** NUL terminator handling doesn't matter here
 label: code-design
11698. Index validation is strict, which differs from duk_equals(). * The strict behavior mimics how instanceof itself works, e.g. * it is a TypeError if rval is not a - callable- object. It would * be somewhat inconsistent if rval would be allowed to be * non-existent without a TypeError.
11699. **comment:** Clamping to zero makes the API more robust to calling code * calculation errors.

label: code-design

11700. Surround with parentheses like in JX, ensures NULL pointer * is distinguishable from null value ("(null)" vs "null").
11701. * Value stack resizing. ** This resizing happens above the current "top": the value stack can be * grown or shrunk, but the "top" is not affected. The value stack cannot * be resized to a size below the current "top". ** The low level reallocation primitive must carefully recompute all value * stack pointers, and must also work if ALL pointers are NULL. The resize * is quite tricky because the valstack realloc may cause a mark-and-sweep, * which may run finalizers. Running finalizers may resize the valstack * recursively (the same value stack we're working on). So, after realloc * returns, we know that the valstack "top" should still be the same (there * should not be live values above the "top"), but its underlying size and * pointer may have changed.
11702. Stack size decreases.
11703. format plus something to avoid just missing
11704. copy values (no overlap even if to_ctx == from_ctx; that's not * allowed now anyway)
11705. unreachable
11706. **comment:** XXX: this is probably a useful shared helper: for a * duk_hbufferobject, get a validated buffer pointer/length.
label: code-design
11707. Represent a null pointer as 'null' to be consistent with * the JX format variant. Native '%p' format for a NULL * pointer may be e.g. '(nil)'.
11708. **comment:** XXX: duk_ssize_t
label: code-design
11709. ... and its 'message' from an instance property
11710. DUK_BUFOBJ_DUKTAPE_BUFFER
11711. DUK_BUFOBJ_UINT16ARRAY
11712. nargs
11713. **comment:** * Push readable string summarizing duk_tval. The operation is side effect * free and will only throw from internal errors (e.g. out of memory). * This is used by e.g. property access code to summarize a key/base safely, * and is not intended to be fast (but small and safe).
label: code-design
11714. [buf? res]
11715. Avoid NaN-to-integer coercion as it is compiler specific.
11716. no prototype
11717. Check that there's room to push one value.
11718. Note: 'q' is top-1
11719. **comment:** XXX: This will now return false for non-numbers, even though they would * coerce to NaN (as a general rule). In particular, duk_get_number() * returns a NaN for non-numbers, so should this function also return * true for non-numbers?
label: code-design
11720. min_new_size
11721. since index non-negative
11722. not popped by side effect
11723. precision:shortest
11724. 'this' binding is just before current activation's bottom
11725. [... (sep) str1 str2 ... strN buf]
11726. no size check is necessary
11727. use stack allocated buffer to ensure reachability in errors (e.g. intern error)
11728. overwrite str1
11729. extra -1 for buffer
11730. **comment:** XXX: could write output in chunks with fewer ensure calls, * but relative benefit would be small here.
label: code-design
11731. Combined size of separators already overflows
11732. [... buf]
11733. always true, arg is unsigned
11734. start (incl) and end (excl) of trimmed part
11735. [... res]
11736. Impose a string maximum length, need to handle overflow * correctly.
11737. overwrite sep
11738. guaranteed by string limits
11739. XXX: this is quite clunky. Add Unicode helpers to scan backwards and * forwards with a callback to process codepoints?
11740. **comment:** XXX: could map/decode be unified with duk_unicode_support.c code? * Case conversion needs also the character surroundings though.
label: code-design
11741. entire string is whitespace
11742. A bit tricky overflow test, see doc/code-issues.rst.
11743. * String manipulation
11744. pointers for scanning
11745. wrapped
11746. This may happen when forward and backward scanning disagree * (possible for non-extended-UTF-8 strings).
11747. is_join
11748. **comment:** get rid of the strings early to minimize memory use before intern
label: code-design
11749. reasonable output estimate
11750. * Variable access
11751. [... val]
11752. always throw ReferenceError for unresolvable
11753. Return value would be pointless: because throw_flag==1, we always * throw if the identifier doesn't resolve.
11754. [... varname val]
11755. **comment:** XXX: tostring?
label: code-design
11756. [...]
11757. [... varname val this] (because throw_flag == 1, always resolved)
11758. -> [... varname val this]
11759. Outside any activation -> put to global.
11760. Outside any activation -> look up from global.
11761. -> [... val retval]
11762. Real world behavior for map(): trailing non-existent * elements don't invoke the user callback, but are still * counted towards result 'length'.
11763. stack top contains 'false'
11764. Steps 12 and 13: reorganize elements to make room for itemCount elements
11765. **comment:** Shared entry code for many Array built-ins. Note that length is left * on stack (it could be popped, but that's not necessary).
label: code-design
11766. **comment:** XXX: expensive check (also shared elsewhere - so add a shared internal API call?)
label: code-design
11767. swap elements; deal with non-existent elements correctly
11768. move pivot to its final place
11769. return ToObject(this)

11770. **comment:** Note: 'this' is not necessarily an Array object. The push() * algorithm is supposed to work for other kinds of objects too, * so the algorithm has e.g. an explicit update for the 'length' * property which is normally "magical" in arrays.
label: code-design

11771. -> [sep ToObject(this) len str]
11772. end of loop (careful with len==0)
11773. stack[0] = start * stack[1] = deleteCount * stack[2...nargs-1] = items * stack[nargs] = ToObject(this) -3 * stack[nargs+1] = ToUint32(length) -2 * stack[nargs+2] = result array -1
11774. for lastIndexOf, result may be -1 (mark immediate termination)
11775. According to E5.1 Section 15.4.4.4 nonexistent trailing * elements do not affect 'length' of the result. Test262 * and other engines disagree, so update idx_last here too.
11776. For now, restrict result array into 32-bit length range.
11777. avoid degenerate cases, so that (len - 1) won't underflow
11778. * shift()
11779. fixed offsets in valstack
11780. k+argCount-1; note that may be above 32-bit range
11781. intermediate join to avoid valstack overflow
11782. orig value
11783. fromPresent = true
11784. XXX: len >= 0x80000000 won't work below because we need to be * able to represent -len.
11785. right exists, [[Put]] regardless whether or not left exists
11786. [ToObject(this) item1 ... itemN arr item(i) item(i)[j]]
11787. topmost element is the result array already
11788. -> [sep ToObject(this) len sep str]
11789. [ToObject(this) item1 ... itemN arr]
11790. The lo/hi indices may be crossed and hi < 0 is possible at entry.
11791. **comment:** XXX: This helper is a bit awkward because the handling for the different iteration * callers is quite different. This now compiles to a bit less than 500 bytes, so with * 5 callers the net result is about 100 bytes / caller.
label: code-design

11792. [sep ToObject(this) len sep result]
11793. **comment:** XXX: there's no explicit recursion bound here now. For the average * qsort recursion depth O(log n) that's not really necessary: e.g. for * 2**32 recursion depth would be about 32 which is OK. However, qsort * worst case recursion depth is O(n) which may be a problem.
label: code-design

11794. Index clamping is a bit tricky, we must ensure that we'll only iterate * through elements that exist and that the specific requirements from E5.1 * Sections 15.4.4.14 and 15.4.4.15 are fulfilled; especially: * * - indexOf: clamp to [-len,len], negative handling -> [0,len], * if clamped result is len, for-loop bails out immediately * * - lastIndexOf: clamp to [-len-1, len-1], negative handling -> [-1, len-1], * if clamped result is -1, for-loop bails out immediately * * If fromIndex is not given, ToInteger(undefined) = 0, which is correct * for indexOf() but incorrect for lastIndexOf(). Hence special handling, * and why lastIndexOf() needs to be a vararg function.
11795. **comment:** XXX: best behavior for real world compatibility?
label: code-design

11796. Range limited to [0, 0xffffffff] range, i.e. range that can be * represented with duk_int32_t. Use this when the method doesn't * handle the full 32-bit unsigned range correctly.
11797. **comment:** Fall back to the initial (original) Object.toString(). We don't * currently have pointers to the built-in functions, only the top * level global objects (like "Array") so this is now done in a bit * of a hacky manner. It would be cleaner to push the (original) * function and use duk_call_method().
label: code-design

11798. -> [... toLocaleString ToObject(val)]
11799. Note that 'l' and 'r' may cross, i.e. r < l
11800. **comment:** XXX: the insert here is a bit expensive if there are a lot of items. * It could also be special cased in the outermost for loop quite easily * (as the element is dup()'d anyway).
label: code-design

11801. In some cases it may be that lo > hi, or hi < 0; these * degenerate cases happen e.g. for empty arrays, and in * recursion leaves.
11802. -> [... ToObject(this) ToUint32(length)]
11803. trivial cases
11804. find elements to swap
11805. args start at index 2
11806. indexOf: clamp fromIndex to [-len, len] * (if fromIndex == len, for-loop terminates directly) * * lastIndexOf: clamp fromIndex to [-len - 1, len - 1] * (if clamped to -len-1 -> fromIndex becomes -1, terminates for-loop directly)
11807. [... ToObject(this) ToUint32(length) val]
11808. **comment:** XXX: an array can have length higher than 32 bits; this is not handled * correctly now.
label: code-design

11809. **comment:** XXX: this compiles to over 500 bytes now, even without special handling * for an array part. Uses signed ints so does not handle full array range correctly.
label: code-design

11810. For len == 0, i is initialized to len - 1 which underflows. * The condition (i < len) will then exit the for-loop on the * first round which is correct. Similarly, loop termination * happens by i underflowing.
11811. Step 16: update length; note that the final length may be above 32 bit range * (but we checked above that this isn't the case here)
11812. stack top contains 'true'
11813. stack[0] = start * stack[1] = end * stack[2] = ToObject(this) * stack[3] = ToUint32(length) * stack[4] = result array
11814. Step 9: copy elements-to-be-deleted into the result array
11815. retval is directly usable
11816. result array is already at the top of stack
11817. has access to 'this' binding
11818. stack[0...nargs-1] = unshift args (vararg) * stack[nargs] = ToObject(this) * stack[nargs+1] = ToUint32(length)
11819. The original value needs to be preserved for filter(), hence * this funny order. We can't re-get the value because of side * effects.
11820. * splice()
11821. * isArray()
11822. The E5.1 Section 15.4.4.4 algorithm doesn't set the length explicitly * in the end, but because we're operating with an internal value which * is known to be an array, this should be equivalent.
11823. Debug print which visualizes the qsort partitioning process.
11824. for indexOf, ToInteger(undefined) would be 0, i.e. correct, but * handle both indexOf and lastIndexOf specially here.
11825. -> [sep ToObject(this) len str sep]
11826. stack[0] = compareFn * stack[1] = ToObject(this) * stack[2] = ToUint32(length)
11827. * indexOf(), lastIndexOf()
11828. move pivot out of the way
11829. **comment:** XXX: must be able to represent -len
label: code-design

11830. * reverse()
11831. !(l < p)
11832. XXX: len >= 0x80000000 won't work below because a signed type * is needed by qsort.

11833. result index for filter()
11834. loop iterator init and limit changed from standard algorithm
11835. [... val callback thisArg val i obj]
11836. !(p < r)
11837. [ToObject(this) ToUint32(length) lowerValue upperValue]
11838. **comment:** XXX: could duk_is_undefined() provide defaulting undefined to 'len' * (the upper limit)?
 label: code-design
11839. Perform an intermediate join when this many elements have been pushed * on the value stack.
11840. **comment:** stack[0] = callback * stack[1] = thisArg * stack[2] = object * stack[3] = ToUint32(length) (unused, but avoid unnecessary pop) * stack[4] = result array (or undefined)
 label: code-design
11841. XXX: if 'len' is low, may want to ensure array part is kept: * the caller is likely to want a dense array.
11842. string compare is the default (a bit oddly)
11843. * pop(), push()
11844. [arg1 ... argN obj length new_length]
11845. lastIndexOf() needs to be a vararg function because we must distinguish * between an undefined fromIndex and a "not given" fromIndex; indexOf() is * made vararg for symmetry although it doesn't strictly need to be.
11846. E5.1 standard behavior when deleteCount is not given would be * to treat it just like if 'undefined' was given, which coerces * ultimately to 0. Real world behavior is to splice to the end * of array, see test-bi-array-proto-splice-no-delcount.js.
11847. no need to check callable; duk_call() will do that
11848. if thisArg not supplied, behave as if undefined was supplied
11849. * reduce(), reduceRight()
11850. [sep ToObject(this) len]
11851. NOTE: The Array special behaviors are NOT invoked by duk_xdef_prop_index() * (which differs from the official algorithm). If no error is thrown, this * doesn't matter as the length is updated at the end. However, if an error * is thrown, the length will be unset. That shouldn't matter because the * caller won't get a reference to the intermediate value.
11852. -> [... fn x y]
11853. -> [... res]
11854. stack[0] = callback fn * stack[1] = initialValue * stack[2] = object (coerced this) * stack[3] = length (not needed, but not popped above) * stack[4] = accumulator
11855. idx_step is +1 for indexOf, -1 for lastIndexOf
11856. [A B C D E F G H] rel_index = 2, del_count 3, item count 3 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)
11857. * concat()
11858. [sep ToObject(this) len sep str0 ... str(count-1)]
11859. * sort() ** Currently qsort with random pivot. This is now really, really slow, * because there is no fast path for array parts. ** Signed indices are used because qsort() leaves and degenerate cases * may use a negative offset.
11860. [... func this]
11861. stack[0] = searchElement * stack[1] = fromIndex * stack[2] = object * stack[3] = length (not needed, but not popped above)
11862. [ToUint32(len) array ToUint32(len)] -> [ToUint32(len) array]
11863. idx_step is +1 for reduce, -1 for reduceRight
11864. The extra (+4) is tight.
11865. -> [ToObject(this)]
11866. [sep ToObject(this) len sep]
11867. XXX: can shift() / unshift() use the same helper? * shift() is (close to?) <-> splice(0, 1) * unshift is (close to?) <-> splice(0, 0, [items])?
11868. stack[0] = object (this) * stack[1] = ToUint32(length) * stack[2] = elem at index 0 (retval)
11869. randomized pivot selection
11870. **comment:** XXX: optimize by creating array into correct size directly, and * operating on the array part directly; values can be memcpy()'d from * value stack directly as long as recounts are increased.
 label: code-design
11871. * slice()
11872. * unshift()
11873. retval to result[i]
11874. * Constructor
11875. if true, the stack already contains the final result
11876. * every(), some(), forEach(), map(), filter()
11877. * toString()
11878. Strict standard behavior, ignore trailing elements for * result 'length'.
11879. [A B C D E F G H] rel_index = 2, del_count 3, item count 4 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F F G H] (actual result at this point)
11880. fixed offset in valstack
11881. rnd in [lo,hi]
11882. Standard behavior for map(): trailing non-existent * elements don't invoke the user callback and are not * counted towards result 'length'.
11883. -> [... ToObject(this) ToUint32(length) final_len final_len]
11884. nop
11885. Technically Array.prototype.push() can create an Array with length * longer than 2^32-1, i.e. outside the 32-bit range. The final length * is *not* wrapped to 32 bits in the specification. ** This implementation tracks length with a uint32 because it's much * more practical. ** See: test-bi-array-push-maxlen.js.
11886. The specification is a bit vague what to do if the return * value is not a number. Other implementations seem to * tolerate non-numbers but e.g. V8 won't apparently do a * ToNumber().
11887. We're a varargs function because we need to detect whether * initialValue was given or not.
11888. -> [... ToObject(this) ToUint32(length) arg[i]]
11889. [arg1 ... argN obj length]
11890. throw flag irrelevant (false in std alg)
11891. Step 15: insert itemCount elements into the hole made above
11892. For join(), nargs is 1. For toLocaleString(), nargs is 0 and * setting the top essentially pushes an undefined to the stack, * thus defaulting to a comma separator.
11893. **comment:** XXX: 'this' will be ToObject() coerced twice, which is incorrect * but should have no visible side effects.
 label: code-design
11894. [A B C D E F G H] rel_index = 2, del_count 3, item count 1 * -> [A B F G H] (conceptual intermediate step) * -> [A B . F G H] (placeholder marked) * [A B C F G H] (actual result at this point, C will be replaced)
11895. If len <= 1, middle will be 0 and for-loop bails out * immediately (0 < 0 -> false).
11896. Note: unshift() may operate on indices above unsigned 32-bit range * and the final length may be >= 2**32. However, we restrict the * final result to 32-bit range for practicality.
11897. i >= 0 would always be true
11898. **comment:** Fast exit if indices are identical. This is valid for a non-existent property, * for an undefined value, and almost always for ToString() coerced comparison of * arbitrary values (corner cases where this is not the case include e.g. a an * object with varying ToString() coercion). ** The specification does not prohibit "caching" of values read from the array, so * assuming equality for comparing an index with itself falls into the category of * "caching". ** Also, compareFn may be inconsistent, so skipping a call to compareFn here may * have an effect on the final result. The specification does not require any * specific behavior for inconsistent compare functions, so again, this fast path * is OK.
 label: code-design

11899. fromPresent = false
11900. -> [... x y fn]
11901. [... this func]
11902. each call this helper serves has nargs==2
11903. -> [ToObject(this) item1 ... itemN arr]
11904. [ToObject(this) item1 ... itemN arr item(i)]
11905. * join(), toLocaleString() ** Note: checking valstack is necessary, but only in the per-element loop. ** Note: the trivial approach of pushing all the elements on the value stack * and then calling duk_join() fails when the array contains a large number * of elements. This problem can't be offloaded to duk_join() because the * elements to join must be handled here and have special handling. Current * approach is to do intermediate joins with very large number of elements. * There is no fancy handling; the prefix gets re-joined multiple times.
11906. **comment:** * Array built-ins ** Note that most Array built-ins are intentionally generic and work even * when the 'this' binding is not an Array instance. To ensure this, * Array algorithms do not assume "magical" Array behavior for the "length" * property, for instance. ** XXX: the "Throw" flag should be set for (almost?) all [[Put]] and * [[Delete]] operations, but it's currently false throughout. Go through * all put/delete cases and check throw flag use. Need a new API primitive * which allows throws flag to be specified. ** XXX: array lengths above 2G won't work reliably. There are many places * where one needs a full signed 32-bit range ([-0xffffffff, 0xffffffff], * i.e. -33- bits). Although array 'length' cannot be written to be outside * the unsigned 32-bit range (E5.1 Section 15.4.5.1 throws a RangeError if so) * some intermediate values may be above 0xffffffff and this may not be always * correctly handled now (duk_uint32_t is not enough for all algorithms). ** For instance, push() can legitimately write entries beyond length 0xffffffff * and cause a RangeError only at the end. To do this properly, the current * push() implementation tracks the array index using a 'double' instead of a * duk_uint32_t (which is somewhat awkward). See test-bi-array-push-maxlen.js. ** On using "put" vs. "def" prop * ===== * * Code below must be careful to use the appropriate primitive as it matters * for compliance. When using "put" there may be inherited properties in * Array.prototype which cause side effects when values are written. When * using "define" there are no such side effects, and many test262 test cases * check for this (for real world code, such side effects are very rare). * Both "put" and "define" are used in the E5.1 specification; as a rule, * "put" is used when modifying an existing array (or a non-array 'this' * binding) and "define" for setting values into a fresh result array. ** Also note that Array instance 'length' should be writable, but not * enumerable and definitely not configurable: even Duktape code internally * assumes that an Array instance will always have a 'length' property. * Preventing deletion of the property is critical.
label: code-design
11907. XXX: proper flags?
11908. * Boolean built-ins
11909. unbalanced stack
11910. -> [val obj val]
11911. **comment:** XXX: there is room to use a shared helper here, many built-ins * check the 'this' type, and if it's an object, check its class, * then get its internal value, etc.
label: code-design
11912. Shared helper to provide toString() and valueOf(). Checks 'this', gets * the primitive value to stack top, and optionally coerces with ToString().
11913. **comment:** XXX: helper; rely on Boolean.prototype as being non-writable, non-configurable
label: code-design
11914. * Duktape.Buffer: toString(), valueOf()
11915. [value offset noAssert], when ftype != DUK_FLD_VARINT
11916. reachable so pop OK
11917. Kronos/ES6 requires zeroing even when DUK_USE_ZERO_BUFFER_DATA * is not set.
11918. throw_flag
11919. **comment:** XXX: encoding is ignored now.
label: code-design
11920. This behavior mostly mimics Node.js now.
11921. nargs
11922. Fast path: source is a TypedArray (or any bufferobject).
11923. NOTE! Caller must ensure that any side effects from the * coercions below are safe. If that cannot be guaranteed * (which is normally the case), caller must coerce the * argument using duk_to_number() before any pointer * validations; the result of duk_to_number() always coerces * without side effects here.
11924. Don't accept relative indices now.
11925. bits 0...1: shift
11926. default is explicit index read/write copy
11927. Clamp to target's end if too long. ** NOTE: there's no overflow possibility in the comparison; * both target_ustart and copy_size are >= 0 and based on * values in duk_int_t range. Adding them as duk_uint_t * values is then guaranteed not to overflow.
11928. [offset noAssert], when ftype != DUK_FLD_VARINT
11929. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT32
11930. Select copy mode. Must take into account element * compatibility and validity of the underlying source * buffer.
11931. For the case n==1 Node.js doesn't seem to type check * the sole member but we do it before returning it. * For this case only the original buffer object is * returned (not a copy).
11932. Push the resulting view object and attach the ArrayBuffer.
11933. DUK_FLD_32BIT
11934. >= 0
11935. arg count
11936. DUK_FLD_8BIT
11937. Neutered. We could go into the switch-case safely with * buf == NULL because check_length == 0. To avoid scanbuild * warnings, fail directly instead.
11938. * TypedArray.prototype.set() ** TypedArray set() is pretty interesting to implement because: ** - The source argument may be a plain array or a typedarray. If the * source is a TypedArray, values are decoded and re-encoded into the * target (not as a plain byte copy). This may happen even when the * element byte size is the same, e.g. integer values may be re-encoded * into floats. ** - Source and target may refer to the same underlying buffer, so that * the set() operation may overlap. The specification requires that this * must work as if a copy was made before the operation. Note that this * is NOT a simple memmove() situation because the source and target * byte sizes may be different -- e.g. a 4-byte source (Int8Array) may * expand to a 16-byte target (Uint32Array) so that the target overlaps * the source both from beginning and the end (unlike in typical memmove). ** - Even if 'buf' pointers of the source and target differ, there's no * guarantee that their memory areas don't overlap. This may be the * case with external buffers. ** Even so, it is nice to optimize for the common case: ** - Source and target separate buffers or non-overlapping. ** - Source and target have a compatible type so that a plain byte copy * is possible. Note that while e.g. uint8 and int8 are compatible * (coercion one way or another doesn't change the byte representation), * e.g. int8 and uint8clamped are NOT compatible when writing int8 * values into uint8clamped typedarray (-1 would clamp to 0 for instance). ** See test-bi-typedarray-proto-set.js.
11939. Non-object argument is simply int coerced, matches * V8 behavior (except for "null", which we coerce to * 0 but V8 TypeErrors).
11940. Array or Array-like
11941. **comment:** Static call style.
label: code-design
11942. [offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
11943. sign doesn't matter when writing
11944. Format of magic, bits: * 0...1: elem size shift (0-3) * 2...5: elem type (DUK_HBUFFEROBJECT_ELEM_xxx)
11945. **comment:** Slow gathering of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. Handling * of negative numbers is a bit non-obvious in both cases.
label: code-design
11946. Resolve start/end offset as element indices first; arguments * at idx_start/idx_end are element offsets. Working with element * indices first also avoids potential for wrapping.
11947. Write in little endian
11948. [targetBuffer targetStart sourceStart sourceEnd]
11949. source starts after dest ends

11950. * Misc helpers
11951. For TypedArrays 'undefined' return value is specified * by ES6 (matches V8).
11952. Check that value is a duk_hbufferobject and return a pointer to it.
11953. **comment:** Fill offset handling is more lenient than in Node.js.
 label: code-design
11954. byte sizes will match
11955. return h_bufres
11956. Must be element size multiple from * start offset to end of buffer.
11957. * Node.js Buffer: toString([encoding], [start], [end])
11958. We want to do a straight memory copy if possible: this is * an important operation because .set() is the TypedArray * way to copy chunks of memory. However, because set() * conceptually works in terms of elements, not all views are * compatible with direct byte copying. ** If we do manage a direct copy, the "overlap issue" handled * below can just be solved using memmove() because the source * and destination element sizes are necessarily equal.
11959. new buffer with string contents
11960. XXX: could add fast path for u8 compatible views
11961. A validated read() is always a number, so it's write coercion * is always side effect free and won't invalidate pointers etc.
11962. Slice offsets are element (not byte) offsets, which only matters * for TypedArray views, Node.js Buffer and ArrayBuffer have shift * zero so byte and element offsets are the same. Negative indices * are counted from end of slice, crossed indices are allowed (and * result in zero length result), and final values are clamped * against the current slice. There's intentionally no check * against the underlying buffer here.
11963. We could fit built-in index into magic but that'd make the magic * number dependent on built-in numbering (genbuiltins.py doesn't * handle that yet). So map both class and prototype from the * element type.
11964. bytes in dest
11965. Use byte copy.
11966. Argument variants. When the argument is an ArrayBuffer a view to * the same buffer is created; otherwise a new ArrayBuffer is always * created.
11967. Accept plain buffer values like array initializers * (new in Duktape 1.4.0).
11968. Shared lenient buffer length clamping helper. Indices are treated as * element indices (though output values are byte offsets) which only * really matters for TypedArray views as other buffer object have a zero * shift. Negative indices are counted from end of input slice; crossed * indices are clamped to zero length; and final indices are clamped * against input slice. Used for e.g. ArrayBuffer slice().
11969. 0xff => 255 - 256 = -1; 0x80 => 128 - 256 = -128
11970. DUK_FLD_16BIT
11971. Update 'buffer_length' to be the effective, safe limit which * takes into account the underlying buffer. This value will be * potentially invalidated by any side effect.
11972. xxx -> DUK_HBUFFEROBJECT_ELEM_INT32
11973. avoid side effects!
11974. Node.js accepts only actual Arrays.
11975. **comment:** XXX: regetting the pointer may be overkill - we're writing * to a side-effect free array here.
 label: code-design
11976. crossed offsets or zero size
11977. Must use memmove() because copy area may overlap (source and target * buffer may be the same, or from different slices).
11978. Gather in little endian
11979. unsigned
11980. We need 'nbytes' even for a failed offset; return value must be * (offset + nbytes) even when write fails due to invalid offset.
11981. **comment:** XXX: function flag to make this automatic?
 label: requirement
11982. XXX: happens e.g. when evaluating: String(Buffer.prototype).
11983. For all duk_hbufferobjects, get the plain buffer inside * without making a copy. This is compatible with Duktape 1.2 * but means that a slice/view information is ignored and the * full underlying buffer is returned. * If called as a constructor, a new Duktape.Buffer object * pointing to the same plain buffer is created below.
11984. DUK_FLD_VARINT; not relevant here
11985. Shared helper.
11986. **comment:** ArrayBuffer argument is handled specially above; the rest of the * argument variants are handled by shared code below.
 label: code-design
11987. Copy values by index reads and writes. Let virtual * property handling take care of coercion.
11988. Accept any duk_hbufferobject, though we're only normally * called for Duktape.Buffer values.
11989. idx_end
11990. may be NULL
11991. -> [buffer]
11992. [offset value littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)
11993. Node.js return value for noAssert out-of-bounds reads is * usually (but not always) NaN. Return NaN consistently.
11994. Copy values through direct validated reads and writes.
11995. Return the Buffer to allow chaining: b.fill(0x11).fill(0x22, 3, 5).toString()
11996. Gather in big endian
11997. compare: similar to string comparison but for buffer data.
11998. **comment:** * Node.js Buffer.prototype.equals() * Node.js Buffer.prototype.compare() * Node.js Buffer.compare()
 label: requirement
11999. use p_src_base from now on
12000. ignore encoding for now
12001. * Node.js Buffer.isBuffer()
12002. * Node.js Buffer.prototype.fill()
12003. * Duktape.Buffer, Node.js Buffer, and Khronos/ES6 TypedArray built-ins
12004. * Node.js Buffer.isEncoding()
12005. TypedArray (or other non-ArrayBuffer duk_hbufferobject). * Conceptually same behavior as for an Array-like argument, * with a few fast paths.
12006. 1=little endian
12007. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT64
12008. * Node.js Buffer.prototype.write(string, [offset], [length], [encoding])
12009. Not enough data.
12010. Note: start_offset/end_offset can still be < 0 here.
12011. bits 2...5: type
12012. **comment:** XXX: V8 throws a TypeError for negative values. Would it * be more useful to interpret negative offsets here from the * end of the buffer too?
 label: code-design
12013. Push a new ArrayBuffer (becomes view .buffer)
12014. Overflow not an issue because subtraction is used on the right * side and guaranteed to be >= 0.
12015. idx_start
12016. should never happen but default here
12017. Now we can check offset validity.
12018. Slow path: quite slow, but we save space by using the property code * to write coerce target values. We don't need to worry about overlap * here because the source is not a TypedArray. ** We could use the bufferobject write coercion helper but since the * property read may have arbitrary side effects, full validity checks * would be needed for every element anyway.
12019. [value offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
12020. inherit
12021. * Node.js Buffer.prototype.copy()

12022. **comment:** not covered, return all zeroes
label: test

12023. [value offset end]

12024. **comment:** * Node.js Buffer.concat()
label: code-design

12025. Value stack intentionally mixed size here.

12026. [array totalLength]

12027. * Shared readfield and writefield methods * * The readfield/writefield methods need support for endianness and field * types. All offsets are byte based so no offset shifting is needed.

12028. not reachable

12029. * Node.js Buffer.prototype: toJSON()

12030. Check actual underlying buffers for validity and that they * cover the copy. No side effects are allowed after the check * so that the validity status doesn't change.

12031. length check

12032. We want to avoid making a copy to process set() but that's * not always possible: the source and the target may overlap * and because element sizes are different, the overlap cannot * always be handled with a memmove() or choosing the copy * direction in a certain way. For example, if source type is * uint8 and target type is uint32, the target area may exceed * the source area from both ends! * * Note that because external buffers may point to the same * memory areas, we must ultimately make this check using * pointers. * * NOTE: careful with side effects: any side effect may cause * a buffer resize (or external buffer pointer/length update)!

12033. There's no need to check for buffer validity status for the * target here: the property access code will do that for each * element. Moreover, if we did check the validity here, side * effects from reading the source argument might invalidate * the results anyway.

12034. Must be ≥ 0 and multiple of element size.

12035. * Duktape.Buffer: constructor

12036. Ready to make the copy. We must proceed element by element * and must avoid any side effects that might cause the buffer * validity check above to become invalid. * * Although we work through the value stack here, only plain * numbers are handled which should be side effect safe.

12037. * Constructor arguments are currently somewhat compatible with * (keep it that way if possible): * * <http://nodejs.org/api/buffer.html> * * Note that the ToBuffer() coercion (duk_to_buffer()) does NOT match * the constructor behavior.

12038. Handle single character fills as memset() even when * the fill data comes from a one-char argument.

12039. Just skip, leaving zeroes in the result.

12040. Equality may be OK but $>\text{length}$ not. Checking * this explicitly avoids some overflow cases * below.

12041. Byte length would overflow.

12042. Set .buffer

12043. equals

12044. We want to compare the slice/view areas of the arguments. * If either slice/view is invalid (underlying buffer is shorter) * ensure equals() is false, but otherwise the only thing that * matters is to be memory safe.

12045. **comment:** Format of magic, bits: * 0..1: field type; 0:uint8, 1:uint16, 2:uint32, 3=float, 4=double, 5=unused, 6=unused, 7=unused * 3: endianness: 0=little, 1=big * 4: signed: 1=yes, 0=no * 5: typedarray: 1=yes, 0=no
label: code-design

12046. Shared offset/length coercion helper.

12047. bytes in source

12048. Source end clamped silently to available length.

12049. **comment:** Internal class is Object: Object.prototype.toString.call(new Buffer(0)) * prints "[object Object]".
label: code-design

12050. elems in source and dest

12051. XXX: fast path for array arguments?

12052. either nonzero value is ok

12053. Handle TypedArray vs. Node.js Buffer arg differences

12054. Negative offsets cause a RangeError.

12055. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8CLAMPED * Note: INT8 is -not- copy compatible, e.g. -1 would coerce to 0x00.

12056. Shared lenient buffer length clamping helper. No negative indices, no * element/byte shifting.

12057. * Node.js Buffer.prototype.slice([start], [end]) * ArrayBuffer.prototype.slice(begin, [end]) * TypedArray.prototype.slice(begin, [end]) * * The API calls are almost identical; negative indices are counted from end * of buffer, and final indices are clamped (allowing crossed indices). Main * differences: * * - Copy/view behavior; Node.js .slice() and TypedArray .subarray() create * views, ArrayBuffer .slice() creates a copy * * - Resulting object has a different class and prototype depending on the * call (or 'this' argument) * * - TypedArray .subarray() arguments are element indices, not byte offsets

12058. offset/value order different from Node.js

12059. Node.js Buffer: return offset + #bytes written (i.e. next * write offset).

12060. Check that 'this' is a duk_hbufferobject and return a pointer to it * (NULL if not).

12061. ArrayBuffer: unlike any other argument variant, create * a view into the existing buffer.

12062. one i_step added at top of loop

12063. **comment:** unnecessary div for last byte
label: code-design

12064. **comment:** Serialize uncovered backing buffer as a null; doesn't * really matter as long we're memory safe.
label: code-design

12065. Ensure copy is covered by underlying buffers.

12066. At the moment Buffer(<str>) will just use the string bytes as * is (ignoring encoding), so we return the string length here * unconditionally.

12067. **comment:** Node.js Buffer variable width integer field. We don't really * care about speed here, so aim for shortest algorithm.
label: code-design

12068. **comment:** XXX: accept any duk_hbufferobject type as an input also?
label: code-design

12069. User totalLength overrides a computed length, but we'll check * every copy in the copy loop. Note that duk_to_uint() can * technically have arbitrary side effects so we need to recheck * the buffers in the copy loop.

12070. * ArrayBuffer, DataView, and TypedArray constructors

12071. guaranteed by duk_to_string()

12072. Map DUK_HBUFFEROBJECT_ELEM_xxx to prototype object built-in index. * Sync with duk_hbufferobject.h

12073. -> [array totalLength buf]

12074. source ends before dest starts

12075. Bitfield for each DUK_HBUFFEROBJECT_ELEM_xxx indicating which element types * are compatible with a blind byte copy for the TypedArray set() method (also * used for TypedArray constructor). Array index is target buffer elem type, * bitfield indicates compatible source types. The types must have same byte * size and they must be coercion compatible.

12076. source out-of-bounds (but positive)

12077. Map DUK_FLX_xxx to byte size.

12078. **comment:** The DataView .buffer property is ordinarily set to the argument * which is an ArrayBuffer. We accept any duk_hbufferobject as * an argument and .buffer will be set to the argument regardless * of what it is. This may be a bit confusing if the argument * is e.g. a DataView or another TypedArray view. * * XXX: Copy .buffer property from a DataView/TypedArray argument? * Create a fresh ArrayBuffer for Duktape.Buffer and Node.js Buffer * arguments? See: test-bug-dataview-buffer-prop.js.
label: code-design

12079. **comment:** XXX: split into separate functions for each field type
label: code-design

12080. [start end]

12081. [array totalLength bufres buf]
12082. Offset is coerced first to signed integer range and then to unsigned. * This ensures we can add a small byte length (1-8) to the offset in * bound checks and not wrap.
12083. one i_step over
12084. default to false
12085. Unlike for negative arguments, some call sites * want length to be clamped if it's positive.
12086. For n == 0, Node.js ignores totalLength argument and * returns a zero length buffer.
12087. Return value is like write(), number of bytes written. * The return value matters because of code like: * "off += buf.copy(...)".
12088. xxx -> DUK_HBUFFEROBJECT_ELEM_INT8
12089. xxx -> DUK_HBUFFEROBJECT_ELEM_INT16
12090. These are not needed when only Duktape.Buffer is supported.
12091. DUK_FLD_FLOAT
12092. undefined coerces to zero which is correct
12093. Copy values, the copy method depends on the arguments. * * Copy mode decision may depend on the validity of the underlying * buffer of the source argument; there must be no harmful side effects * from there to here for copy_mode to still be valid.
12094. DUK_FLD_DOUBLE
12095. Map DUK_HBUFFEROBJECT_ELEM_xxx to duk_hobject class number. * Sync with duk_hbufferobject.h and duk_hobject.h.
12096. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT32
12097. [offset littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)
12098. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8
12099. even for zero-length string
12100. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT16
12101. new buffer of specified size
12102. pop plain buffer, now reachable through h_bufres
12103. Node.js return value for failed writes is offset + #bytes * that would have been written.
12104. DUK_USE_BUFFEROBJECT_SUPPORT
12105. Custom behavior: plain buffer is used as internal buffer * without making a copy (matches Duktape.Buffer).
12106. ignore_loop
12107. Shift to sign extend.
12108. No copy, leave zero bytes in the buffer. There's no * ambiguity with Float32/Float64 because zero bytes also * represent 0.0.
12109. **comment:** XXX: for negative input offsets, 'offset' will be a large * positive value so the result here is confusing.
 label: code-design
12110. no overflow
12111. **comment:** Copy the .buffer property, needed for TypedArray.prototype.subarray(). * * XXX: limit copy only for TypedArray classes specifically?
 label: code-design
12112. stack is unbalanced, but: [<something> buf]
12113. **comment:** The condition could be more narrow and check for the * copy area only, but there's no need for fine grained * behavior when the underlying buffer is misconfigured.
 label: code-design
12114. Set .buffer to the argument ArrayBuffer.
12115. **comment:** unnecessary shift for last byte
 label: code-design
12116. **comment:** Slow writing of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. There's * no special sign handling when writing varints.
 label: code-design
12117. * ArrayBuffer.isView()
12118. **comment:** XXX: easier check with less code?
 label: code-design
12119. **comment:** The resulting buffer object gets the same class and prototype as * the buffer in 'this', e.g. if the input is a Node.js Buffer the * result is a Node.js Buffer; if the input is a Float32Array, the * result is a Float32Array. * * For the class number this seems correct. The internal prototype * is not so clear: if 'this' is a bufferobject with a non-standard * prototype object, that value gets copied over into the result * (instead of using the standard prototype for that object type).
 label: code-design
12120. return input buffer, converted to a Duktape.Buffer object * if called as a constructor (no change if called as a * function).
12121. * Indexed read/write helpers (also used from outside this file)
12122. * Node.js Buffer: constructor
12123. Write in big endian
12124. replaced
12125. * Node.js Buffer.byteLength()
12126. non-zero: make copy
12127. There's overlap: the desired end result is that * conceptually a copy is made to avoid "trampling" * of source data by destination writes. We make * an actual temporary copy to handle this case.
12128. Note: unbalanced stack on purpose
12129. Copy slice, respecting underlying buffer limits; remainder * is left as zero.
12130. Nominal size check.
12131. Coerce value to a number before computing check_length, so that * the field type specific coercion below can't have side effects * that would invalidate check_length.
12132. **comment:** Neutered checks not necessary here: neutered buffers have * zero 'length' so we'll effectively skip them.
 label: code-design
12133. Check that 'this' is a duk_hbufferobject and return a pointer to it.
12134. **comment:** XXX: The duk_to_number() cast followed by integer coercion * is platform specific so NaN, +/- Infinity, and out-of-bounds * values result in platform specific output now. * See: test-bi-nodejs-buffer-proto-varint-special.js
 label: code-design
12135. as elements
12136. Argument must be a string, e.g. a buffer is not allowed.
12137. target out-of-bounds (but positive)
12138. unbalanced stack on purpose
12139. Compute result length and validate argument buffers.
12140. only accept lowercase 'utf8' now.
12141. Cannot overlap.
12142. Convert indices to byte offsets.
12143. no need to decref
12144. [... this]
12145. stack top: new time value, return 1 to allow tail calls
12146. Convert day from one-based to zero-based (internal). This may * cause the day part to be negative, which is OK.
12147. 3
12148. Apply timezone offset to get the main parts in UTC
12149. **comment:** XXX: there is a small risk here: because the ISO 8601 parser is * very loose, it may end up parsing some datetime values which * would be better parsed with a platform specific parser.

label: code-design

12150. 8: getFullYear
12151. now expected to fit into a 32-bit integer
12152. All components default to 0 except day-of-month which defaults * to 1. However, because our internal day-of-month is zero-based, * it also defaults to zero here.
12153. MakeDate
12154. 37: setUTCFullYear
12155. * Return $(t - \text{LocalTime}(t))$ in minutes: * * $t - \text{LocalTime}(t) = t - (t + \text{LocalTZA} + \text{DaylightSavingTA}(t)) * = -(\text{LocalTZA} + \text{DaylightSavingTA}(t))$ * * where DaylightSavingTA() is checked for time 't'. * * Note that the sign of the result is opposite to common usage, * e.g. for EE(S)T which normally is +2h or +3h from UTC, this * function returns -120 or -180. *
12156. Integer division which floors also negative values correctly.
12157. `""`-
12158. invalid value which never matches
12159. 7
12160. estimate year upwards (towards positive infinity), then back down; * two iterations should be enough
12161. Helper for string conversion calls: check 'this' binding, get the * internal time value, and format date and/or time in a few formats. * Return value allows tail calls.
12162. Here we'd have the option to normalize -0 to +0.
12163. $\rightarrow [\text{O} \text{ toISOString O}]$
12164. $\rightarrow [\text{timeval this} \text{ timeval }]$
12165. Optional UTC conversion.
12166. Note: %06d for positive value, %07d for negative value to include * sign and 6 digits.
12167. Internal timevalue is already NaN, so don't touch it.
12168. 5: toLocaleTimeString
12169. Old solution: don't iterate, incorrect
12170. 38: getYear
12171. 6
12172. * Other file level defines
12173. zero-based \rightarrow one-based
12174. called as a normal function: return new Date().toString()
12175. * ISO 8601 subset parser.
12176. 39: setYear
12177. Rule control flags.
12178. 1: toDateString
12179. E5.1 Section 15.9.1.6
12180. tzoffset seconds are dropped; 16 bits suffice for * time offset in minutes
12181. if-digit-else-ctrl
12182. Causes a ToNumber() coercion, but doesn't break coercion order since * year is coerced first anyway.
12183. Unlike most built-ins, the internal [[PrimitiveValue]] of a Date * is mutable.
12184. Parts are in local time, convert when setting.
12185. **comment:** * String/JSON conversions * * Human readable conversions are now basically ISO 8601 with a space * (instead of 'T') as the date/time separator. This is a good baseline * and is platform independent. * * A shared native helper to provide many conversions. Magic value contains * a set of flags. The helper provides: * * $\text{toString}()$ * $\text{toDateString}()$ * $\text{toTimeString}()$ * $\text{toLocaleString}()$ * $\text{toLocaleDateString}()$ * $\text{toLocaleTimeString}()$ * $\text{toUTCString}()$ * $\text{toISOString}()$ * * Notes: * * - Date.prototype.toGMTString() and Date.prototype.toUTCString() are * required to be the same EcmaScript function object (!), so it is * omitted from here. * * - Date.prototype.toUTCString(): E5.1 specification does not require a * specific format, but result should be human readable. The * specification suggests using ISO 8601 format with a space (instead of 'T') separator if a more human readable format is not available. * * - Date.prototype.toISOString(): unlike other conversion functions, * $\text{toISOString}()$ requires a RangeError for invalid date values.
label: code-design
12186. $\rightarrow [\dots \text{ this}]$
12187. 16: getHours
12188. Note2
12189. char loop
12190. 33: setUTCDate
12191. 25: setUTCMilliseconds
12192. Use explicit steps in computation to try to ensure that * computation happens with intermediate results coerced to * double values (instead of using something more accurate). * E.g. E5.1 Section 15.9.1.11 requires use of IEEE 754 * rules (= EcmaScript '+' and '*' operators). * * Without 'volatile' even this approach fails on some platforms * and compiler combinations. For instance, gcc 4.8.1 on Ubuntu * 64-bit, with -m32 and without -std=c99, test-bi-date-canceling.js * would fail because of some optimizations when computing tmp_time * (MakeTime below). Adding 'volatile' to tmp_time solved this * particular problem (annoyingly, also adding debug prints or * running the executable under valgrind hides it).
12193. match against rule part/sep bits
12194. Coerce all finite parts with ToInteger(). ToInteger() must not * be called for NaN/Infinity because it will convert e.g. NaN to * zero. If ToInteger() has already been called, this has no side * effects and is idempotent. * * Don't read dparts[DUK_DATE_IDX_WEEKDAY]; it will cause Valgrind * issues if the value is uninitialized.
12195. NaN timevalue: we need to coerce the arguments, but * the resulting internal timestamp needs to remain NaN. * This works but is not pretty: parts and dparts will * be partially uninitialized, but we only write to them.
12196. Rule table: first matching rule is used to determine what to do next.
12197. flags
12198. fits into duk_small_int_t
12199. 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.
12200. formatters always get one-based month/day-of-month
12201. Setter APIs detect special year numbers (0...99) and apply a +1900 * only in certain cases. The legacy getYear() getter applies -1900 * unconditionally.
12202. 2: toTimeString
12203. **comment:** TimeClip() should never be necessary
label: code-design
12204. continue matching, set neg_tzoffset flag
12205. $\rightarrow [\dots \text{ timeval_new}]$
12206. 30: setHours
12207. TimeClip(), which also handles Infinity \rightarrow NaN conversion
12208. unsigned
12209. zero-based day
12210. 34: setMonth
12211. Note: rely on index ordering
12212. Parser part masks.
12213. 20: getSeconds
12214. Expects 'this' at top of stack on entry.
12215. 28: setMinutes
12216. $t1 = \text{milliseconds}$ within day (fits 32 bit) * $t2 = \text{day number from epoch}$ (fits 32 bit, may be negative)
12217. * Indirect magic value lookup for Date methods. * * Date methods don't put their control flags into the function magic value * because they wouldn't fit into a LIGHFUNC's magic field. Instead, the * magic value is set to an index pointing to the array of control flags * below. * * This must be kept in strict sync with genbuiltins.py!

12218. Day-of-month is one-based in the API, but zero-based * internally, so fix here. Note that month is zero-based * both in the API and internally.
12219. 1
12220. * Determining which datetime components to overwrite based on * stack arguments is a bit complicated, but important to factor * out from setters themselves for compactness. ** If DUK_DATE_FLAG_TIMESETTER, maxnargs indicates setter type: ** 1 -> millisecond * 2 -> second, [millisecond] * 3 -> minute, [second], [millisecond] * 4 -> hour, [minute], [second], [millisecond] ** Else: ** 1 -> date * 2 -> month, [date] * 3 -> year, [month], [date] ** By comparing nargs and maxnargs (and flags) we know which * components to override. We rely on part index ordering.
12221. zero-based month
12222. Parser part count.
12223. 6: toUTCString
12224. no argument given -> leave components untouched
12225. msec
12226. Compute time value from (double) parts. The parts can be either UTC * or local time; if local, they need to be (conceptually) converted into * UTC time. The parts may represent valid or invalid time, and may be * wildly out of range (but may cancel each other and still come out in * the valid Date range).
12227. Matching separator index is used in the control table
12228. don't care value, year is mandatory
12229. **comment:** * Date built-ins ** Unlike most built-ins, Date has some platform dependencies for getting * UTC time, converting between UTC and local time, and parsing and * formatting time values. These are all abstracted behind DUK_USE_xxx * config options. There are built-in platform specific providers for * POSIX and Windows, but external providers can also be used. ** See doc/datetime.rst.
label: code-design
12230. SCANBUILD: complains about use of uninitialized values. * The complaint is correct, but operating in undefined * values here is intentional in some cases and the caller * ignores the results.
12231. 27: setUTCSeconds
12232. unreferenced with some options
12233. 36: setFullYear
12234. -> [this]
12235. Leaves new timevalue on stack top and returns 1, which is correct * for part setters.
12236. 4
12237. 21: getUTCSeconds
12238. seconds, doesn't fit into 16 bits
12239. Equivalent year for DST calculations outside [1970,2038[range, see * E5 Section 15.9.1.8. Equivalent year has the same leap-year-ness and * starts with the same weekday on Jan 1. * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
12240. may be NaN
12241. 3: toLocaleString
12242. 17: getUTCHours
12243. 22: getMilliseconds
12244. Set datetime parts from stack arguments, defaulting any missing values. * Day-of-week is not set; it is not required when setting the time value.
12245. 12: getDate
12246. There are at most 7 args, but we use 8 here so that also * DUK_DATE_IDX_WEEKDAY gets initialized (to zero) to avoid the potential * for any Valgrind gripes later.
12247. 35: setUTCMonth
12248. Special handling for year sign.
12249. caller checks
12250. 24: setMilliseconds
12251. MakeTime
12252. 0x08583b00
12253. 5
12254. If 'day' is NaN, returns NaN.
12255. **comment:** DUK_USE_DATE_GET_LOCAL_TZOFFSET() needs to be called with a * time value computed from UTC parts. At this point we only * have 'd' which is a time value computed from local parts, so * it is off by the UTC-to-local time offset which we don't know * yet. The current solution for computing the UTC-to-local * time offset is to iterate a few times and detect a fixed * point or a two-cycle loop (or a sanity iteration limit), * see test-bi-date-local-parts.js and test-bi-date-tzoffset-basic-fi.js. ** E5.1 Section 15.9.1.9: * UTC(t) = t - LocalTZA - DaylightSavingTA(t - LocalTZA) ** For NaN/inf, DUK_USE_DATE_GET_LOCAL_TZOFFSET() returns 0.
label: code-design
12256. 9: getUTCFullYear
12257. During parsing, month and day are one-based; set defaults here.
12258. The algorithm in E5.1 Section 15.9.1.12 normalizes month, but * does not normalize the day-of-month (nor check whether or not * it is finite) because it's not necessary for finding the day * number which matches the (year,month) pair. ** We assume that duk_day_from_year() is exact here. ** Without an explicit infinity / NaN check in the beginning, * day_num would be a bogus integer here. ** It's possible for 'year' to be out of integer range here. * If so, we need to return NaN without integer overflow. * This fixes test-bug-setyear-overflow.js.
12259. MakeDay
12260. Push 'this' binding, check that it is a Date object; then push the * internal time value. At the end, stack is: [... this timeval]. * Returns the time value. Local time adjustment is done if requested.
12261. Helper for component getter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), push a specified component as a return value to the * value stack and return 1 (caller can then tail call us).
12262. Different calling convention than above used because the helper * is shared.
12263. non-leap year: sunday, monday, ...
12264. Note1: the specification doesn't require matching a time form with * just hours ("HH"), but we accept it here, e.g. "2012-01-02T12Z". ** Note2: the specification doesn't require matching a timezone offset * with just hours ("HH"), but accept it here, e.g. "2012-01-02T03:04:05+02"
12265. * API oriented helpers
12266. 11: getUTCMonth
12267. leap year: sunday, monday, ...
12268. **comment:** This shouldn't be necessary, but check just in case * to avoid any chance of overruns.
label: code-design
12269. Note: DST adjustment is determined using UTC time.
12270. handle negative values
12271. 32: setDate
12272. Note: DST adjustment is determined using UTC time. * If 'd' is NaN, tzoffset will be 0.
12273. **comment:** XXX: idx_val would fit into 16 bits, but using duk_small_uint_t * might not generate better code due to casting.
label: code-design
12274. 7: toISOString
12275. **comment:** * Setters. ** Setters are a bit more complicated than getters. Component setters * break down the current time value into its (normalized) component * parts, replace one or more components with -unnormalized- new values, * and the components are then converted back into a time value. As an * example of using unnormalized values: ** var d = new Date(1234567890); ** is equivalent to: ** var d = new Date(0); * d.setUTCSeconds(1234567890); ** A shared native helper to provide almost all setters. Magic value * contains a set of flags and also packs the "maxnargs" argument. The * helper provides: ** setMilliseconds() * setUTCSeconds() * setSeconds() * setUTCSeconds() * setMinutes() * setUTCMinutes() * setHours() * setUTCHours() * setDate() * setUTCDate() * setMonth() * setUTCMonth() * setFullYear() * setUTCFullYear() * setYear() * Notes: ** - Date.prototype.setYear() (Section B addition): special year check * is omitted. NaN / Infinity will just flow through and ultimately * result in a NaN internal time value. ** - Date.prototype.setYear() does not have optional arguments

for * setting month and day-in-month (like setFullYear()), but we indicate * 'maxnargs' to be 3 to get the year written to the correct component * index in duk__set_part_helper(). The function has nargs == 1, so only * the year will be set regardless of actual argument count.

label: code-design

12276. 13: getUTCDate

12277. No locale specific formatter; this is OK, we fall back * to ISO 8601.

12278. 2

12279. Maximum iteration count for computing UTC-to-local time offset when * creating an Ecmascript time value from local parts.

12280. Assume that year, month, day are all coerced to whole numbers. * They may also be NaN or infinity, in which case this function * must return NaN or infinity to ensure time value becomes NaN. * If 'day' is NaN, the final return will end up returning a NaN, * so it doesn't need to be checked here.

12281. 23: getUTCMilliseconds

12282. -> [timeval this]

12283. We should never exit the loop above.

12284. e.g. a = -4, b = 5 --> -4 - 5 + 1 / 5 --> -8 / 5 --> -1 * a = -5, b = 5 --> -5 - 5 + 1 / 5 --> -9 / 5 --> -1 * a = -6, b = 5 --> -6 - 5 + 1 / 5 --> -10 / 5 --> -2

12285. Parser separator indices.

12286. -> [... this timeval_new timeval_new]

12287. "-1234560"

12288. rule match

12289. deal with negative values

12290. 10: getMonth

12291. * Calendar helpers * * Some helpers are used for getters and can operate on normalized values * which can be represented with 32-bit signed integers. Other helpers are * needed by setters and operate on un-normalized double values, must watch * out for non-finite numbers etc.

12292. **comment:** Debug macro to print all parts and dparts (used manually because of debug level).

label: code-design

12293. Unlike year, the other parts fit into 16 bits so %d format * is portable.

12294. * Date/time parsing helper. * * Parse a datetime string into a time value. We must first try to parse * the input according to the standard format in E5.1 Section 15.9.1.15. * If that fails, we can try to parse using custom parsing, which can * either be platform neutral (custom code) or platform specific (using * existing platform API calls). * * Note in particular that we must parse whatever toString(), toUTCString(), * and toISOString() can produce; see E5.1 Section 15.9.4.2. * * Returns 1 to allow tail calling. * * There is much room for improvement here with respect to supporting * alternative datetime formats. For instance, V8 parses '2012-01-01' as * UTC and '2012/01/01' as local time.

12295. unpack args

12296. Set timeval to 'this' from dparts, push the new time value onto the * value stack and return 1 (caller can then tail call us). Expects * the value stack to contain 'this' on the stack top.

12297. Given a day number, determine year and day-within-year.

12298. Parser separator masks.

12299. Equivalent year mapping, used to avoid DST trouble when platform * may fail to provide reasonable DST answers for dates outside the * ordinary range (e.g. 1970-2038). An equivalent year has the same * leap-year-ness as the original year and begins on the same weekday * (Jan 1). * * The year 2038 is avoided because there seem to be problems with it * on some platforms. The year 1970 is also avoided as there were * practical problems with it; an equivalent year is used for it too, * which breaks some DST computations for 1970 right now, see e.g. * test-bi-date-tzoffset-brute-fi.js.

12300. * Helper to format a time value into caller buffer, used by logging. * 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.

12301. * Constructor calls

12302. Helper for component setter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), modify one or more components as specified, recompute * the time value, set it as the internal value. Finally, push the * new time value as a return value to the value stack and return 1 * (caller can then tail call us).

12303. ignore millisecond fractions after 3

12304. try locale specific formatter; if it refuses to format the * string, fall back to an ISO 8601 formatted value in local * time.

12305. 15: getUTCDate

12306. This assert depends on the input parts representing time inside * the Ecmascript range.

12307. **comment:** XXX: getter with class check, useful in built-ins

label: code-design

12308. get as double to handle huge numbers correctly

12309. these computations are guaranteed to be exact for the valid * E5 time value range, assuming milliseconds without fractions.

12310. if no NaN handling flag, may still be NaN here, but not Inf

12311. 0: toString

12312. 26: setSeconds

12313. Split time value into parts. The time value is assumed to be an internal * one, i.e. finite, no fractions. Possible local time adjustment has already * been applied when reading the time value.

12314. Because 'day since epoch' can be negative and is used to compute weekday * using a modulo operation, add this multiple of 7 to avoid negative values * when year is below 1970 epoch. Ecmascript time values are restricted to * +/- 100 million days from epoch, so this adder fits nicely into 32 bits. * Round to a multiple of 7 (= floor(100000000 / 7) * 7) and add margin.

12315. Contract, either: * - Push string to value stack and return 1 * - Don't push anything and return 0

12316. 19: getUTCMilliseconds

12317. 14: getDay

12318. **comment:** Note: this is very tricky; we must never 'overshoot' the * correction downwards.

label: code-design

12319. * Forward declarations

12320. conservative

12321. Note: in integer arithmetic, (x / 4) is same as floor(x / 4) for non-negative * values, but is incorrect for negative ones.

12322. Compute day number of the first day of a given year.

12323. 29: setUTCMilliseconds

12324. Behavior for nargs < 2 is implementation dependent: currently we'll * set a NaN time value (matching V8 behavior) in this case.

12325. "+11:22:0"

12326. Note: toJSON() is a generic function which works even if 'this' * is not a Date. The sole argument is ignored.

12327. **comment:** Two value cycle, see e.g. test-bi-date-tzoffset-basic-fi.js. * In these cases, favor a higher tzoffset to get a consistent * result which is independent of iteration count. Not sure if * this is a generically correct solution.

label: code-design

12328. complete the millisecond field

12329. * Getters. * * Implementing getters is quite easy. The internal time value is either * NaN, or represents milliseconds (without fractions) from Jan 1, 1970. * The internal time value can be converted to integer parts, and each * part will be normalized and will fit into a 32-bit signed integer. * * A shared native helper to provide all getters. Magic value contains * a set of flags and also packs the date component index argument. The * helper provides: * * getFullYear() * getUTCFullYear() * getMonth() * getUTCMonth() * getDate() * getUTCDate() * getDay() * getUTCDay() * getHours() * getUTCHours() * getMinutes() * getUTCMilliseconds() * getSeconds() * getUTCSeconds() * getMilliseconds() * getUTCMinutes() * getYear() * * Notes: * * - Date.prototype.getDate(): 'date' means day-of-month, and is * zero-based in internal calculations but public API expects it to * be one-based. * * - Date.prototype.getTime() and Date.prototype.valueOf() have identical * behavior. They have separate function objects, but share the same C * function (duk.bi.date.prototype.valueOf).

12330. day number for Jan 1 since epoch

12331. -> [... this timeval]

12332. Parser part indices.

12333. no fractions in internal time

12334. no need to mask idx portion

12335. does not fit into 16 bits
12336. This is based on Rhino EquivalentYear() algorithm: *
 <https://github.com/mozilla/rhino/blob/f99cc11d616f0cdda2c42bde72b3484df6182947/src/org.mozilla/javascript/NativeDate.java>
12337. 0
12338. Iteration solution
12339. accept string if next char is NUL (otherwise reject)
12340. SCANBUILD: scan-build complains here about assigned value * being garbage or undefined. This is correct but operating * on undefined values has no ill effect and is ignored by the * caller in the case where this happens.
12341. 31: setUTCHours
12342. This native function is also used for Date.prototype.getTime() * as their behavior is identical.
12343. Note1
12344. Given a (year, month, day-within-month) triple, compute day number. * The input triple is un-normalized and may contain non-finite values.
12345. accept string
12346. No platform-specific parsing, this is not an error.
12347. Apply ToNumber() to specified index; if ToInteger(val) in [0,99], add * 1900 and replace value at idx_val.
12348. 4: toLocaleDateString
12349. This is based on V8 EquivalentYear() algorithm (see src/genequivyear.py): * <http://code.google.com/p/v8/source/browse/trunk/src/date.h#146>
12350. Use double parts, they tolerate unnormalized time. * * Note: DUK_DATE_IDX_WEEKDAY is initialized with a bogus value (DUK_PL_TZHOUR) * on purpose. It won't be actually used by duk_bi_date_get_timeval_from_dparts(), * but will make the value initialized just in case, and avoid any * potential for Valgrind issues.
12351. See comments below on MakeTime why these are volatile.
12352. Contract, either: * - Push value on stack and return 1 * - Don't push anything on stack and return 0
12353. E5 Sections 15.9.3.1, B.2.4, B.2.5
12354. tzoffset
12355. The timevalue must be in valid Ecmascript range, but since a local * time offset can be applied, we need to allow a +/- 24h leeway to * the value. In other words, although the UTC time is within the * Ecmascript range, the local part values can be just outside of it.
12356. seconds
12357. 18: getMinutes
12358. Buffer sizes for some UNIX calls. Larger than strictly necessary * to avoid Valgrind errors.
12359. DUK_USE_DATE_TZO_GMTIME
12360. one-based -> zero-based
12361. DUK_USE_DATE_PRS_STRPTIME
12362. **comment:** XXX: return something more useful, so that caller can throw?
 label: code-design
12363. For NaN/inf, the return value doesn't matter.
12364. **comment:** If the platform doesn't support the entire Ecmascript range, we need * to return 0 so that the caller can fall back to the default formatter. * * For now, assume that if time_t is 8 bytes or more, the whole Ecmascript * range is supported. For smaller time_t values (4 bytes in practice), * assumes that the signed 32-bit range is supported. * * XXX: detect this more correctly per platform. The size of time_t is * probably not an accurate guarantee of strftime() supporting or not * supporting a large time range (the full Ecmascript range).
 label: code-design
12365. Compute final offset in seconds, positive if local time ahead of * UTC (returned value is UTC-to-local offset). * * difftime() returns a double, so coercion to int generates quite * a lot of code. Direct subtraction is not portable, however. * XXX: allow direct subtraction on known platforms.
12366. no fractions
12367. The necessary #includes are in place in duk_config.h.
12368. already one-based
12369. DUK_USE_DATE_PRS_GETDATE
12370. DUK_USE_DATE_NOW_GETTIMEOFDAY
12371. be paranoid for 32-bit time values (even avoiding negative ones)
12372. unsigned 31-bit range
12373. Get local time offset (in seconds) for a certain (UTC) instant 'd'.
12374. negative: dst info not available
12375. Not a very good provider: only full seconds are available.
12376. copy to buffer with spare to avoid Valgrind gripes from strftime
12377. valgrind whine without this
12378. This check used to be for ($t < 0$) but on some platforms * time_t is unsigned and apparently the proper way to detect * an mktime() error return is the cast above.
 See e.g.: * <http://pubs.opengroup.org/onlinepubs/009695299/functions/mktime.html>
12379. * Unix-like Date providers * * Generally useful Unix / POSIX / ANSI Date providers.
12380. DUK_USE_DATE_NOW_TIME
12381. Get current Ecmascript time (= UNIX/Posix time, but in milliseconds).
12382. UTC
12383. * This is a bit tricky to implement portably. The result depends * on the timestamp (specifically, DST depends on the timestamp). * If e.g. UNIX APIs are used, they'll have portability issues with * very small and very large years. * * Current approach: * * - Stay within portable UNIX limits by using equivalent year mapping. * Avoid year 1970 and 2038 as some conversions start to fail, at * least on some platforms. Avoiding 1970 means that there are * currently DST discrepancies for 1970. * * - Create a UTC and local time breakdowns from 't'. Then create * a time_t using gmtime() and localtime() and compute the time * difference between the two. * * Equivalent year mapping (E5 Section 15.9.1.8): * * If the host environment provides functionality for determining * daylight saving time, the implementation of ECMAScript is free * to map the year in question to an equivalent year (same * leap-year-ness and same starting week day for the year) for which * the host environment provides daylight saving time information. * The only restriction is that all equivalent years should produce * the same result. * * This approach is quite reasonable but not entirely correct, e.g. * the specification also states (E5 Section 15.9.1.8): * * The implementation of ECMAScript should not try to determine * whether the exact time was subject to daylight saving time, but * just whether daylight saving time would have been in effect if * the _current daylight saving time algorithm_ had been used at the * time. This avoids complications such as taking into account the * years that the locale observed daylight saving time year round. * * Since we rely on the platform APIs for conversions between local * time and UTC, we can't guarantee the above. Rather, if the platform * has historical DST rules they will be applied. This seems to be the * general preferred direction in Ecmascript standardization (or at least * implementations) anyway, and even the equivalent year mapping should * be disabled if the platform is known to handle DST properly for the * full Ecmascript range. * * The following has useful discussion and links: * * https://bugzilla.mozilla.org/show_bug.cgi?id=351066

12384. local
12385. DUK_USE_DATE_FMT_STRFTIME
12386. tm_isdst is both an input and an output to mktime(), use 0 to * avoid DST handling in mktime(): * - <https://github.com/svaarala/duktape/issues/406> * - <http://stackoverflow.com/questions/8558919/mktime-and-tm-isdst>
12387. flags
12388. For this to work, DATEMSK must be set, so this is not very * convenient for an embeddable interpreter.
12389. If not within Ecmascript range, some integer time calculations * won't work correctly (and some asserts will fail), so bail out * if so. This fixes test-bug-date-insane-setyear.js. There is * a +/- 24h leeway in this range check to avoid a test262 corner * case documented in test-bug-date-timeval-edges.js.
12390. DUK_USE_DATE_NOW_WINDOWS
12391. input 'd' in Windows UTC, 100ns units
12392. Difference is in 100ns units, convert to milliseconds w/o fractions
12393. millisec -> 100ns units since jan 1, 1970
12394. XXX: handling of timestamps outside Windows supported range. * How does Windows deal with dates before 1600? Does windows * support all Ecmascript years (like -200000 and +200000)? * Should equivalent year mapping be used here too? If so, use * a shared helper (currently integrated into timeval-to-parts).

12395. The necessary #includes are in place in duk_config.h.
12396. * Windows Date providers ** Platform specific links: * * - [http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473(v=vs.85).aspx)
12397. Use the approach described in "Remarks" of FileTimeToLocalFileTime: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx)
12398. Suggested step-by-step method from documentation of RtTimeToSecondsSince1970: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928(v=vs.85).aspx)

12399. Shared Windows helpers.
12400. Positive if local time ahead of UTC.
12401. **comment:** not sure whether or not needed; Thursday
 label: requirement
12402. seconds
12403. defined(DUK_USE_DATE_NOW_WINDOWS) || defined(DUK_USE_DATE_TZO_WINDOWS)
12404. DUK_USE_DATE_TZO_WINDOWS
12405. address
12406. -> [val arr]
12407. XXX: Not sure what the best return value would be in the API. * Return a boolean for now. Note that rc == 0 is success (true).
12408. type tag (public)
12409. return the argument object
12410. result array
12411. idx_value
12412. idx_space
12413. **comment:** XXX: primitive to make array from valstack slice
 label: code-design
12414. XXX: version specific array format instead?
12415. **comment:** * Duktape built-ins ** Size optimization note: it might seem that vararg multipurpose functions * like fin(), enc(), and dec() are not very size optimal, but using a single * user-visible Ecmascript function saves a lot of run-time footprint; each * Function instance takes >100 bytes. Using a shared native helper and a * 'magic' value won't save much if there are multiple Function instances * anyway.
 label: code-design
12416. refcount
12417. **comment:** When alloc_size == 0 the second allocation may not * actually exist.
 label: code-design
12418. because arg count is 1
12419. heapdhr size and additional allocation size, followed by * type specific stuff (with varying value count)
12420. **comment:** Providing access to e.g. act->lex_env would be dangerous: these * internal structures must never be accessible to the application. * Duktape relies on them having consistent data, and this consistency * is only asserted for, not checked for.
 label: code-design
12421. Raw helper to extract internal information / statistics about a value. * The return values are version specific and must not expose anything * that would lead to security issues (e.g. exposing compiled function * 'data' buffer might be an issue). Currently only counts and sizes and * such are given so there should not be a security impact.
12422. Note: e_next indicates the number of gc-reachable entries * in the entry part, and also indicates the index where the * next new property would be inserted. It does *not* indicate * the number of non-NULL keys present in the object. That * value could be counted separately but requires a pass through * the key list.
12423. set values into ret array
12424. **comment:** Set: currently a finalizer is disabled by setting it to * undefined; this does not remove the property at the moment. * The value could be type checked to be either a function * or something else; if something else, the property could * be deleted.
 label: code-design
12425. [level obj func pc line]
12426. idx_replacer
12427. Relevant PC is just before current one because PC is * post-incremented. This should match what error augment * code does.
12428. * Compact an object
12429. -1 = top callstack entry, callstack[callstack_top - 1] * -callstack_top = bottom callstack entry, callstack[0]
12430. flags
12431. **comment:** Vararg function: must be careful to check/require arguments. * The JSON helpers accept invalid indices and treat them like * non-existent optional parameters.
 label: code-design
12432. internal type tag
12433. Get.
12434. [... this]
12435. fixed arg count: value
12436. **comment:** * Traceback handling ** The unified helper decodes the traceback and produces various requested * outputs. It should be optimized for size, and may leave garbage on stack, * only the topmost return value matters. For instance, traceback separator * and decoded strings are pushed even when looking for filename only. * * NOTE: although _Tracedata is an internal property, user code can currently * write to the array (or replace it with something other than an array). * The code below must tolerate arbitrary _Tracedata. It can throw errors * etc, but cannot cause a segfault or memory unsafe behavior.
 label: code-design
12437. noblame_fileline
12438. **comment:** If message is undefined, the own property 'message' is not set at * all to save property space. An empty message is inherited anyway.
 label: code-design
12439. When looking for .fileName/.lineNumber, blame first * function which has a .fileName.
12440. [... this tracedata sep this str1 ... strN]
12441. may be NULL
12442. constants arbitrary, chosen for small loads
12443. * Ecmascript/native function call or lightfunc call
12444. not enumerable
12445. The 'this' after 'sep' will get ToString() coerced by * duk_join() automatically. We don't want to do that * coercion when providing .fileName or .lineNumber (GH-254).
12446. **comment:** XXX: Are steps 6 and 7 in E5 Section 15.11.4.4 duplicated by * accident or are they actually needed? The first ToString() * could conceivably return 'undefined'.
 label: code-design
12447. **comment:** XXX: skip null filename?
 label: code-design
12448. NULL for lightfunc
12449. **comment:** XXX: remove this native function and map 'stack' accessor * to the toString() implementation directly.
 label: code-design
12450. unknown, ignore
12451. [... v1 v2 name filename]
12452. traceback depth ensures fits into 16 bits
12453. [... obj key value]
12454. * Traceback handling when tracebacks disabled. * * The fileName / lineNumber stubs are now necessary because built-in * data will include the accessor properties in Error.prototype. If those * are removed for builds without tracebacks, these can also be removed. * 'stack' should still be present and produce a

`ToString()` equivalent: * this is useful for user code which prints a stacktrace and expects to * see something useful. A normal stacktrace also begins with a `ToString()` * of the error so this makes sense.

12455. fixed arg count

12456. [... this name message]

12457. name is empty -> return message

12458. **comment:** XXX: Output type could be encoded into native function 'magic' value to * save space. For setters the stridx could be encoded into 'magic'.
label: code-design

12459. Augment the error if called as a normal function. `_FILE_` and `_LINE_` * are not desirable in this case.

12460. [... v1(filename) v2(line+flags)]

12461. [... v1 v2 name filename str] -> [... str v2 name filename]

12462. Behavior for constructor and non-constructor call is * the same except for augmenting the created error. When * called as a constructor, the caller (`duk_new()`) will handle * augmentation; when called as normal function, we need to do * it here.

12463. -> [... str]

12464. * `_FILE_` / `_LINE_` entry, here 'pc' is line number directly. * Sometimes `_FILE_` / `_LINE_` is reported as the source for * the error (`fileName`, `lineNumber`), sometimes not.

12465. Possibly truncated; there is no explicit truncation * marker so this is the best we can do.

12466. [... v1 v2 str] -> [... str v2]

12467. `DUK_USE_TRACEBACKS`

12468. traceback depth fits into 16 bits

12469. [message error message]

12470. same for both error and each subclass like `TypeError`

12471. Attempt to write 'stack', 'fileName', 'lineNumber' works as if * user code called `Object.defineProperty()` to create an overriding * own property. This allows user code to overwrite `.fileName` etc * intuitively as e.g. "`err.fileName = 'dummy'`" as one might expect. * See <https://github.com/svaarala/duktape/issues/387>.

12472. [... v1(func) v2(pc+flags)]

12473. ... name ': ' message

12474. **comment:** XXX: optimize with more direct internal access
label: code-design

12475. **comment:** XXX: Could be improved by coercing to a readable `duk_tval` (especially string escaping)
label: code-design

12476. [... this name]

12477. message is empty -> return name

12478. [... this tracedata sep this]

12479. * Error built-ins

12480. **comment:** XXX: Change 'anon' handling here too, to use empty string for anonymous functions?
label: code-design

12481. count, not including sep

12482. stack type fits into 16 bits

12483. When looking for `.fileName/.lineNumber`, blame compilation * or C call site unless flagged not to do so.

12484. Current tracedata contains 2 entries per callstack entry.

12485. vararg function, careful arg handling (e.g. `thisArg` may not be present)

12486. create bound function object

12487. not a vararg function

12488. these non-standard properties are copied for convenience

12489. step 15.a

12490. only outer `lex_env` matters, as functions always get a new * variable declaration environment.

12491. [body formals source template closure]

12492. `add_auto_proto`

12493. [body formals source]

12494. **comment:** Function name: missing/undefined is mapped to empty string, * otherwise coerce to string.
label: code-design

12495. XXX: the implementation now assumes "chained" bound functions, * whereas "collapsed" bound functions (where there is ever only * one bound function which directly points to a non-bound, final * function) would require a "collapsing" implementation which * merges argument lists etc here.

12496. [body formals], formals is comma separated list that needs to be parsed

12497. [arg1 ... argN-1 body] -> [body arg1 ... argN-1]

12498. attrs in E5 Section 15.3.5.1

12499. **comment:** * E5 Section 15.3.4.2 places few requirements on the output of * this function: * * - The result is an implementation dependent representation * of the function; in particular * * - The result must follow the syntax of a `FunctionDeclaration`. * In particular, the function must have a name (even in the * case of an anonymous function or a function with an empty * name). * * - Note in particular that the output does NOT need to compile * into anything useful.
label: code-design

12500. bound function 'length' property is interesting

12501. caller and arguments must use the same thrower, `[[ThrowTypeError]]`

12502. [func thisArg arg1 ... argN]

12503. `thisArg`

12504. vararg function, `thisArg` needs special handling

12505. **comment:** XXX: 'copy properties' API call?

label: code-design

12506. **comment:** XXX: currently no handling for non-allowed identifier characters, * e.g. a '{' in the function name.
label: code-design

12507. Step 1 is not necessary because `duk_call_method()` will take * care of it.

12508. * Function built-ins

12509. **comment:** XXX: cover this with the generic >1 case?
label: code-design

12510. **comment:** XXX: make this an internal helper
label: code-design

12511. = 1 + arg count

12512. [thisArg arg1 ... argN func] (`thisArg+args == nargs total`)

12513. [thisArg arg1 ... argN]

12514. `lightfunc`

12515. XXX: this placeholder is not always correct, but use for now. * It will fail in corner cases; see `test-dev-func-cons-args.js`.

12516. [thisArg arg1 ... argN func `boundFunc` argArray]

12517. **comment:** The 'strict' flag is copied to get the special `[[Get]]` of E5.1 * Section 15.3.5.4 to apply when a 'caller' value is a strict bound * function. Not sure if this is correct, because the specification * is a bit ambiguous on this point but it would make sense.
label: code-design

12518. [thisArg arg1 ... argN func `boundFunc`]

12519. Lightfuncs are always strict.

12520. [body formals source template]

12521. ignore arguments, return undefined (E5 Section 15.3.4)

12522. **comment:** XXX: ignored now

label: code-design

12523. func

12524. For lightfuncs, simply read the virtual property.

12525. **comment:** XXX: faster internal way to get this

label: code-design

12526. 'func' in the algorithm

12527. [func thisArg argArray]

12528. ToUint32() coercion required

12529. **comment:** XXX: copy from caller?

label: code-design

12530. normal and constructor calls have identical semantics

12531. Indicate function type in the function body using a dummy * directive.

12532. strictness is not inherited, intentional

12533. Duktape object

12534. Finish the wrapped module source. Force module.filename as the * function .fileName so it gets set for functions defined within a * module. This also ensures loggers created within the module get * the module ID (or overridden filename) as their default logger name. * (Note capitalization: .filename matches Node.js while .fileName is * used elsewhere in Duktape.)

12535. ignore non-strings

12536. Duktape.modSearch(resolved_id, fresh_require, exports, module).

12537. Module id requested

12538. DUK_USE_COMMONJS_MODULES

12539. Module loading failed. Node.js will forget the module * registration so that another require() will try to load * the module again. Mimic that behavior.

12540. Module object containing module.exports, etc

12541. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func]

12542. 2 when called by debugger

12543. return module.exports

12544. eat dup slashes

12545. at most sizeof(buf) - 1

12546. requested identifier

12547. E5.1 Section 15.1.3.2: empty

12548. **comment:** NOTE: we try to minimize code size by avoiding unnecessary pops, * so the stack looks a bit cluttered in this function. DUK_ASSERT_TOP() * assertions are used to ensure stack configuration is correct at each * step.

label: code-design

12549. spaces (nargs - 1) + newline

12550. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded Duktape.modLoaded[id]]

12551. [... Duktape.modSearch resolved_id last_comp fresh_require exports module]

12552. * Parsing of ints and floats

12553. Module has now evaluated to a wrapped module function. Force its * .name to match module.name (defaults to last component of resolved * ID) so that it is shown in stack traces too. Note that we must not * introduce an actual name binding into the function scope (which is * usually the case with a named function) because it would affect the * scope seen by the module and shadow accesses to globals of the same name. * This is now done by compiling the function as anonymous and then forcing * its .name without setting a "has name binding" flag.

12554. Characters outside BMP cannot be escape()'d. We could * encode them as surrogate pairs (for codepoints inside * valid UTF-8 range, but not extended UTF-8). Because * escape() and unescape() are legacy functions, we don't.

12555. Duktape.modLoaded[] module cache

12556. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module]

12557. 1110 xxxx; 3 bytes

12558. [source template result]

12559. if direct eval, calling activation must exist

12560. If argument count is 1 and first argument is a buffer, write the buffer * as raw data into the file without a newline; this allows exact control * over stdout/stderr without an additional entrypoint (useful for now). * * Otherwise current print/alert semantics are to ToString() coerce * arguments, join them with a single space, and append a newline.

12561. last component

12562. Term was '.', backtrack resolved name by one component. * q[-1] = previous slash (or beyond start of buffer) * q[-2] = last char of previous component (or beyond start of buffer)

12563. Duktape.modLoaded

12564. p overshoots

12565. **comment:** XXX: copy from caller?

label: code-design

12566. output is never longer than input during resolution

12567. DUK_USE_PREFER_SIZE

12568. fall through

12569. 0x10-0x1f

12570. Output #2: last component name

12571. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined]

12572. print provider

12573. Eval code doesn't need an automatic .prototype object.

12574. **comment:** * A few notes on the algorithm: * * - Terms are not allowed to begin with a period unless the term * is either '.' or '..'. This simplifies implementation (and * is within CommonJS modules specification). * * - There are few output bound checks here. This is on purpose: * the resolution input is length checked and the output is never * longer than the input. The resolved output is written directly * over the input because it's never longer than the input at any * point in the algorithm. * * - Non-ASCII characters are processed as individual bytes and * need no special treatment. However, U+0000 terminates the * algorithm; this is not an issue because U+0000 is not a * desirable term character anyway.

label: code-design

12575. Term was '!' and is eaten entirely (including dup slashes).

12576. E5.1 Section B.2.2, step 7.

12577. Backtrack to previous slash or start of buffer.

12578. * Number checkers

12579. Decode UTF-8 codepoint from a sequence of hex escapes. The * first byte of the sequence has been decoded to 't'. * * Note that UTF-8 validation must be strict according to the * specification: E5.1 Section 15.1.3, decode algorithm step * 4.d.vii.8. URIError from non-shortest encodings is also * specifically noted in the spec.

12580. Resolved, normalized absolute module ID

12581. Here 'p' always points to the start of a term. * * We can also unconditionally reset q_last here: if this is * the last (non-empty) term q_last will have the right value * on loop exit.

12582. add_auto_proto

12583. 0x20-0x2f

12584. module.filename, default to resolved ID if * not present.

12585. E5.1 Section 15.1.3.1: uriReserved + '#'

12586. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func exports fresh_require exports module]

12587. New require() function for module, updated resolution base
12588. this may have side effects, so re-lookup act
12589. a fresh require() with require.id = resolved target module id
12590. Delete entry in Duktape.modLoaded[] and rethrow.
12591. 110x xxxx; 2 bytes
12592. Module table: * - module.exports: initial exports table (may be replaced by user) * - module.id is non-writable and non-configurable, as the CommonJS * spec suggests this if possible * - module.filename: not set, defaults to resolved ID if not explicitly * set by modSearch() (note capitalization, not .fileName, matches Node.js) * - module.name: not set, defaults to last component of resolved ID if * not explicitly set by modSearch()
12593. 0x30-0x3f
12594. initial size guess
12595. decode '%xx' to "%xx" if decoded char in reserved set
12596. The base ID of the current require() function, resolution base
12597. [requested_id require require.id resolved_id last_comp]
12598. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined fresh_require exports module result(ignored)]
12599. Current require() function
12600. require.id of current module
12601. mod_id may be NULL
12602. **comment:** Higher footprint, less churn.
label: code-design
12603. DUK_USE_SECTION_B
12604. compact the exports table
12605. Needs lookahead
12606. 'undefined', artifact of lookup
12607. 1111 0xxx; 4 bytes
12608. **comment:** NOTE: level is used only by the debugger and should never be present * for an Ecmascript eval().
label: code-design
12609. not reachable
12610. 0x50-0x5f
12611. If user callback did not return source code, module loading * is finished (user callback initialized exports table directly).
12612. Last component name in resolved path
12613. -> [... source]
12614. module table remains registered to modLoaded, minimize its size
12615. Although we can allow non-BMP characters (they'll decode * back into surrogate pairs), we don't allow extended UTF-8 * characters; they would encode to URIs which won't decode * back because of strict UTF-8 checks in URI decoding. * (However, we could just as well allow them here.)
12616. this is just beneath bottom
12617. p points to digit part ('%xy', p points to 'x')
12618. * Set up the resolution input which is the requested ID directly * (if absolute or no current module path) or with current module * ID prepended (if relative and current module path exists). * * Suppose current module is 'foo/bar' and relative path is './quux'. * The 'bar' component must be replaced so the initial input here is * 'foo/bar/../quux'.
12619. by default, use caller's environment
12620. eat (first) input slash
12621. module.id = resolved_id
12622. indirect eval
12623. %6x%0xx...%xx', p points to char after first '%'
12624. [source template closure this]
12625. module
12626. E5.1 Section 15.1.3.4: uriUnescaped
12627. invalidated
12628. E5.1 Section 15.1.3.3: uriReserved + uriUnescaped + '#'
12629. this function
12630. exports (this binding)
12631. * Resolve module identifier into canonical absolute form.
12632. continuation byte
12633. * CommonJS require() and modules support
12634. Copy term name until end or '/'.
12635. Non-BMP characters within valid UTF-8 range: encode as is. * They'll decode back into surrogate pairs if the escaped * output is decoded.
12636. delete Duktape.modLoaded[resolved_id]
12637. * Module not loaded (and loading not started previously). * * Create a new require() function with 'id' set to resolved ID * of module being loaded. Also create 'exports' and 'module' * tables but don't register exports to the loaded table yet. * We don't want to do that unless the user module search callbacks * succeeds in finding the module.
12638. bytes left
12639. [source template closure]
12640. * Encoding/decoding helpers
12641. Have a calling activation, check for direct eval (otherwise * assume indirect eval).
12642. **comment:** Compact but lots of churn.
label: code-design
12643. **comment:** The E5.1 algorithm checks whether or not a decoded codepoint * is below 0x80 and perhaps may be in the "reserved" set. * This seems pointless because the single byte UTF-8 case is * handled separately, and non-shortest encodings are rejected. * So, 'cp' cannot be below 0x80 here, and thus cannot be in * the reserved set.
label: code-design
12644. Potentially truncated, NUL not guaranteed in any case. * The (int_rc < 0) case should not occur in practice.
12645. 0x40-0x4f
12646. **comment:** Fresh require: require.id is left configurable (but not writable) * so that is not easy to accidentally tweak it, but it can still be * done with Object.defineProperty(). * * XXX: require.id could also be just made non-configurable, as there * is no practical reason to touch it.
label: code-design
12647. * callstack_top - 1 --> this function * callstack_top - 2 --> caller (may not exist) * * If called directly from C, callstack_top might be 1. If calling * activation doesn't exist, call must be indirect.
12648. This was the last term, and q_last was * updated to match this term at loop top.
12649. User callback did not return source code, so module loading * is finished: just update modLoaded with final module.exports * and we're done.
12650. e.g. require('/foo'), empty terms not allowed
12651. at least this function exists
12652. 0x70-0x7f
12653. **comment:** XXX: check flags
label: requirement
12654. write on next loop
12655. Default exports table
12656. Duktape.modLoaded[resolved_id] = module

12657. * Call user provided module search function and build the wrapped * module source code (if necessary). The module search function * can be used to implement pure Ecmascript, pure C, and mixed * Ecmascript/C modules. ** The module search function can operate on the exports table directly * (e.g. DLL code can register values to it). It can also return a * string which is interpreted as module source code (if a non-string * is returned the module is assumed to be a pure C one). If a module * cannot be found, an error must be thrown by the user callback. ** Because Duktape.modLoaded[] already contains the module being * loaded, circular references for C modules should also work * (although expected to be quite rare).

12658. stash to bottom of value stack to keep new_env reachable for duration of eval

12659. XXX: Could add fast path (for each transform callback) with direct byte * lookups (no shifting) and no explicit check for x < 0x80 before table * lookup.

12660. extended utf-8 not allowed for URIs

12661. This is guaranteed by debugger code.

12662. Maximum write size: XUTF8 path writes max DUK_UNICODE_MAX_XUTF8_LENGTH, * percent escape path writes max two times CESU-8 encoded BMP length.

12663. E5 Section 10.4.2

12664. Only direct eval inherits strictness from calling code * (E5.1 Section 10.1.1).

12665. Macros for creating and checking bitmasks for character encoding. * Bit number is a bit counterintuitive, but minimizes code size.

12666. exports

12667. return arg as-is

12668. * Global object built-ins

12669. **comment:** * Eval * * Eval needs to handle both a "direct eval" and an "indirect eval". * Direct eval handling needs access to the caller's activation so that its * lexical environment can be accessed. A direct eval is only possible from * Ecmascript code; an indirect eval call is possible also from C code. * When an indirect eval call is made from C code, there may not be a * calling activation at all which needs careful handling.
label: code-design

12670. module.name for .name, default to last component if * not present.

12671. rethrow original error

12672. module.exports = exports

12673. 0x60-0x6f

12674. UTF-8 encoded bytes escaped as %xx%xx%xx... -> 3 * nbytes. * Codepoint range is restricted so this is a slightly too large * but doesn't matter.

12675. Output #1: resolved absolute name

12676. [source]

12677. **comment:** Stack indices for better readability
label: code-design

12678. **comment:** XXX: refactor and share with other code
label: code-design

12679. Specification stripPrefix maps to DUK_S2N_FLAG_ALLOW_AUTO_HEX_INT. ** Don't autodetect octals (from leading zeroes), require user code to * provide an explicit radix 8 for parsing octal. See write-up from Mozilla: * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt#ECMAScript_5_Removes_Octal_Interpretation

12680. * Cached module check. * * If module has been loaded or its loading has already begun without * finishing, return the same cached value ('exports'). The value is * registered when module load starts so that circular references can * be supported to some extent.

12681. Register the module table early to modLoaded[] so that we can * support circular references even in modSearch(). If an error * is thrown, we'll delete the reference.

12682. Duktape.modSearch

12683. module (argument)

12684. fresh require (argument)

12685. resolved id: require(id) must return this same module

12686. caller

12687. avoid dereference after potential callstack realloc

12688. relookup exports from module.exports in case it was changed by modSearch

12689. must be signed

12690. * Resolution loop. At the top of the loop we're expecting a valid * term: '.', '..', or a non-empty identifier not starting with a period.

12691. * URI handling

12692. * Call the wrapped module function. * * Use a protected call so that we can update Duktape.modLoaded[resolved_id] * even if the module throws an error.

12693. utf-8 validation ensures these

12694. 0x00-0x0f

12695. nargs

12696. compiler's responsibility

12697. backtrack to last output slash (dups already eliminated)

12698. [source template]

12699. 'buf' contains the string to write, 'sz_buf' contains the length * (which may be zero).

12700. **comment:** Certain boxed types are required to go through * automatic unboxing. Rely on internal value being * sane (to avoid infinite recursion).
label: code-design

12701. DUK_USE_JX

12702. There is no need to check the first character specially here * (i.e. reject digits): the caller only accepts valid initial * characters and won't call us if the first character is a digit. * This also ensures that the plain string won't be empty.

12703. Note: intentionally signed.

12704. XXX: push_uint_string / push_u32_string

12705. [... buf loop]

12706. idx_value

12707. backtrack (safe)

12708. * Top level wrappers

12709. Loop check.

12710. len: 9

12711. required to keep recursion depth correct

12712. [... buf loop (proplist) (gap) holder ""]

12713. Final result is at stack top.

12714. ToInteger() coercion; NaN -> 0, infinities are clamped to 0 and 10

12715. * Local defines and forward declarations.

12716. already zero

12717. [... val root ""] -> [... val val']

12718. **comment:** * JSON built-ins. * * See doc/json.rst. ** Codepoints are handled as duk_uint_fast32_t to ensure that the full * unsigned 32-bit range is supported. This matters to e.g. JX. * * Input parsing doesn't do an explicit end-of-input check at all. This is * safe: input string data is always NUL-terminated (0x00) and valid JSON * inputs never contain plain NUL characters, so that as long as syntax checks * are correct, we'll never read past the NUL. This approach reduces code size * and improves parsing performance, but it's critical that syntax checks are * indeed correct!
label: code-design

12719. flags

12720. Guaranteed by recursion_limit setup so we don't have to * check twice.

12721. The Quote(value) operation: quote a string. * * Stack policy: [] -> [].

12722. [... obj]

12723. The coercion potentially invokes user .valueOf() and .toString() * but can't result in a function value because [[DefaultValue]] would * reject such a result: test-dev-json-stringify-coercion-1.js.

12724. if gap is empty, behave as if not given at all

12725. When JX/JC not in use, the type mask above will avoid this case if needed.
12726. Select a safe loop count where no output checks are * needed assuming we won't encounter escapes. Input * bound checks are not necessary as a NUL (guaranteed) * will cause a SyntaxError before we read out of bounds.
12727. Shared handler to minimize parser size. Cause will be * hidden, unfortunately, but we'll have an offset which * is often quite enough.
12728. DUK_USE_JSON_EATWHITE_FASTPATH
12729. "" was eaten by caller
12730. DUK_USE_HEX_FASTPATH
12731. parse value
12732. -> [... key val replacer holder key]
12733. as is
12734. **comment:** Buffer values are encoded in (lowercase) hex to make the * binary data readable. Base64 or similar would be more * compact but less readable, and the point of JX/JC * variants is to be as useful to a programmer as possible.
label: code-design
12735. unconditional block
12736. any other printable -> as is
12737. emitted
12738. -> [... holder name val val ToString(i)]
12739. replacer is a mutation risk
12740. skip ''
12741. toJSON() can also be a lightfunc
12742. parse key and value
12743. default attrs ok
12744. Trailing whitespace has been eaten by duk__dec_value(), so if * we're not at end of input here, it's a SyntaxError.
12745. * Parsing implementation. * * JSON lexer is now separate from duk_lexer.c because there are numerous * small differences making it difficult to share the lexer. * * The parser here works with raw bytes directly; this works because all * JSON delimiters are ASCII characters. Invalid xUTF-8 encoded values * inside strings will be passed on without normalization; this is not a * compliance concern because compliant inputs will always be valid * CESU-8 encodings.
12746. 0x7F is special
12747. -> [... key val val']
12748. skip ')'
12749. [... val]
12750. Fast path, decode as is.
12751. A non-Array object should not have an array part in practice. * But since it is supported internally (and perhaps used at some * point), check and abandon if that's the case.
12752. syntax error
12753. -> [... proplist enum_obj key]
12754. avoid attempts to add/remove object keys
12755. XXX: a "first" flag would suffice
12756. avail_bytes += need_bytes
12757. Parse a number, other than NaN or +/- Infinity
12758. 0x00: finish (non-white) * 0x01: continue
12759. Plus sign must be accepted for positive exponents * (e.g. '1.5e+2'). This clause catches NULs.
12760. nothing now
12761. caller guarantees
12762. DUK_USE_JX && DUK_USE_JC
12763. [... buf]
12764. -> [... key val replacer]
12765. serialize the wrapper with empty string key
12766. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor). * * Using duk_put_prop() works incorrectly with '__proto__' * if the own property with that name has been deleted. This * does not happen normally, but a clever reviver can trigger * that, see complex reviver case in: test-bug-json-parse-__proto__.js.
label: code-design
12767. Note that we currently parse -bytes-, not codepoints. * All non-ASCII extended UTF-8 will encode to bytes >= 0x80, * so they'll simply pass through (valid UTF-8 or not).
12768. Encode a double (checked by caller) from stack top. Stack top may be * replaced by serialized string but is not popped (caller does that).
12769. Function values are handled completely above (including * coercion results):
12770. '\Udeadbeef'
12771. enum_index
12772. eat closing bracket
12773. Encode string in small chunks, estimating the maximum expansion so that * there's no need to ensure space while processing the chunk.
12774. Loop check using a hybrid approach: a fixed-size visited[] array * with overflow in a loop check object.
12775. Must decrease recursion depth before returning.
12776. * Create wrapper object and serialize
12777. All other object types.
12778. * Helper tables
12779. Flag handling currently assumes that flags are consistent. This is OK * because the call sites are now strictly controlled.
12780. **comment:** XXX: duplicates should be eliminated here
label: code-design
12781. **comment:** In compatible mode and standard JSON mode, output * something useful for non-BMP characters. This won't * roundtrip but will still be more or less readable and * more useful than an error.
label: code-design
12782. error message doesn't matter, ignored anyway
12783. must not truncate
12784. negative top-relative indices not allowed now
12785. Encode a fastint from duk_tval ptr, no value stack effects.
12786. avoid attempt to compact any objects
12787. Ensure space for 1:1 output plus one escape.
12788. **comment:** XXX: Could just lookup .toJSON() and continue in fast path, * as it would almost never be defined.
label: code-design
12789. DUK_USE_JSON_DECSTRING_FASTPATH
12790. [... number]-> [... string]
12791. **comment:** XXX: share helper from lexer; duk_lexer.c / hexval().
label: code-design
12792. **comment:** We've ensured space for one escaped input; then * bail out and recheck (this makes escape handling * quite slow but it's uncommon).
label: code-design
12793. **comment:** Use recursion_limit to ensure we don't overwrite js_ctx->visiting[] * array so we don't need two counter checks in the fast path. The * slow path has a much larger recursion limit which we'll use if * necessary.
label: code-design

12794. Recursive value reviver, implements the Walk() algorithm. No C recursion * check is done here because the initial parsing step will already ensure * there is a reasonable limit on C recursion depth and hence object depth.

12795. Note: space must cater for both JX and JC.

12796. **comment:** Indent helper. Calling code relies on js_ctx->recursion_depth also being * directly related to indent depth.
label: code-design

12797. temp copy, write back for next loop

12798. [... buf loop (proplist)]

12799. unsigned

12800. but don't allow leading plus

12801. **comment:** XXX: Here a "slice copy" would be useful.
label: code-design

12802. 0x00: finish (not part of number) * 0x01: continue

12803. -> [... val root ""]

12804. [... holder name val enum obj_key val obj_key]

12805. "-Infinity", '-' has been eaten

12806. JSON parsing code is allowed to read [p_start,p_end]: p_end is * valid and points to the string NUL terminator (which is always * guaranteed for duk_hstrings).

12807. Could also rely on native sprintf(), but it will handle * values like NaN, Infinity, -0, exponent notation etc in * a JSON-incompatible way.

12808. original target at entry_top - 1

12809. catches EOF and invalid digits

12810. safe, because matched (NUL causes a break)

12811. not emitted

12812. [... arr val]

12813. maxval

12814. To handle deeper indents efficiently, make use of copies we've * already emitted. In effect we can emit a sequence of 1, 2, 4, * 8, etc copies, and then finish the last run. Byte counters * avoid multiply with gap_len on every loop.

12815. Must prevent finalizers which may have arbitrary side effects.

12816. expensive flag

12817. Accept ASCII strings which conform to identifier requirements * as being emitted without key quotes. Since we only accept ASCII * there's no need for actual decoding: 'p' is intentionally signed * so that bytes >= 0x80 extend to negative values and are rejected * as invalid identifier codepoints.

12818. The stack has a variable shape here, so force it to the * desired one explicitly.

12819. The Str(key, holder) operation. ** Stack policy: [... key] -> [...]

12820. -> [... holder name val new_elem]

12821. NUL term or -1 (EOF), NUL check would suffice

12822. proplist is very rare

12823. DUK_USE_JC

12824. -> [... key val replacer holder key val]

12825. Decode a plain string consisting entirely of identifier characters. * Used to parse plain keys (e.g. "foo: 123").

12826. **comment:** 0x00 ... 0x7f: as is * 0x80: escape generically * 0x81: slow path * 0xa0 ... 0xff: backslash + one char
label: code-design

12827. -> [...]

12828. -> [... key val toJSON val key]

12829. Caller ensures space for at least DUK_JSON_MAX_ESC_LEN.

12830. Zero length string is not accepted without quotes

12831. Getter might have arbitrary side effects, * so bail out.

12832. nret

12833. Negative zero needs special handling in JX/JC because * it would otherwise serialize to '0', not '-0'.

12834. -> [... voidp voidp]

12835. [... holder name val enum obj_key new_elem]

12836. Disabled until fixed, see above.

12837. Error message doesn't matter: the error is ignored anyway.

12838. -> [... key val']

12839. Value would normally be omitted, replace with 'null'.

12840. Value would yield 'undefined', so skip key altogether. * Side effects have already happened.

12841. XXX: array internal?

12842. For JX, expressing the whole unsigned 32-bit range matters.

12843. Caller must ensure 'tv' is indeed a fastint!

12844. safe

12845. will result in undefined

12846. **comment:** Execute the fast path in a protected call. If any error is thrown, * fall back to the slow path. This includes e.g. recursion limit * because the fast path has a smaller recursion limit (and simpler, * limited loop detection).
label: code-design

12847. typed for duk_unicode_decode_xutf8()

12848. **comment:** XXX: Would be nice to share the fast path loop from duk_hex_decode() * and avoid creating a temporary buffer. However, there are some * differences which prevent trivial sharing: ** - Pipe char detection * - EOF detection * - Unknown length of input and output ** The best approach here would be a bufwriter and a reasonably sized * safe inner loop (e.g. 64 output bytes at a time).
label: code-design

12849. Once recursion depth is increased, exit path must decrease * it (though it's OK to abort the fast path).

12850. If object has a .toJSON() property, we can't be certain * that it wouldn't mutate any value arbitrarily, so bail * out of the fast path. ** If an object is a Proxy we also can't avoid side effects * so abandon.

12851. **comment:** Unlike in duk_hex_encode() 'dst' is not necessarily aligned by 2. * For platforms where unaligned accesses are not allowed, shift 'dst' * ahead by 1 byte to get alignment and then DUK_MEMMOVE() the result * in place. The faster encoding loop makes up the difference. * There's always space for one extra byte because a terminator always * follows the hex data and that's been accounted for by the caller.
label: code-design

12852. catches EOF (NUL)

12853. The code below is incorrect if .toString() or .valueOf() have * have been overridden. The correct approach would be to look up * the method(s) and if they resolve to the built-in function we * can safely bypass it and look up the internal value directly. * Unimplemented for now, abort fast path for boxed values.

12854. accept comma, expect new value

12855. Array part may be larger than 'length'; if so, iterate * only up to array 'length'.

12856. -> [... reviver holder name val]

12857. [... holder name val]

12858. We rely on a few object flag / class number relationships here, * assert for them.

12859. Standard JSON omits functions

12860. unreachable

12861. nargs

12862. The JA(value) operation: encode array. ** Stack policy: [array] -> [array].

12863. x == 0x00 (EOF) causes syntax_error

12864. **comment:** XXX: move masks to js_ctx? they don't change during one * fast path invocation.
label: code-design

12865. Note: duk__dec_req_stridx() backtracks one char
12866. -> [... key val toJSON val]
12867. Here the specification requires correct array index enumeration * which is a bit tricky for sparse arrays (it is handled by the * enum setup code). We now enumerate ancestors too, although the * specification is not very clear on whether that is required.
12868. First pass parse is very lenient (e.g. allows '1.2.3') and extracts a * string for strict number parsing.
12869. -> [... key val]
12870. 0x00: slow path * other: as is
12871. value was undefined/unsupported
12872. **comment:** spaces[] must be static to allow initializer with old compilers like BCC
 label: code-design
12873. Ordinary gap, undefined encodes to 'null' in * standard JSON (and no JX/JC support here now).
12874. Conceptually we'd extract the plain underlying buffer * or its slice and then do a type mask check below to * see if we should reject it. Do the mask check here * instead to avoid making a copy of the buffer slice.
12875. * JSON.stringify() fast path ** Otherwise supports full JSON, JX, and JC features, but bails out on any * possible side effect which might change the value being serialized. The * fast path can take advantage of the fact that the value being serialized * is unchanged so that we can walk directly through property tables etc.
12876. **comment:** XXX: helper
 label: code-design
12877. coerce in-place
12878. Number serialization has a significant impact relative to * other fast path code, so careful fast path for fastints.
12879. [... num]
12880. * Entry points
12881. Caller has already eaten the first character '(' which we don't need.
12882. C recursion check.
12883. standard JSON; array gaps
12884. catches EOF (0x00)
12885. **comment:** XXX: refactor into an internal helper, pretty awkward
 label: code-design
12886. EOF (-1) will be cast to an unsigned value first * and then re-cast for the switch. In any case, it * will match the default case (syntax error).
12887. idx_space
12888. Normal object which doesn't get automatically coerced to a * primitive value. Functions are checked for specially. The * primitive value coercions for Number, String, Pointer, and * Boolean can't result in functions so suffices to check here.
12889. -> [... val']
12890. Caller must ensure 'tv' is indeed a double and not a fastint!
12891. Must set 'length' explicitly when using duk_xdef_prop_xxx() to * set the values.
12892. Caller has already eaten the first character ('|') which we don't need.
12893. Shared entry handling for object/array serialization.
12894. **comment:** XXX: There are no format strings in duk_config.h yet, could add * one for formatting duk_int64_t. For now, assumes "%lld" and that * "long long" type exists. Could also rely on C99 directly but that * won't work for older MSVC.
 label: code-design
12895. Previous entry was inside visited[], nothing to do.
12896. slow path decode
12897. We come here for actual aborts (like encountering .toJSON()) * but also for recursion/loop errors. Bufwriter size can be * kept because we'll probably need at least as much as we've * allocated so far.
12898. **comment:** This fast path is pretty marginal in practice. * XXX: candidate for removal.
 label: code-design
12899. [... proplist enum_obj key val]
12900. **comment:** XXX: substring in-place at idx_place?
 label: code-design
12901. * Process replacer/proplist (2nd argument to JSON.stringify)
12902. Gap in array; check for inherited property, * bail out if one exists. This should be enough * to support gappy arrays for all practical code.
12903. handle comma and closing brace
12904. '-0'
12905. * Process space (3rd argument to JSON.stringify)
12906. caller checks
12907. [... holder name val enum obj_key]
12908. XXX: duk_dup_unvalidated(ctx, -2) etc.
12909. accept anything, expect first value (EOF will be * caught by key parsing below).
12910. First character has already been eaten and checked by the caller. * We can scan until a NUL in stridx string because no built-in strings * have internal NULs.
12911. Assume that the native representation never contains a closing * parenthesis.
12912. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor).
 label: code-design
12913. get_value
12914. -> [... res]
12915. Initial '{' has been checked and eaten by caller.
12916. [... key]
12917. **comment:** This seems faster than emitting bytes one at a time and * then potentially rewinding.
 label: code-design
12918. pop enum
12919. DUK_USE_JSON_STRINGIFY_FASTPATH
12920. leave stack unbalanced on purpose
12921. Most input bytes go through here.
12922. -> [... holder name val]
12923. [... buf loop (proplist) (gap)]
12924. disabled for now
12925. extra coercion of strings is OK
12926. ascii fast path: avoid decoding utf-8
12927. We only get here when doing non-standard JSON encoding
12928. idx_replacer
12929. Handle both full and partial slice (as long as covered).
12930. End of input (NUL) goes through slow path and causes SyntaxError.
12931. Assume arrays are dense in the fast path.
12932. handle comma and closing bracket
12933. Maximum expansion per input byte is 6: * - invalid UTF-8 byte causes "\uXXXX" to be emitted (6/1 = 6). * - 2-byte UTF-8 encodes as "\uXXXX" (6/2 = 3). * - 4-byte UTF-8 encodes as "\Uxxxxxxxx" (10/4 = 2.5).
12934. **comment:** XXX: keys is an internal object with all keys to be processed * in its (gapless) array part. Because nobody can touch the keys * object, we could iterate its array part directly (keeping in mind * that it can be reallocated).
 label: code-design

12935. need_bytes may be zero
12936. clamp to 10 chars
12937. A lightfunc might also inherit a .toJSON() so just bail out.
12938. **comment:** XXX: Stack discipline is annoying, could be changed in numconv.
 label: code-design
12939. XXX: A fast path for usual integers would be useful when * fastint support is not enabled.
12940. We could use a switch-case for the class number but it turns out * a small if-else ladder on class masks is better. The if-ladder * should be in order of relevancy.
12941. **comment:** This approach is a bit shorter than a straight * if-else-ladder and also a bit faster.
 label: code-design
12942. **comment:** XXX: for real world code, could just ignore array inheritance * and only look at array own properties.
 label: code-design
12943. This loop intentionally does not ensure characters are valid * ([0-9a-fA-F]) because the hex decode call below will do that.
12944. -> [... key val replacer holder]
12945. Select appropriate escape format automatically, and set 'tmp' to a * value encoding both the escape format character and the nybble count: * * (nybble_count << 16) | (escape_char1) | (escape_char2)
12946. * Fast path: assume no mutation, iterate object property tables * directly; bail out if that assumption doesn't hold.
12947. [... holder key] -> [... holder]
12948. **comment:** Fastint range is signed 48-bit so longest value is $-2^{47} = -140737488355328$ * (16 chars long), longest signed 64-bit value is $-2^{63} = -9223372036854775808$ * (20 chars long). Alloc space for 64-bit range to be safe.
 label: code-design
12949. Same coercion behavior as for Number.
12950. **comment:** For objects JSON.stringify() only looks for own, enumerable * properties which is nice for the fast path here. * * For arrays JSON.stringify() uses [[Get]] so it will actually * inherit properties during serialization! This fast path * supports gappy arrays as long as there's no actual inherited * property (which might be a getter etc). * * Since recursion only happens for objects, we can have both * recursion and loop checks here. We use a simple, depth-limited * loop check in the fast path because the object-based tracking * is very slow (when tested, it accounted for 50% of fast path * execution time for input data with a lot of small objects!).
 label: code-design
12951. DUK_USE_JSON_DECNUMBER_FASTPATH
12952. -> [... voidp true]
12953. If any non-Array value had enumerable virtual own * properties, they should be serialized here. Standard * types don't.
12954. eat closing brace
12955. Catches EOF of JSON input.
12956. number
12957. radix
12958. end-of-input breaks
12959. **comment:** XXX: would be nice to enumerate an object at specified index
 label: code-design
12960. DUK_USE_JSON_QUOTESTRING_FASTPATH
12961. slow path is shared
12962. **comment:** bytes of indent available for copying
 label: code-design
12963. [... target] -> [... target keys]
12964. [... str]
12965. -> [... value]
12966. object or array
12967. DUK_USE_PREFER_SIZE
12968. overflow not possible, buffer limits
12969. XXX: duk_push_uint_string()
12970. DUK_USE_JX || DUK_USE_JC
12971. E5 Section 15.12.3, main algorithm, step 4.b.ii steps 1-4.
12972. Array length is larger than 'asize'. This shouldn't * happen in practice. Bail out just in case.
12973. not undefined
12974. Restore stack top after unbalanced code paths.
12975. **comment:** When ptr == NULL, the format argument is unused.
 label: code-design
12976. * Context init
12977. -> [... val root val]
12978. [... obj key val]
12979. [...]
12980. * Stringify implementation.
12981. NUL also comes here. Comparison order matters, 0x20 * is most common whitespace.
12982. [... ptr]
12983. double quote or backslash
12984. [... arr]
12985. **comment:** XXX: how to figure correct size?
 label: code-design
12986. c recursion check
12987. catches EOF (NUL) and initial comma
12988. The #ifdef clutter here needs to handle the three cases: * (1) JX+JC, (2) JX only, (3) JC only.
12989. eat trailing comma
12990. disabled
12991. The #ifdef clutter here handles the JX/JC enable/disable * combinations properly.
12992. [... key val]
12993. Decode failed.
12994. end switch
12995. Result is undefined.
12996. **comment:** XXX: non-callable .toJSON() doesn't need to cause an abort * but does at the moment, probably not worth fixing.
 label: code-design
12997. [... proplist]
12998. digits
12999. accept anything, expect first value (EOF will be * caught by duk_dec_value() below.
13000. [... key val] -> [...]
13001. minval
13002. -> [... key]
13003. [... obj key]
13004. Here again we parse bytes, and non-ASCII UTF-8 will cause end of * parsing (which is correct except if there are non-shortest encodings). * There is also no need to check explicitly for end of input buffer as * the input is NUL padded and NUL will exit the parsing loop. * * Because no unescaping takes place, we can just scan to the end of the * plain string and intern from the input buffer.

13005. The JO(value) operation: encode object. ** Stack policy: [object] -> [object].
13006. Caller has already eaten the first char so backtrack one byte.
13007. **comment:** bytes of indent still needed
 label: code-design
13008. nop
13009. If XUTF-8 decoding fails, treat the offending byte as a codepoint directly * and go forward one byte. This is of course very lossy, but allows some kind * of output to be produced even for internal strings which don't conform to * XUTF-8. All standard Ecmascript strings are always CESU-8, so this behavior * does not violate the Ecmascript specification. The behavior is applied to * all modes, including Ecmascript standard JSON. Because the current XUTF-8 * decoding is not very strict, this behavior only really affects initial bytes * and truncated codepoints. ** Another alternative would be to scan forwards to start of next codepoint * (or end of input) and emit just one replacement codepoint.
13010. Initial '[' has been checked and eaten by caller.
13011. We already ate 'x', so backup one byte.
13012. Convert buffer to result string.
13013. Steps 8-10 have been merged to avoid a "partial" variable.
13014. Shared exit handling for object/array serialization.
13015. first character has been matched
13016. There is no need to NUL delimit the sscanf() call: trailing garbage is * ignored and there is always a NUL terminator which will force an error * if no error is encountered before it. It's possible that the scan * would scan further than between [js_ctx->p,p[though and we'd advance * by less than the scanned value. ** Because pointers are platform specific, a failure to scan a pointer * results in a null pointer which is a better placeholder than a missing * value or an error.
13017. [arg1 ... argN this loggerLevel loggerName buffer]
13018. the stack is unbalanced here on purpose; we only rely on the * initial two values: [name this].
13019. **comment:** XXX: better multiline
 label: code-design
13020. [arg undefined]
13021. * Logging support
13022. [arg toLogString]
13023. Default function to format objects. Tries to use toLogString() but falls * back to toString(). Any errors are propagated out without catching.
13024. **comment:** Stripping the filename might be a good idea * (""/foo/bar/quux.js" -> logger name "quux"), * but now used verbatim.
 label: code-design
13025. separators: space, space, colon
13026. Do debugger forwarding before raw() because the raw() function * doesn't get the log level right now.
13027. **comment:** * Logger arguments are: * * magic: log level (0-5) * this: logger * stack: plain log args * * We want to minimize memory churn so a two-pass approach * is used: first pass formats arguments and computes final * string length, second pass copies strings either into a * pre-allocated and reused buffer (short messages) or into a * newly allocated fixed buffer. If the backend function plays * nice, it won't coerce the buffer to a string (and thus * intern it).
 label: code-design
13028. [arg result]
13029. time
13030. * Pass 1
13031. duk_pcall_prop() may itself throw an error, but we're content * in catching the obvious errors (like toLogString() throwing an * error).
13032. **comment:** log level could be popped but that's not necessary
 label: code-design
13033. **comment:** XXX: There used to be a shared log buffer here, but it was removed * when dynamic buffer spare was removed. The problem with using * bufwriter is that, without the spare, the buffer gets passed on * as an argument to the raw() call so it'd need to be resized * (reallocated) anyway. If raw() call convention is changed, this * could be made more efficient.
 label: code-design
13034. Keep the error as the result (coercing it might fail below, * but we don't catch that now).
13035. Constructor
13036. obj_index
13037. keep default instance
13038. nop
13039. [arg1 ... argN this loggerLevel loggerName buffer 'raw' buffer]
13040. * Log level check
13041. don't set 'n' at all, inherited value is used as name
13042. call: this(fmt(arg)
13043. [name this]
13044. 3-letter log level strings
13045. [arg1 ... argN this loggerLevel loggerName]
13046. this.raw(buffer)
13047. * Pass 2
13048. level string
13049. **comment:** Default function to write a formatted log line. Writes to stderr, * appending a newline to the log line. ** The argument is a buffer whose visible size contains the log message. * This function should avoid coercing the buffer to a string to avoid * string table traffic.
 label: code-design
13050. Line format: <time> <entryLev> <loggerName>: <msg>
13051. **comment:** Calling as a non-constructor is not meaningful.
 label: code-design
13052. [arg1 ... argN this]
13053. sep (even before first one)
13054. nargs
13055. **comment:** Automatic defaulting of logger name from caller. This would * work poorly with tail calls, but constructor calls are currently * never tail calls, so tail calls are not an issue now.
 label: code-design
13056. **comment:** Call this.raw(msg); look up through the instance allows user to override * the raw() function in the instance or in the prototype for maximum * flexibility.
 label: code-design
13057. **comment:** When formatting an argument to a string, errors may happen from multiple * causes. In general we want to catch obvious errors like a toLogString() * throwing an error, but we don't currently try to catch every possible * error. In particular, internal errors (like out of memory or stack) are * not caught. Also, we expect Error.toString() to not throw an error.
 label: code-design
13058. **comment:** XXX: sanitize to printable (and maybe ASCII)
 label: code-design
13059. loggerName
13060. **comment:** Log frontend shared helper, magic value indicates log level. Provides * frontend functions: trace(), debug(), info(), warn(), error(), fatal(). * This needs to have small footprint, reasonable performance, minimal * memory churn, etc.
 label: code-design
13061. [arg1 ... argN this loggerLevel loggerName 'fmt' arg]
13062. A stubbed built-in is useful for e.g. compilation torture testing with BCC.
13063. fmin() with args -0 and +0 is not guaranteed to return * -0 as Ecmascript requires.

13064. Not odd, or y == -Infinity
13065. fmax() with args -0 and +0 is not guaranteed to return * +0 as Ecmascript requires.
13066. Wrappers for calling standard math library methods. These may be required * on platforms where one or more of the math built-ins are defined as macros * or inline functions and are thus not suitable to be used as function pointers.
13067. Numbers half-way between integers must be rounded towards +Infinity, * e.g. -3.5 must be rounded to -3 (not -4). When rounded to zero, zero * sign must be set appropriately. E5.1 Section 15.8.2.15. ** Note that ANSI C round() is "round to nearest integer, away from zero", * which is incorrect for negative values. Here we make do with floor().
13068. DUK_USE_MATH_BUILTIN
13069. See test-bug-netbsd-math-pow.js: NetBSD 6.0 on x86 (at least) does not * correctly handle some cases where x=+/-0. Specific fixes to these * here.
13070. * Note: fmax() does not match the E5 semantics. E5 requires * that if -any- input to Math.max() is a NaN, the result is a * NaN. fmax() will return a NaN only if - both- inputs are NaN. * Same applies to fmin(). ** Note: every input value must be coerced with ToNumber(), even * if we know the result will be a NaN anyway: ToNumber() may have * side effects for which even order of evaluation matters.
13071. * x is finite and non-zero ** -1.6 -> floor(-1.1) -> -2 * -1.5 -> floor(-1.0) -> -1 (towards +Inf) * -1.4 -> floor(-0.9) -> -1 * -0.5 -> -0.0 (special case) * -0.1 -> -0.0 (special case) * +0.1 -> +0.0 (special case) * +0.5 -> floor(+1.0) -> 1 (towards +Inf) * +1.4 -> floor(+1.9) -> 1 * +1.5 -> floor(+2.0) -> 2 (towards +Inf) * +1.6 -> floor(+2.1) -> 2
13072. order must match constants in genbuiltins.py
13073. DUK_USE_AVOID_PLATFORM_FUNCPTRS
13074. * Math built-ins
13075. +0.5 is handled by floor, this is on purpose
13076. Math.pow(-0,y) where y<0 should be: * - -Infinity if y<0 and an odd integer * - Infinity otherwise * NetBSD pow() returns -Infinity for all finite y<0. The * if- clause also catches y == -Infinity (which works even * without the fix).
13077. Note: not normalized, but duk_push_number() will normalize
13078. **comment:** XXX: what's the safest way of creating a negative zero?
 label: code-design
13079. Math.pow(+0,y) should be Infinity when y<0. NetBSD pow() * returns -Infinity instead when y is <0 and finite. The * if-clause also catches y == -Infinity (which works even * without the fix).
13080. The ANSI C pow() semantics differ from Ecmascript. ** E.g. when x==1 and y is +/- infinite, the Ecmascript required * result is NaN, while at least Linux pow() returns 1.
13081. fmod() return value has same sign as input (negative) so * the result here will be in the range]-2,0], 1 indicates * odd. If x is -Infinity, NaN is returned and the odd check * always concludes "not odd" which results in desired outcome.
13082. * Use static helpers which can work with math.h functions matching * the following signatures. This is not portable if any of these math * functions is actually a macro. ** Typing here is intentionally 'double' wherever values interact with * the standard library APIs.
13083. The specification has quite awkward order of coercion and * checks for toPrecision(). The operations below are a bit * reordered, within constraints of observable side effects.
13084. * E5 Section 15.7.2.1 requires that the constructed object * must have the original Number.prototype as its internal * prototype. However, since Number.prototype is non-writable * and non-configurable, this doesn't have to be enforced here: * The default object (bound to 'this') is OK, though we have * to change its class. ** Internal value set to ToNumber(arg) or +0; if no arg given, * ToNumber(undefined) = NaN, so special treatment is needed * (above). String internal value is immutable.
13085. digits
13086. leading digit + fractions
13087. Used when precision is undefined; also used for NaN (-> "NaN"), * and +/- infinity (-> "Infinity", "-Infinity").
13088. XXX: shared helper fortoFixed(), toExponential(), toPrecision()
13089. * Number built-ins
13090. radix
13091. for side effects
13092. Number built-in accepts a plain number or a Number object (whose * internal value is operated on). Other types cause TypeError.
13093. **comment:** XXX: helper
 label: code-design
13094. -> [val obj val]
13095. flags
13096. no return value -> don't replace created value
13097. **comment:** XXX: just use toString() for now; permitted although not recommended. * nargs==1, so radix is passed to toString().
 label: code-design
13098. * toFixed(), toExponential(), toPrecision()
13099. **comment:** * The Number constructor uses ToNumber(arg) for number coercion * (coercing an undefined argument to NaN). However, if the * argument is not given at all, +0 must be used instead. To do * this, a vararg function is used.
 label: code-design
13100. target
13101. [hobject props enum(props) key desc]
13102. fall thru
13103. Lightfunc, always success.
13104. [hobject props enum(props)]
13105. **comment:** XXX: missing trap result validation for non-configurable target keys * (must be present), for non-extensible target all target keys must be * present and no extra keys can be present. * http://www.ecma-international.org/ecma-262/6.0/#sec-proxy-object-internal-methods-and-internal-slots-ownpropertykeys
 label: code-design
13106. Shared helper for Object.getOwnPropertyNames() and Object.keys(). * Magic: 0=getOwnPropertyNames, 1=Object.keys.
13107. -> [obj trap_result]
13108. enum_flags
13109. magic: 0 getter call, 1=Object.getPrototypeOf
13110. * Validate and convert argument property descriptor (an Ecmascript * object) into a set of defprop_flags and possibly property value, * getter, and/or setter values on the value stack. ** Lightfunc set/get values are coerced to full Functions.
13111. **comment:** XXX: for Object.keys() the [[OwnPropertyKeys]] result (trap result) * should be filtered so that only enumerable keys remain. Enumerability * should be checked with [[GetOwnProperty]] on the original object * (i.e., the proxy in this case). If the proxy has a getOwnPropertyDescriptor * trap, it should be triggered for every property. If the proxy doesn't have * the trap, enumerability should be checked against the target object instead. * We don't do any of this now, so Object.keys() and Object.getOwnPropertyNames() * return the same result now for proxy traps. We still do clean up the trap * result, so that Object.keys() and Object.getOwnPropertyNames() will return a * clean array of strings without gaps.
 label: code-design
13112. **comment:** XXX: no need for indirect call
 label: code-design
13113. Just call the "original" Object.defineProperties() to * finish up.
13114. Pointer and buffer primitive values are treated like other * primitives values which have a fully fledged object counterpart: * promote to an object value. Lightfuncs are coerced with * ToObject() even they could also be returned as is.
13115. Loop prevention
13116. -> [hobject props]
13117. h_new_proto may be NULL
13118. -> [O toString O]
13119. [obj Properties]
13120. [obj trap_result res_arr]

13121. properties object
13122. Careful with reachability here: don't pop 'obj' before pushing * proxy target.
13123. [obj trap_result res_arr proppname]
13124. **comment:** TODO: implement Proxy object support here
 label: requirement
13125. ignore_loop
13126. required_desc_flags
13127. get_value
13128. Shared helper to implement ES6 Object.setPrototypeOf and * Object.prototype.__proto__ setter. * * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__ * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.setprototypeof
13129. * Object built-ins
13130. * Use Object.defineProperty() helper for the actual operation.
13131. Preliminaries for __proto__ and setPrototypeOf (E6 19.1.2.18 steps 1-4); * magic: 0=setter call, 1=Object.setPrototypeOf
13132. [obj handler trap]
13133. h is NULL for lightfunc
13134. * Return target object
13135. A non-extensible object cannot gain any more properties, * so this is a good time to compact.
13136. **comment:** XXX: tail call: return duk_push_false(ctx)
 label: code-design
13137. [[SetPrototypeOf]] standard behavior, E6 9.1.2
13138. Sealed and frozen objects cannot gain any more properties, * so this is a good time to compact them.
13139. [obj key desc]
13140. [hobject props enum(props) key desc value? getter? setter?]
13141. frozen and sealed
13142. **comment:** Lightfuncs are currently supported by coercing to a temporary * Function object; changes will be allowed (the coerced value is * extensible) but will be lost.
 label: code-design
13143. Ignore the normalize/validate helper outputs on the value stack, * they're popped automatically.
13144. **comment:** XXX: for Object.keys() we should check enumerability of key
 label: code-design
13145. * Two pass approach to processing the property descriptors. * On first pass validate and normalize all descriptors before * any changes are made to the target object. On second pass * make the actual modifications to the target object. * * Right now we'll just use the same normalize/validate helper * on both passes, ignoring its outputs on the first pass.
13146. Object.keys
13147. **comment:** XXX: should the API call handle this directly, i.e. attempt * to duk_push_hobject(ctx, null) would push a null instead? * (On the other hand 'undefined' would be just as logical, but * not wanted here.)
 label: code-design
13148. __proto__ setter returns 'undefined' on success unlike the * setPrototypeOf() call which returns the target object.
13149. [O Properties obj]
13150. * Return target object.
13151. idx_desc
13152. retval for success path
13153. Lightfunc handling by ToObject() coercion.
13154. -> [obj trap handler target]
13155. nargs
13156. DUK_USE_ES6_PROXY
13157. is_frozen
13158. Object.getOwnPropertyNames
13159. **comment:** E5.1 Section 15.2.4.6, step 3.a, lookup proto once before compare. * Prototype loops should cause an error to be thrown.
 label: code-design
13160. Shared helper to implement Object.getPrototypeOf and the ES6 * Object.prototype.__proto__ getter. * * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__
13161. **comment:** XXX: call method tail call?
 label: code-design
13162. Pointer object internal value is immutable
13163. * Constructor
13164. * Pointer built-ins
13165. nop
13166. Must be a "pointer object", i.e. class "Pointer"
13167. Note: unbalanced stack on purpose
13168. * toString(), valueOf()
13169. **comment:** XXX: this behavior is quite useless now; it would be nice to be able * to create pointer values from e.g. numbers or strings. Numbers are * problematic on 64-bit platforms though. Hex encoded strings?
 label: code-design
13170. DUK_BUILTIN_PROTOS_H_INCLUDED
13171. * Prototypes for built-in functions not automatically covered by the * header declarations emitted by genbuiltins.py.
13172. Helpers exposed for internal use
13173. Buffer size needed for duk.bi_date_format_timeval(). * Accurate value is 32 + 1 for NUL termination: * >>> len('+123456-01-23T12:34:56.123+12:34') * 32 *
 Include additional space to be safe.
13174. Maximum length of CommonJS module identifier to resolve. Length includes * both current module ID, requested (possibly relative) module ID, and a * slash in between.
13175. Built-in providers
13176. Make _Target and _Handler non-configurable and non-writable. * They can still be forcibly changed by C code (both user and * Duktape internal), but not by EcmaScript code.
13177. Reject a proxy object as the target because it would need * special handler in property lookups. (ES6 has no such restriction)
13178. Proxy target
13179. Proxy handler
13180. DUK_USE_ES6_PROXY
13181. **comment:** XXX: the returned value is exotic in ES6, but we use a * simple object here with no prototype. Without a prototype, * [[DefaultValue]] coercion fails
 which is abit confusing. * No callable check/handling in the current Proxy subset.
 label: code-design
13182. Reject a proxy object as the handler because it would cause * potentially unbounded recursion. (ES6 has no such restriction)
13183. * Proxy built-in (ES6)
13184. replacement handler
13185. flags: "m"
13186. DUK_USE_REGEXEXP_SUPPORT
13187. * RegExp built-ins
13188. flags: "gim"

13189. [result]

13190. **comment:** A little tricky string approach to provide the flags string. * This depends on the specific flag values in duk_regex.h, * which needs to be asserted for. In practice this doesn't * produce more compact code than the easier approach in use.
label: code-design

13191. [... pattern flags]

13192. **comment:** This is a cleaner approach and also produces smaller code than * the other alternative. Use duk_require_string() for format * safety (although the source property should always exist).
label: code-design

13193. require to be safe

13194. three flags

13195. prepend regexp to valstack 0 index

13196. **comment:** XXX: much to improve (code size)
label: code-design

13197. flags: "gi"

13198. [... bytecode escaped_source]

13199. flags: "gm"

13200. [... RegExp]

13201. flags: ""

13202. Called as a function, pattern has [[Class]] "RegExp" and * flags is undefined -> return object as is.

13203. [regexp input]

13204. Else functionality is identical for function call and constructor * call.

13205. flags: "g"

13206. [regexp]

13207. This should not be necessary because no-one should tamper with the * regexp bytecode, but is prudent to avoid potential segfaults if that * were to happen for some reason.

13208. flags: "im"

13209. flags: "i"

13210. [regexp source bytecode]

13211. result object is created and discarded; wasteful but saves code space

13212. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array

13213. * The specification uses RegExp [[Match]] to attempt match at specific * offsets. We don't have such a primitive, so we use an actual RegExp * and tweak lastIndex. Since the RegExp may be non-global, we use a * special variant which forces global-like behavior for matching.

13214. * Case conversion

13215. **comment:** XXX: It would be nice to build the string directly but ToUint16() * coercion is needed so a generic helper would not be very * helpful (perhaps coerce the value stack first here and then * build a string from a duk_tval number sequence in one go?).
label: code-design

13216. lastIndex already set up for next match

13217. DUK_USE_REGEXP_SUPPORT

13218. Empty searchstring always matches; cpos must be clamped here. * (If q_blen were < 0 due to clamped coercion, it would also be * caught here.)

13219. [regexp string]

13220. **comment:** This loop is optimized for size. For speed, there should be * two separate loops, and we should ensure that memcmp() can be * used without an extra "will searchstring fit" check. Doing * the preconditioning for 'p' and 'p_end' is easy but cpos * must be updated if 'p' is wound back (backward scanning).
label: code-design

13221. The current implementation of localeCompare() is simply a codepoint * by codepoint comparison, implemented with a simple string compare * because UTF-8 should preserve codepoint ordering (assuming valid * shortest UTF-8 encoding). * * The specification requires that the return value must be related * to the sort order: e.g. negative means that 'this' comes before * 'that' in sort order. We assume an ascending sort order.

13222. prefix matches, lengths matter now

13223. min(incl)

13224. not found

13225. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer * stack[4] = regexp match OR match string

13226. For ECMAScript strings, this check can only match for * initial UTF-8 bytes (not continuation bytes). For other * strings all bets are off.

13227. match == search string, by definition

13228. res_obj

13229. ToUint16() coercion is mandatory in the E5.1 specification, but * this non-compliant behavior makes more sense because we support * non-BMP codepoints. Don't use CESU-8 because that'd create * surrogate pairs.

13230. match string

13231. empty match (may happen with empty separator) -> bump and continue

13232. * Various

13233. * toString(), valueOf()

13234. when going backwards, we decrement cpos 'early'; * 'p' may point to a continuation byte of the char * at offset 'cpos', but that's OK because we'll * backtrack all the way to the initial byte.

13235. trailer

13236. don't allow an empty match at the end of the string

13237. [... replacer match [captures] match_char_offset input]

13238. XXX: There are several limitations in the current implementation for * strings with $\geq 0x80000000UL$ characters. In some cases one would need * to be able to represent the range [-0xffffffff, 0xffffffff] and so on. * Generally character and byte length are assumed to fit into signed 32 * bits (< 0x80000000UL). Places with issues are not marked explicitly * below in all cases, look for signed type usage (duk_int_t etc) for * offsets/lengths.

13239. **comment:** XXX: could share code with duk_js_ops.c, duk_js_compare_helper
label: code-design

13240. * indexOf() and lastIndexOf()

13241. ch1 = (r_increment << 8) + byte

13242. * If matching with a regexp: * - non-global RegExp: lastIndex not touched on a match, zeroed * on a non-match * - global RegExp: on match, lastIndex will be updated by regexp * executor to point to next char after the matching part (so that * characters in the matching part are not matched again) * * If matching with a string: * - always non-global match, find first occurrence * * We need: * - The character offset of start-of-match for the replacer function * - The byte offsets for start-of-match and end-of-match to implement * the replacement values \$\$, \$, and \$', and to copy non-matching * input string portions (including header and trailer) verbatim. * * NOTE: the E5.1 specification is a bit vague how the RegExp should * behave in the replacement process; e.g. is matching done first for * all matches (in the global RegExp case) before any replacer calls * are made? See: test-bi-string-proto-replace.js for discussion.

13243. return as is

13244. repl_value

13245. Not found. Empty string case is handled specially above.

13246. stack[0] = regexp * stack[1] = string

13247. * substring(), substr(), slice()

13248. -> [... regexp string] -> [... res_obj]

13249. [start end str]

13250. never here, but fall through

13251. at index 1

13252. **comment:** this is relatively expensive
label: code-design

13253. no issues with memcmp() zero size, even if broken
13254. switch (ch2)
13255. lastIndex is initialized to zero by new RegExp()
13256. String constructor needs to distinguish between an argument not given at all * vs. given as 'undefined'. We're a vararg function to handle this properly.
13257. pop regexp res_obj or match string
13258. unconditionally
13259. start match from beginning
13260. h_match is borrowed, remains reachable through match_obj
13261. if (is_regexp)
13262. **comment:** XXX: faster implementation
 label: code-design
13263. Add trailer if: * a) non-empty input * b) empty input and no (zero size) match found (step 11)
13264. If the separator is a RegExp, make a "clone" of it. The specification * algorithm calls [[Match]] directly for specific indices; we emulate this * by tweaking lastIndex and using a "force global" variant of duk_reEXP_match() * which will use global-style matching even when the RegExp itself is non-global.
13265. DUK_USE_SECTION_B
13266. Zero size compare not an issue with DUK_MEMCMP.
13267. out_clamped
13268. **comment:** XXX: could improve bufwriter handling to write multiple codepoints * with one ensure call but the relative benefit would be quite small.
 label: code-design
13269. empty separator can always match
13270. leading byte of match string
13271. **comment:** XXX: the current implementation works but is quite clunky; it compiles * to almost 1.4kB of x86 code so it needs to be simplified (better approach, * shared helpers, etc). Some ideas for refactoring: * * - a primitive to convert a string into a regexp matcher (reduces matching * code at the cost of making matching much slower) * - use replace() as a basic helper for match() and split(), which are both * much simpler * - API call to get_prop and to_boolean
 label: code-design
13272. track utf-8 non-continuation bytes
13273. **comment:** XXX: optional check, match_caps is zero if no regexp, * so dollar will be interpreted literally anyway.
 label: code-design
13274. input size is good output starting point
13275. match_caps == 0 without regexps
13276. **comment:** XXX: very messy now, but works; clean up, remove unused variables (nominally * used so compiler doesn't complain).
 label: code-design
13277. unconditionally; is_global==0
13278. Unlike non-obsolete String calls, substr() algorithm in E5.1 * specification will happily coerce undefined and null to strings * ("undefined" and "null").
13279. string limits
13280. Easiest way to implement the search required by the specification * is to do a RegExp test() with lastIndex forced to zero. To avoid * side effects on the argument, "clone" the RegExp if a RegExp was * given as input. * * The global flag of the RegExp should be ignored; setting lastIndex * to zero (which happens when "cloning" the RegExp) should have an * equivalent effect.
13281. regexp res_obj is at index 4
13282. [... re_obj input]
13283. max(incl)
13284. Must be a "string object", i.e. class "String"
13285. The spec algorithm first does "R = ToString(separator)" before checking * whether separator is undefined. Since this is side effect free, we can * skip the ToString() here.
13286. indexOf: NaN should cause pos to be zero. * lastIndexOf: NaN should cause pos to be +Infinity * (and later be clamped to len).
13287. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer
13288. * Character and charcode access
13289. no action
13290. default case
13291. initial estimate for ASCII only codepoints
13292. * replace()
13293. cannot have >4G captures
13294. while repl
13295. return 'res_obj'
13296. never an empty match, so step 13.c.iii can't be triggered
13297. for
13298. no issues with empty memcmp()
13299. * Constructor
13300. Shared helper for match() steps 3-4, search() steps 3-4.
13301. track cpos while scanning
13302. 0=indexOf, 1=lastIndexOf
13303. regexp res_obj is at offset 4
13304. [start length str]
13305. **comment:** Global case is more complex.
 label: code-design
13306. * String built-ins
13307. return 'res_arr' or 'null'
13308. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array * stack[4] = regexp res_obj (if is_regexp)
13309. This should be equivalent to match() algorithm step 8.f.iii.2: * detect an empty match and allow it, but don't allow it twice.
13310. combines steps 2 and 5; -len ensures max() not needed for step 5
13311. force_new
13312. No arr_idx update or limit check
13313. set to 1 if any match exists (needed for empty input special case)
13314. * split()
13315. Combined step 11 (empty string special case) and 14-15.
13316. Avoid using RegExp.prototype methods, as they're writable and * configurable and may have been changed.
13317. match followed by capture(s)
13318. -> [... res_obj]
13319. empty match -> bump and continue
13320. -> [res_obj]
13321. [regexp string res_arr]
13322. String object internal value is immutable
13323. index
13324. repl string scan
13325. ditto
13326. Note: unbalanced stack on purpose
13327. **comment:** XXX: any chance of merging these three similar but still slightly * different algorithms so that footprint would be reduced?
 label: code-design

13328. ensure full memcmp() fits in while
13329. Use match charlen instead of bytelen, just in case the input and * match codepoint encodings would have different lengths.
13330. **comment:** combines steps 3, 6; step 7 is not needed
 label: requirement
13331. [... regexp input] -> [res_obj]
13332. -> [... repl_value]
13333. single match always
13334. constrained by string length
13335. if (is_repl_func)
13336. Handle empty separator case: it will always match, and always * triggers the check in step 13.c.iii initially. Note that we * must skip to either end of string or start of first codepoint, * skipping over any continuation bytes! ** Don't allow an empty string to match at the end of the input.
13337. duk_concat() coerces arguments with ToString() in correct order
13338. input string scan
13339. undefined -> skip (replaced with empty)
13340. [... RegExp val] -> [... res]
13341. The implementation for computing of start_pos and end_pos differs * from the standard algorithm, but is intended to result in the exactly * same behavior. This is not always obvious.
13342. * Thread state and calling context checks
13343. * Constructor
13344. [value]
13345. * Resume a thread. ** The thread must be in resumable state, either (a) new thread which hasn't * yet started, or (b) a thread which has previously yielded. This method * must be called from an Ecmascript function. ** Args: * - thread * - value * - isError (defaults to false) ** Note: yield and resume handling is currently asymmetric.
13346. in yielder's context
13347. return thread
13348. * Thread builtins
13349. never here
13350. no pre-checks now, assume a previous yield() has left things in * tip-top shape (longjmp handler will assert for these).
13351. XXX: need a duk_require_func_or_lfunc_coerce()
13352. in resumer's context
13353. Note: the only yield-preventing call is Duktape.Thread.yield(), hence check for 1, not 0
13354. value (error) is at stack top
13355. us
13356. * Process yield ** After longjmp(), processing continues in bytecode executor longjmp * handler, which will e.g. update thr->resumer to NULL.
13357. caller
13358. push initial function call to new thread stack; this is * picked up by resume().
13359. Note: there is no requirement that: 'thr->callstack_preventcount == 1' * like for yield.
13360. Note: cannot be a bound function either right now, * this would be easy to relax though.
13361. call is from executor, so we know we have a jmpbuf
13362. side effects
13363. execution resumes in bytecode executor
13364. * The error object has been augmented with a traceback and other * info from its creation point -- usually the current thread. * The error handler, however, is called right before throwing * and runs in the yielder's thread.
13365. * Yield the current thread. ** The thread must be in yieldable state: it must have a resumer, and there * must not be any yield-preventing calls (native calls and constructor calls, * currently) in the thread's call stack (otherwise a resume would not be * possible later). This method must be called from an Ecmascript function. ** Args: * - value * - isError (defaults to false) ** Note: yield and resume handling is currently asymmetric.
13366. lj value2: thread
13367. **comment:** should never be zero, because we (Duktape.Thread.yield) are on the stack
 label: code-design
13368. lj value1: value
13369. Further state-dependent pre-checks
13370. [thread value]
13371. * The error object has been augmented with a traceback and other * info from its creation point -- usually another thread. The * error handler is called here right before throwing, but it also * runs in the resumer's thread. It might be nice to get a traceback * from the resumee but this is not the case now.
13372. * Type error thrower, E5 Section 13.2.3.
13373. DUK_USE_ROM_OBJECTS
13374. * Automatically generated by genbuiltins.py, do not edit!
13375. DUK_USE_ROM_STRINGS
13376. native functions: 149
13377. DUK_USE_BUILTIN_INITJS
13378. 'length'
13379. !DUK_SINGLE_FILE
13380. 'T'
13381. 'Arguments'
13382. 'toLogString'
13383. '\xffNext'
13384. '\xffBytecode'
13385. '\xffLexenv'
13386. 'Date'
13387. 'instanceof'
13388. 'has'
13389. 'do'
13390. 'var'
13391. 'exports'
13392. 'pointer'
13393. 'void'
13394. 'string'
13395. 'protected'
13396. 'filename'
13397. 'try'
13398. 'continue'
13399. 'catch'
13400. 'toJSON'
13401. 'if'
13402. 'index'
13403. 'deleteProperty'
13404. 'value'
13405. 'Object'

13406. 'switch'
13407. 'toGMTString'
13408. 'let'
13409. ' [...] '
13410. 'enumerable'
13411. 'Uint32Array'
13412. DUK_USE_ROM_STRINGS
13413. ''
13414. 'enumerate'
13415. 'with'
13416. 'join'
13417. 'resume'
13418. 'debugger'
13419. 'get'
13420. ''
13421. '{"_inf":true}'
13422. '\xffFinalizer'
13423. '\xffArgs'
13424. 'Error'
13425. 'Float64Array'
13426. 'enum'
13427. 'default'
13428. '\xffRegbase'
13429. 'Thread'
13430. '\xffThread'
13431. 'import'
13432. 'set'
13433. 'export'
13434. 'prototype'
13435. 'ObjEnv'
13436. 'jx'
13437. 'fmt'
13438. 'typeof'
13439. 'Math'
13440. '-Infinity'
13441. 'name'
13442. 'BYTES_PER_ELEMENT'
13443. 'info'
13444. 'interface'
13445. 'class'
13446. 'Boolean'
13447. 'Buffer'
13448. 'defineProperty'
13449. '\xffVarmap'
13450. 'Uint16Array'
13451. To convert a heap stridx to a token number, subtract * DUK_STRIDX_START_RESERVED and add DUK_TOK_START_RESERVED.
13452. 'Infinity'
13453. '\xffFormals'
13454. 'implements'
13455. '\xffHandler'
13456. '{"_undef":true}'
13457. '\xffSource'
13458. '{_func:true}'
13459. 'env'
13460. 'toLocaleString'
13461. '\n'
13462. 'fatal'
13463. 'hex'
13464. 'trace'
13465. 'jc'
13466. 'case'
13467. 'toUTCString'
13468. '(?:)'
13469. '\xffTarget'
13470. 'private'
13471. 'Invalid Date'
13472. 'multiline'
13473. 'errCreate'
13474. 'warn'
13475. 'finally'
13476. 'ownKeys'
13477. 'configurable'
13478. 'caller'
13479. 'undefined'
13480. 'byteLength'
13481. 'lineNumber'
13482. 'Int8Array'
13483. 'data'
13484. 'type'
13485. '-0'
13486. '{"_ninf":true}'
13487. 'modLoaded'
13488. 'true'
13489. 'fileName'
13490. 'Uint8Array'
13491. 'break'
13492. 'Undefined'
13493. "

13494. 'setPrototypeOf'
13495. '\xffVarenv'
13496. 'number'
13497. '{"_func":true}'
13498. 'return'
13499. 'throw'
13500. 'valueOf'
13501. 'buffer'
13502. 'DecEnv'
13503. 'writable'
13504. DUK_USE_ROM_OBJECTS
13505. 'yield'
13506. 'id'
13507. 'callee'
13508. 'ArrayBuffer'
13509. 'Null'
13510. '\xffPc2line'
13511. '\xffMap'
13512. 'static'
13513. 'package'
13514. 'ignoreCase'
13515. 'message'
13516. 'NaN'
13517. 'byteOffset'
13518. 'extends'
13519. 'source'
13520. 'constructor'
13521. 'clog'
13522. 'raw'
13523. 'Int16Array'
13524. 'base64'
13525. 'in'
13526. 'Array'
13527. 'pc'
13528. 'errThrow'
13529. 'while'
13530. 'toISOString'
13531. 'Float32Array'
13532. 'object'
13533. 'require'
13534. 'compile'
13535. 'const'
13536. 'false'
13537. 'null'
13538. 'boolean'
13539. 'Int32Array'
13540. 'new'
13541. 'Function'
13542. 'stack'
13543. '\xffThis'
13544. 'toString'
13545. 'debug'
13546. '\xffCallee'
13547. 'Uint8ClampedArray'
13548. 'arguments'
13549. 'n'
13550. 'modSearch'
13551. 'for'
13552. DUK_USE_BUILTIN_INITJS
13553. 'eval'
13554. 'function'
13555. '\xffValue'
13556. 'DataView'
13557. 'error'
13558. '{"_nan":true}'
13559. 'delete'
13560. * Automatically generated by genbuiltins.py, do not edit!
13561. 'exclusive endpoint'
13562. 'RegExp'
13563. 'this'
13564. 'public'
13565. '__proto__'
13566. '\xffTracedata'
13567. 'input'
13568. 'super'
13569. 'lastIndex'
13570. 'JSON'
13571. 'else'
13572. 'Number'
13573. 'global'
13574. DUK_BUILTINS_H_INCLUDED
13575. 'String'
13576. 'Pointer'
13577. This should not really happen, but would indicate x64.
13578. **comment:** Minimize warnings for unused internal functions with GCC >= 3.1.1 and * Clang. Based on documentation it should suffice to have the attribute * in the declaration only, but in practice some warnings are generated unless * the attribute is also applied to the definition.
 label: code-design
13579. Emscripten (provided explicitly by user), improve if possible

13580. Both MinGW and MSVC have a 64-bit type.
13581. * Platform autodetection
13582. --- TinyC ---
13583. mint clib is missing these
13584. SuperH
13585. --- x64 ---
13586. MSVC does not have sys/param.h
13587. --- Motorola 68k ---
13588. since gcc-2.5
13589. --- MIPS 64-bit ---
13590. **comment:** XXX: DUK_UNREACHABLE for msvc?
 label: code-design
13591. OpenBSD
13592. NetBSD
13593. Apple OSX, iOS
13594. Cannot determine byte order; __ORDER_PDP_ENDIAN__ is related to 32-bit * integer ordering and is not relevant.
13595. Type for public API calls.
13596. **comment:** Rely as little as possible on compiler behavior for NaN comparison, * signed zero handling, etc. Currently never activated but may be needed * for broken compilers.
 label: code-design
13597. --- Generic BSD ---
13598. uclibc may be missing these
13599. FreeBSD
13600. not defined by default
13601. **comment:** Windows, both 32-bit and 64-bit
 label: code-design
13602. --- OpenBSD ---
13603. **comment:** * Alternative customization header * * If you want to modify the final DUK_USE_xxx flags directly (without * using the available DUK_OPT_xxx flags), define DUK_OPT_HAVE_CUSTOM_H * and tweak the final flags there.
 label: code-design
13604. Cygwin
13605. integer endianness is little on purpose
13606. e.g. getdate_r
13607. * Autogenerated defaults
13608. Atari Mint
13609. **comment:** GCC older than 4.6: avoid overflow warnings related to using INFINITY
 label: code-design
13610. Most portable, wastes space
13611. GCC/clang inaccurate math would break compliance and probably duk_tval, * so refuse to compile. Relax this if -ffast-math is tested to work.
13612. --- Generic ---
13613. VS2012+ has stdint.h, < VS2012 does not (but it's available for download).
13614. **comment:** * Alignment requirement and support for unaligned accesses * * Assume unaligned accesses are not supported unless specifically allowed * in the target platform. Some platforms may support unaligned accesses * but alignment to 4 or 8 may still be desirable.
 label: requirement
13615. Clang
13616. AmigaOS. Neither AMIGA nor __amigaos__ is defined on VBCC, so user must * define 'AMIGA' manually when using VBCC.
13617. --- GCC ---
13618. Duktape/C function return value, platform int is enough for now to * represent 0, 1, or negative error code. Must be compatible with * assigning truth values (e.g. duk_ret_t rc = (foo == bar);).
13619. Low memory algorithm: separate chaining using arrays, fixed size hash
13620. __ OVERRIDE_DEFINES __
13621. <http://stackoverflow.com/questions/5919996/how-to-detect-reliably-mac-os-x-ios-linux-windows-in-c-preprocessor>
13622. vbcc + AmigaOS has C99 but no inttypes.h
13623. MSVC
13624. * Check whether or not a packed duk_tval representation is possible. * What's basically required is that pointers are 32-bit values * (sizeof(void *) == 4). Best effort check, not always accurate. * If guess goes wrong, crashes may result; self tests also verify * the guess.
13625. --- Linux ---
13626. --- AmigaOS ---
13627. BCC, assume we're on x86.
13628. Pointer size determination based on __WORDSIZE or architecture when * that's not available.
13629. Unsigned index variant.
13630. External provider already defined.
13631. Byte order varies, so rely on autodetect.
13632. DUK_USE_UNION_INITIALIZERS: required from compilers, so no fill-in.
13633. **comment:** Special naming to avoid conflict with e.g. DUK_FREE() in duk_heap.h * (which is unfortunately named). May sometimes need replacement, e.g. * some compilers don't handle zero length or NULL correctly in realloc().
 label: code-design
13634. empty
13635. --- MSVC ---
13636. We're generally assuming that we're working on a platform with a 32-bit * address space. If DUK_SIZE_MAX is a typecast value (which is necessary * if SIZE_MAX is missing), the check must be avoided because the * preprocessor can't do a comparison.
13637. 64-bit constants. Since LL / ULL constants are not always available, * use computed values. These values can't be used in preprocessor * comparisons; flag them as such.
13638. --- x86 ---
13639. --- MIPS 32-bit ---
13640. XXX: add feature options to force basic types from outside?
13641. Array index values, could be exact 32 bits. * Currently no need for signed duk_arridx_t.
13642. SIZE_MAX may be missing so use an approximate value for it.
13643. DUK_F_BCC
13644. Complex condition broken into separate parts.
13645. POSIX
13646. If not provided, use safe default for alignment.
13647. * Date provider selection * * User may define DUK_USE_DATE_GET_NOW() etc directly, in which case we'll * rely on an external provider. If this is not done, revert to previous * behavior and use Unix/Windows built-in provider.
13648. Cannot determine byte order.
13649. --- Cygwin ---
13650. !defined(DUK_USE_BYTORDER) && defined(__BYTE_ORDER__)

13651. More or less standard endianness predefines provided by header files. * The ARM hybrid case is detected by assuming that `__FLOAT_WORD_ORDER` will be big endian, see: <http://lists.mysql.com/internals/443>. * On some platforms some defines may be present with an empty value which * causes comparisons to fail: <https://github.com/svaarala/duktape/issues/453>.
13652. !defined(DUK_USE_BYTEORDER)
13653. **comment:** For custom platforms allow user to define byteorder explicitly. * Since endianness headers are not standardized, this is a useful * workaround for custom platforms for which endianness detection * is not directly supported. Perhaps custom hardware is used and * user cannot submit upstream patches.
label: code-design
13654. **comment:** Don't know how to declare unreachable point, so don't do it; this * may cause some spurious compilation warnings (e.g. "variable used * uninitialized").
label: code-design
13655. Good default is a bit arbitrary because alignment requirements * depend on target. See <https://github.com/svaarala/duktape/issues/102>.
13656. --- Windows ---
13657. User provided InitJS.
13658. These have been tested from VS2008 onwards; may work in older VS versions * too but not enabled by default.
13659. Error codes are represented with platform int. High bits are used * for flags and such, so 32 bits are needed.
13660. Intermediate define for 'have inttypes.h'
13661. build is not C99 or C++11, play it safe
13662. GCC: test not very accurate; enable only in relatively recent builds * because of bugs in gcc-4.4 (<http://lists.debian.org/debian-gcc/2010/04/msg00000.html>)
13663. Intel x86 (32-bit), x64 (64-bit) or x32 (64-bit but 32-bit pointers), * define only one of DUK_F_X86, DUK_F_X64, DUK_F_X32. * <https://sites.google.com/site/x32abi/>
13664. Enabled with debug/assertions just so that any issues can be caught.
13665. C99 or compatible
13666. --- Mac OSX, iPhone, Darwin ---
13667. C99 or above
13668. MinGW. Also GCC flags (DUK_F_GCC) are enabled now.
13669. This detection is not very reliable.
13670. Support for 48-bit signed integer duk_tval with transparent semantics.
13671. GCC and Clang provide endianness defines as built-in predefines, with * leading and trailing double underscores (e.g. `__BYTE_ORDER__`). See * output of "make gccpredefs" and "make clangpredefs". Clang doesn't * seem to provide `__FLOAT_WORD_ORDER__`; assume not mixed endian for clang. * <http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html>
13672. 64-bit type detection is a bit tricky. * * `ULLONG_MAX` is a standard define. `__LONG_LONG_MAX__` and `__ULONGLONG_MAX__` * are used by at least GCC (even if system headers don't provide `ULLONG_MAX`). * Some GCC variants may provide `__LONG_LONG_MAX__` but not `__ULONGLONG_MAX__`. * * ULL / LL constants are rejected / warned about by some compilers, even if * the compiler has a 64-bit type and the compiler/system headers provide an * unsupported constant (ULL/LL)! Try to avoid using ULL / LL constants. * As a side effect we can only check that e.g. `ULONG_MAX` is larger than 32 * bits but can't be sure it is exactly 64 bits. Self tests will catch such * cases.
13673. Convert any input pointer into a "void **", losing a const qualifier. * This is not fully portable because casting through `duk_uintptr_t` may * not work on all architectures (e.g. those with long, segmented pointers).
13674. Small integers (16 bits or more) can fall back to the 'int' type, but * have a typedef so they are marked "small" explicitly.
13675. **comment:** Most portable, potentially wastes space
label: code-design
13676. PowerPC
13677. --- Solaris ---
13678. **comment:** Technically C99 (C++11) but found in many systems. On some systems * `__STDC_LIMIT_MACROS` and `__STDC_CONSTANT_MACROS` must be defined before * including stdint.h (see above).
label: code-design
13679. * `duk_config.h` configuration header generated by genconfig.py. * * Git commit: `cad34ae155acb0846545ca6bf2d29f9463b22bbb` * Git describe: v1.5.2 * Git branch: HEAD * * Supported platforms: * - Mac OSX, iPhone, Darwin * - OpenBSD * - Generic BSD * - Atari ST TOS * - AmigaOS * - Windows * - Flashplayer (Crossbridge) * - QNX * - TI-Nspire * - Emscripten * - Linux * - Solaris * - Generic POSIX * - Cygwin * - Generic UNIX * - Generic fallback * * Supported architectures: * - x86 * - x64 * - x32 * - ARM 32-bit * - ARM 64-bit * - MIPS 32-bit * - MIPS 64-bit * - PowerPC 32-bit * - PowerPC 64-bit * - SPARC 32-bit * - SPARC 64-bit * - SuperH * - Motorola 68k * - Emscripten * - Generic * * Supported compilers: * - Clang * - GCC * - MSVC * - Emscripten * - TinyC * - VBCC * - Bruce's C compiler * - Generic *
13680. Missing some obvious constants.
13681. No provider for `DUK_USE_DATE_PARSE_STRING()`, fall back to ISO 8601 only.
13682. no endian.h or stdint.h
13683. --- Atari ST TOS ---
13684. GCC: assume we have `__va_copy()` in non-C99 mode.
13685. User forced alignment to 4 or 8.
13686. C99 / C++11 and above: rely on `va_copy()` which is required.
13687. <http://bellard.org/tcc/tcc-doc.html#SEC9>
13688. Check that architecture is two's complement, standard C allows e.g. * `INT_MIN` to be $-2^{31}+1$ (instead of -2^{31}).
13689. Based on 'make checkalign' there are no alignment requirements on * Linux MIPS except for doubles, which need align by 4. Alignment * requirements vary based on target though.
13690. AmigaOS on M68k
13691. not configured for DLL build
13692. * You may add overriding #define/#undef directives below for * customization. You of course cannot un-#include or un-typedef * anything; these require direct changes above.
13693. **comment:** cannot detect 64-bit type, not always needed so don't error
label: code-design
13694. --- Generic UNIX ---
13695. std::exception
13696. A few types are assumed to always exist.
13697. Float word order not known, assume not a hybrid.
13698. --- Generic fallback ---
13699. **comment:** --- Generic POSIX ---
label: code-design
13700. no parsing (not an error)
13701. **comment:** ANSI C (various versions) and some implementations require that the * pointer arguments to `memset()`, `memcpy()`, and `memmove()` be valid values * even when byte size is 0 (even a NULL pointer is considered invalid in * this context). Zero-size operations as such are allowed, as long as their * pointer arguments point to a valid memory area. The `DUK_MEMSET()`, * `DUK_MEMCPY()`, and `DUK_MEMMOVE()` macros require this same behavior, i.e.: * (1) pointers must be valid and non-NULL, (2) zero size must otherwise be * allowed. If these are not fulfilled, a macro wrapper is needed. * * <http://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0> * <http://lists.cs.uiuc.edu/pipermail/l1vmdev/2007-October/011065.html> * * Not sure what's the required behavior when a pointer points just past the * end of a buffer, which often happens in practice (e.g. zero size memmoves). * For example, if allocation size is 3, the following pointer would not * technically point to a valid memory byte: * * <- alloc --> * | 0 | 1 | 2 | * ^ - p=3, points after last valid byte (2)
label: code-design
13702. C++11 or above
13703. SPARC byte order varies so rely on autodetection.
13704. `DUK_F_PACKED_TVAL_PROVIDED`
13705. <http://www.monkey.org/openbsd/archive/ports/0401/msg00089.html>

13706. VBCC
13707. sigsetjmp() alternative
13708. GCC, Clang also defines __GNUC__ so don't detect GCC if using Clang.
13709. --- SuperH ---
13710. Pre-C99: va_list type is implementation dependent. This replacement * assumes it is a plain value so that a simple assignment will work. * This is not the case on all platforms (it may be a single-array element, * for instance).
13711. **comment:** The best type for an "all around int" in Duktape internals is "at least * 32 bit signed integer" which is most convenient. Same for unsigned type. * Prefer 'int' when large enough, as it is almost always a convenient type.
- label:** code-design
13712. uclibc
13713. Explicit marker needed; may be 'defined', 'undefined', 'or 'not provided'.
13714. No provider for DUK_USE_DATE_FORMAT_STRING(), fall back to ISO 8601 only.
13715. **comment:** Many platforms are missing fpclassify() and friends, so use replacements * if necessary. The replacement constants (FP_NAN etc) can be anything but * match Linux constants now.
- label:** code-design
13716. * Checks for config option consistency (DUK_USE_xxx)
13717. autodetect compiler
13718. Rely on C89 headers only; time.h must be here.
13719. These functions don't currently need replacement but are wrapped for * completeness. Because these are used as function pointers, they need * to be defined as concrete C functions (not macros).
13720. --- TI-Nspire ---
13721. **comment:** We need va_copy() which is defined in C99 / C++11, so an awkward * replacement is needed for pre-C99 / pre-C++11 environments. This * will quite likely need portability hacks for some non-C99 * environments.
- label:** code-design
13722. AmigaOS on M68K or PPC is always big endian.
13723. Feature option forcing.
13724. --- ARM 64-bit ---
13725. * Feature option handling
13726. **comment:** * Wrapper typedefs and constants for integer types, also sanity check types. * C99 typedefs are quite good but not always available, and we want to avoid * forcibly redefining the C99 typedefs. So, there are Duktape wrappers for * all C99 typedefs and Duktape code should only use these typedefs. Type * detection when C99 is not supported is best effort and may end up detecting * some types incorrectly. * * Pointer sizes are a portability problem: pointers to different types may * have a different size and function pointers are very difficult to manage * portably. * * http://en.wikipedia.org/wiki/C_data_types#Fixed-width_integer_types * * Note: there's an interesting corner case when trying to define minimum * signed integer value constants which leads to the current workaround of * defining e.g. -0x80000000 as (-0x7fffffffL - 1L). See doc/code-issues.txt * for a longer discussion. * * Note: avoid typecasts and computations in macro integer constants as they * can then no longer be used in macro relational expressions (such as * #if DUK_SIZE_MAX < 0xffffffffUL). There is internal code which relies on * being able to compare DUK_SIZE_MAX against a limit.
- label:** code-design
13727. Not standard but common enough
13728. --- QNX ---
13729. **comment:** Object property allocation layout has implications for memory and code * footprint and generated code size/speed. The best layout also depends * on whether the platform has alignment requirements or benefits from * having mostly aligned accesses.
- label:** code-design
13730. (v)sprintf() is missing before MSVC 2015. Note that _(v)sprintf() does * NOT NUL terminate on truncation, but Duktape code never assumes that. *
- <http://stackoverflow.com/questions/2915672/snprintf-and-visual-studio-2010>
13731. Motorola 68K. Not defined by VBCC, so user must define one of these * manually when using VBCC.
13732. [http://msdn.microsoft.com/en-us/library/aa235362\(VS.60\).aspx](http://msdn.microsoft.com/en-us/library/aa235362(VS.60).aspx)
13733. Non-C99 case, still relying on DUK_UINTPTR_MAX, as long as it is not a computed value
13734. <http://bellard.org/tcc/tcc-doc.html#SEC7>
13735. * Intermediate helper defines
13736. **comment:** Macro hackery to convert e.g. __LINE__ to a string without formatting, * see: <http://stackoverflow.com/questions/240353/convert-a-preprocessor-token-to-a-string>
- label:** code-design
13737. Flash player (e.g. Crossbridge)
13738. no endian.h
13739. Shared includes: C89
13740. integer byte order
13741. float word order
13742. * Convert DUK_USE_BYTEORDER, from whatever source, into currently used * internal defines. If detection failed, #error out.
13743. e.g. ptrdiff_t
13744. **comment:** XXX: DUK_NOINLINE, DUK_INLINE, DUK_ALWAYS_INLINE for msvc?
- label:** code-design
13745. SPARC
13746. VBCC supports C99 so check only for C99 for union initializer support. * Designated union initializers would possibly work even without a C99 check.
13747. C++11 apparently ratified stdint.h
13748. QNX gcc cross compiler seems to define e.g. __LITTLEENDIAN__ or __BIGENDIAN__: * \$ /opt/qnx650/host/linux/x86/usr/bin/i486-pc-nto-qnx6.5.0-gcc -dM -E - </dev/null | grep -ni endian * 67:#define __LITTLEENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/mips-unknown-nto-qnx6.5.0-gcc -dM -E - </dev/null | grep -ni endian * 81:#define __BIGENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/arm-unknown-nto-qnx6.5.0-gcc -dM -E - </dev/null | grep -ni endian * 70:#define __LITTLEENDIAN__ 1
13749. At least some uclibc versions have broken floating point math. For * example, fpclassify() can incorrectly classify certain NaN formats. * To be safe, use replacements.
13750. Placeholder fix for (detection is wider than necessary): * http://llvm.org/bugs/show_bug.cgi?id=17788
13751. snprintf() is technically not part of C89 but usually available.
13752. --- x32 ---
13753. Basic integer typedefs and limits, preferably from inttypes.h, otherwise * through automatic detection.
13754. **comment:** Only include when compiling Duktape to avoid polluting application build * with a lot of unnecessary defines.
- label:** code-design
13755. --- VBCC ---
13756. On some systems SIZE_MAX can be smaller than max unsigned 32-bit value * which seems incorrect if size_t is (at least) an unsigned 32-bit type. * However, it doesn't seem useful to error out compilation if this is the * case.
13757. **comment:** On other platforms use layout 2, which requires some padding but * is a bit more natural than layout 3 in ordering the entries. Layout * 3 is currently not used.
- label:** code-design
13758. VS2005+ should have variadic macros even when they're not C99.
13759. illumos / Solaris
13760. DUK_COMPILING_DUKTAPE
13761. DUK_OPT_FORCE_BYTEORDER
13762. QNX
13763. since gcc-4.5

13764. **comment:** Windows 32-bit and 64-bit are currently the same.
label: code-design
13765. This is optionally used by panic handling to cause the program to segfault * (instead of e.g. abort()) on panic. Valgrind will then indicate the C * call stack leading to the panic.
13766. IEEE float/double typedef.
13767. --- ARM 32-bit ---
13768. **comment:** XXX: This is technically not guaranteed because it's possible to configure * an x86 to require aligned accesses with Alignment Check (AC) flag.
label: code-design
13769. TinyC
13770. **comment:** Note: the funny looking computations for signed minimum 16-bit, 32-bit, and * 64-bit values are intentional as the obvious forms (e.g. -0x80000000L) are * -not- portable. See code-issues.txt for a detailed discussion.
label: code-design
13771. These are necessary wild guesses.
13772. Already provided above
13773. Atari ST TOS. __TOS__ defined by PureC. No platform define in VBCC * apparently, so to use with VBCC user must define __TOS__ manually.
13774. Byte order is big endian but cannot determine IEEE double word order.
13775. --- Flashplayer (Crossbridge) ---
13776. MIPS byte order varies so rely on autodetection.
13777. stdint.h not available
13778. **comment:** The most portable way to figure out local time offset is gmtime(), * but it's not thread safe so use with caution.
label: requirement
13779. Codepoint type. Must be 32 bits or more because it is used also for * internal codepoints. The type is signed because negative codepoints * are used as internal markers (e.g. to mark EOF or missing argument). * (X)UTF-8/CESU-8 encode/decode take and return an unsigned variant to * ensure duk_uint32_t casts back and forth nicely. Almost everything * else uses the signed one.
13780. DUK_USE_VARIADIC_MACROS: required from compilers, so no fill-in.
13781. TOS on M68K is always big endian.
13782. don't use strftime() for now
13783. **comment:** not sure, not needed with C99 anyway
label: requirement
13784. C++ doesn't have standard designated union initializers ({ .foo = 1 }).
13785. --- Emscripten ---
13786. nop
13787. DLL build detection
13788. **comment:** Avoid custom date parsing and formatting for portability.
label: code-design
13789. Workaround for older C++ compilers before including <inttypes.h>, * see e.g.: https://sourceware.org/bugzilla/show_bug.cgi?id=15366
13790. * Compiler autodetection
13791. --- SPARC 32-bit ---
13792. --- Clang ---
13793. On some platforms int is 16-bit but long is 32-bit (e.g. PureC)
13794. **comment:** * Byte order and double memory layout detection * * Endianness detection is a major portability hassle because the macros * and headers are not standardized. There's even variance across UNIX * platforms. Even with "standard" headers, details like underscore count * varies between platforms, e.g. both __BYTE_ORDER and _BYTE_ORDER are used * (Crossbridge has a single underscore, for instance). * * The checks below are structured with this in mind: several approaches are * used, and at the end we check if any of them worked. This allows generic * approaches to be tried first, and platform/compiler specific hacks tried * last. As a last resort, the user can force a specific endianness, as it's * not likely that automatic detection will work on the most exotic platforms. * * Duktape supports little and big endian machines. There's also support * for a hybrid used by some ARM machines where integers are little endian * but IEEE double values use a mixed order (12345678 -> 43218765). This * byte order for doubles is referred to as "mixed endian".
label: code-design
13795. **comment:** Check whether we should use 64-bit integers or not. * * Quite incomplete now. Use 64-bit types if detected (C99 or other detection) * unless they are known to be unreliable. For instance, 64-bit types are * available on VBCC but seem to misbehave.
label: code-design
13796. VS2013+ supports union initializers but there's a bug involving union-inside-struct: * <https://connect.microsoft.com/VisualStudio/feedback/details/805981> * The bug was fixed (at least) in VS2015 so check for VS2015 for now: * <https://blogs.msdn.microsoft.com/vcblog/2015/07/01/c-compiler-front-end-fixes-in-vs2015/> * Manually tested using VS2013, CL reports 18.00.31101, so enable for VS2013 too.
13797. no user declarations
13798. DUK_CONFIG_H_INCLUDED
13799. BCC (Bruce's C compiler): this is a "torture target" for compilation
13800. Generic Unix (includes Cygwin)
13801. There was a curious bug where test-bi-date-canceling.js would fail e.g. * on 64-bit Ubuntu, gcc-4.8.1, -m32, and no -std=c99. Some date computations * using doubles would be optimized which then broke some corner case tests. * The problem goes away by adding 'volatile' to the datetime computations. * Not sure what the actual triggering conditions are, but using this on * non-C99 systems solves the known issues and has relatively little cost * on other platforms.
13802. C99 / C++11 and above: rely on va_copy() which is required. * Omit parenthesis on macro right side on purpose to minimize differences * to direct use.
13803. C99 types
13804. DUK_SIZE_MAX (= SIZE_MAX) is often reliable
13805. --- PowerPC 64-bit ---
13806. See: /opt/qnx650/target/qnx6/usr/include/sys/platform.h
13807. --- SPARC 64-bit ---
13808. On platforms without any alignment issues, layout 1 is preferable * because it compiles to slightly less code and provides direct access * to property keys.
13809. Strict C99 case: DUK_UINTPTR_MAX (= UINTPTR_MAX) should be very reliable
13810. autodetect platform
13811. **comment:** NetBSD 6.0 x86 (at least) has a few problems with pow() semantics, * see test-bug-netbsd-math-pow.js. Use NetBSD specific workaround. * (This might be a wider problem; if so, generalize the define name.)
label: code-design
13812. autodetect architecture
13813. In VBCC (0.0 / 0.0) results in a warning and 0.0 instead of NaN. * In MSVC (VS2010 Express) (0.0 / 0.0) results in a compile error. * Use a computed NaN (initialized when a heap is created at the * latest).
13814. Clang: assume we have __va_copy() in non-C99 mode.
13815. --- Bruce's C compiler ---
13816. Same as 'duk_int_t' but guaranteed to be a 'fast' variant if this * distinction matters for the CPU. These types are used mainly in the * executor where it might really matter.
13817. vbcc + AmigaOS may be missing these
13818. C99 or C++11, no known issues
13819. **comment:** XXX: DUK_LIKELY, DUK_UNLIKELY for msvc?
label: code-design
13820. Initial fix: disable secure CRT related warnings when compiling Duktape * itself (must be defined before including Windows headers). Don't define * for user code including duktape.h.
13821. varargs
13822. * Fill-ins for platform, architecture, and compiler

13823. e.g. `strptime`
13824. **comment:** Note: PRS and FMT are intentionally left undefined for now. This means * there is no platform specific date parsing/formatting but there is still * the ISO 8601 standard format.
- label:** code-design
13825. `vsnprintf()` is technically not part of C89 but usually available.
13826. Rely on autodetection for byte order, alignment, and packed tval.
13827. AmigaOS + M68K seems to have math issues even when using GCC cross * compilation. Use replacements for all AmigaOS versions on M68K * regardless of compiler.
13828. Byte order is little endian but cannot determine IEEE double word order.
13829. C++
13830. MIPS. Related defines: `_MIPSEB_`, `_MIPSEL_`, `_mips_isa_rev`, `_LP64_`
13831. VBCC is missing the built-ins even in C99 mode (perhaps a header issue).
13832. `_MSC_VER`
13833. **comment:** Macro for suppressing warnings for potentially unreferenced variables. * The variables can be actually unreferenced or unreferenced in some * specific cases only; for instance, if a variable is only debug printed, * it is unreferenced when debug printing is disabled.
- label:** code-design
13834. Byte order varies, rely on autodetection.
13835. Use `_setjmp()` on Apple by default, see GH-55.
13836. TI-Nspire (using Ndless)
13837. --- PowerPC 32-bit ---
13838. Some math functions are C99 only. This is also an issue with some * embedded environments using uclibc where uclibc has been configured * not to provide some functions. For now, use replacements whenever * using uclibc.
13839. Most portable
13840. Boolean values are represented with the platform 'int'.
13841. Shared includes: `stdint.h` is C99
13842. BSD variant
13843. Old uclibcs have a broken `memcpy` so use `memmove` instead (this is overly wide * now on purpose): <http://lists.uclibc.org/pipermail/uclibc-cvs/2008-October/025511.html>
13844. `date.h` is omitted, and included per platform
13845. * Architecture autodetection
13846. The most portable current time provider is `time()`, but it only has a * one second resolution.
13847. Convenience, e.g. gcc 4.5.1 == 40501; <http://stackoverflow.com/questions/6031819/emulating-gccs-built-in-unreachable>
13848. `defined(DUK_USE_BYTEORDER)`
13849. In VBCC (1.0 / 0.0) results in a warning and 0.0 instead of infinity. * Use a computed infinity (initialized when a heap is created at the * latest).
13850. For now, hash part is dropped if and only if 16-bit object fields are used.
13851. MSVC dllexport/dllimport: appropriate `_declspec` depends on whether we're * compiling Duktape or the application.
13852. **comment:** On Windows, assume we're little endian. Even Itanium which has a * configurable endianness runs little endian in Windows.
- label:** code-design
13853. **comment:** Compiler specific hackery needed to force struct size to match alignment, * see e.g. `duk_hbuffer.h`. * * <http://stackoverflow.com/questions/11130109/c-struct-size-alignment> * <http://stackoverflow.com/questions/10951039/specifying-64-bit-alignment>
- label:** code-design
13854. Index values must have at least 32-bit signed range.
13855. **comment:** Workaround for GH-323: avoid inlining control when compiling from * multiple sources, as it causes compiler portability trouble.
- label:** code-design
13856. byte order
13857. Linux
13858. Based on 'make checkalign' there are no alignment requirements on * Linux SH4, but align by 4 is probably a good basic default.
13859. MSVC preprocessor defines: <http://msdn.microsoft.com/en-us/library/b0084kay.aspx> * `_MSC_FULL_VER` includes the build number, but it has at least two formats, see e.g. * `BOOST_MSVC_FULL_VER` in http://www.boost.org/doc/libs/1_52_0/boost/config/compiler/visualc.hpp
13860. ARM
13861. Fast variants of small integers, again for really fast paths like the * executor.
13862. **comment:** When C99 types are not available, we use heuristic detection to get * the basic 8, 16, 32, and (possibly) 64 bit types. The fast/least * types are then assumed to be exactly the same for now: these could * be improved per platform but C99 types are very often now available. * 64-bit types are not available on all platforms; this is OK at least * on 32-bit platforms. * * This detection code is necessarily a bit hacky and can provide `typedefs` * and defines that won't work correctly on some exotic platform.
- label:** code-design
13863. **comment:** parameter passing, not thread safe
- label:** requirement
13864. args go here in parens
13865. * Structs
13866. Note: combining `_FILE_`, `_LINE_`, and `_func_` into fmt would be * possible compile time, but waste some space with shared function names.
13867. `DUK_USE_VARIADIC_MACROS`
13868. * Prototypes
13869. **comment:** * Debugging macros, `DUK_DPRINT()` and its variants in particular. * * `DUK_DPRINT()` allows formatted debug prints, and supports standard * and Duktape specific formatters. See `duk_debug_vsnprintf.c` for details. * * `DUK_D(x)`, `DUK_DD(x)`, and `DUK_DDD(x)` are used together with log macros * for technical reasons. They are concretely used to hide 'x' from the * compiler when the corresponding log level is disabled. This allows * clean builds on non-C99 compilers, at the cost of more verbose code. * Examples: * * `DUK_D(DUK_DPRINT("foo"))`; * `DUK_DD(DUK_DPRINT("foo"))`; * `DUK_DDD(DUK_DPRINT("foo"))`; * * This approach is preferable to the old "double parentheses" hack because * double parentheses make the C99 solution worse: `_FILE_` and `_LINE_` can * no longer be added transparently without going through globals, which * works poorly with threading.
- label:** code-design
13870. `DUK_DEBUG_H_INCLUDED`
13871. omit
13872. **comment:** Without variadic macros resort to comma expression trickery to handle debug * prints. This generates a lot of harmless warnings. These hacks are not * needed normally because `DUK_D()` and friends will hide the entire debug log * statement from the compiler.
- label:** code-design
13873. args go here as a comma expression in parens
13874. * Exposed debug macros: debugging disabled
13875. `DUK_USE_DEBUG`
13876. args
13877. * Exposed debug macros: debugging enabled
13878. **comment:** unused
- label:** code-design
13879. (maybe) truncated
13880. normal
13881. * Fixed buffer helper useful for debugging, requires no allocation * which is critical for debugging.
13882. actual chars dropped (not just NUL term)
13883. `DUK_USE_DEBUG`
13884. error

13885. * Debug dumping of duk_heap.
13886. Note: there is no standard formatter for function pointers
13887. DUK_USE_DEBUG
13888. heap->strs: not worth dumping
13889. **comment:** unused
 label: code-design
13890. **comment:** * Debugging macro calls.
 label: code-design
13891. http://en.wikipedia.org/wiki/ANSI_escape_code
13892. DUK_USE_DPRINT_COLORS
13893. DUK_USE_VARIADIC_MACROS
13894. * Debugging disabled
13895. DUK_USE_DEBUG
13896. * Debugging enabled
13897. %lf, %ld etc
13898. encode \xffBar as _Bar if no quotes are applied, this is for * readable internal keys.
13899. XXX: print built-ins array
13900. %ld
13901. terminal type: no depth check
13902. may be NULL
13903. XXX: use DUK_HSTRING_FLAG_INTERNAL?
13904. **comment:** * Custom formatter for debug printing, allowing Duktape specific data * structures (such as tagged values and heap objects) to be printed with * a nice format string. Because debug printing should not affect execution * state, formatting here must be independent of execution (see implications * below) and must not allocate memory. * * Custom format tags begin with a '%!' to safely distinguish them from * standard format tags. The following conversions are supported: * * %!T tagged value (duk_tval *) * %!O heap object (duk_heaphdr *) * %!I decoded bytecode instruction * %!C bytecode instruction opcode name (arg is long) * * Everything is serialized in a JSON-like manner. The default depth is one * level, internal prototype is not followed, and internal properties are not * serialized. The following modifiers change this behavior: * * @ print pointers * # print binary representations (where applicable) * d deep traversals of own properties (not prototype) * p follow prototype chain (useless without 'd') * i include internal properties (other than prototype) * x hexdump buffers * h heavy formatting * * For instance, the following serializes objects recursively, but does not * follow the prototype chain nor print internal properties: "%!dO". * * Notes: * * * Standard snprintf return value semantics seem to vary. This * implementation returns the number of bytes it actually wrote * (excluding the null terminator). If retval == buffer size, * output was truncated (except for corner cases). * * * Output format is intentionally different from Ecmascript * formatting requirements, as formatting here serves debugging * of internals. * * * Depth checking (and updating) is done in each type printer * separately, to allow them to call each other freely. * * * Some pathological structures might take ages to print (e.g. * self recursion with 100 properties pointing to the object * itself). To guard against these, each printer also checks * whether the output buffer is full; if so, early exit. * * * Reference loops are detected using a loop stack.
 label: code-design
13905. **comment:** XXX: option to fix opcode length so it lines up nicely
 label: code-design
13906. %p
13907. after this, return paths should 'goto finished' for decrement
13908. %lx
13909. * Notation: double underscore used for internal properties which are not * stored in the property allocation (e.g. '__valstack').
13910. %f and %lf both consume a 'long'
13911. %x; only 16 bits are guaranteed
13912. maximum length of standard format tag that we support
13913. **comment:** loop_stack_index could be perhaps be replaced by 'depth', but it's nice * to not couple these two mechanisms unnecessarily.
 label: code-design
13914. leave out trailing 'unused' elements
13915. depth check is done when printing an actual type
13916. IEEE double is approximately 16 decimal digits; print a couple extra
13917. ignore
13918. char format: use int
13919. maximum recursion depth for loop detection stacks
13920. unsupported: would consume multiple args
13921. currently implicitly also DUK_USE_DOUBLE_LINKED_HEAP
13922. DUK_USE_DEBUG
13923. Should not happen.
13924. helpers
13925. %s
13926. '%p' and NULL is portable, but special case it * anyway to get a standard NULL marker in logs.
13927. list of conversion specifiers that terminate a format tag; * this is unfortunately guesswork.
13928. **comment:** unused
 label: code-design
13929. must match bytecode defines now; build autogenerate?
13930. %lu
13931. Note: string is a terminal heap object, so no depth check here
13932. two special escapes: '\ and "", other printables as is
13933. from curr pc
13934. **comment:** Formatting function pointers is tricky: there is no standard pointer for * function pointers and the size of a function pointer may depend on the * specific pointer type. This helper formats a function pointer based on * its memory layout to get something useful on most platforms.
 label: code-design
13935. %c', passed concretely as int
13936. **comment:** format is too large, abort
 label: code-design
13937. Quite approximate but should be useful for little and big endian.
13938. %d; only 16 bits are guaranteed
13939. %s' and NULL is not portable, so special case * it for debug printing.
13940. **comment:** prototype should be last, for readability
 label: code-design
13941. fall through
13942. **comment:** dump all allocated entries, unused entries print as 'unused', * note that these may extend beyond current 'length' and look * a bit funny.
 label: code-design
13943. assume exactly 1 arg, which is why '*' is forbidden; arg size still * depends on type though.
13944. %u; only 16 bits are guaranteed
13945. heapobj recursion depth when deep printing is selected
13946. * Format tag parsing. Since we don't understand all the * possible format tags allowed, we just scan for a terminating * specifier and keep track of relevant modifiers that we do * understand. See man 3 printf.
13947. from next pc
13948. if property key begins with underscore, encode it with * forced quotes (e.g. "_Foo") to distinguish it from encoded * internal properties (e.g. \xffBar -> _Bar).

13949. **comment:** XXX: limit to quoted strings only, to save keys from being cluttered?
label: code-design

13950. return total chars written excluding terminator

13951. own pointer

13952. throw_flag

13953. nargs == 2 so we can pass a callstack level to eval().

13954. Write unsigned 32-bit integer.

13955. Use result value as is.

13956. PC/line semantics here are: * - For callstack top we're conceptually between two * opcodes and current PC indicates next line to * execute, so report that (matches Status). * - For other activations we're conceptually still * executing the instruction at PC-1, so report that * (matches error stacktrace behavior). * - See: <https://github.com/svaarala/duktape/issues/281>

13957. * Debug connection peek and flush primitives

13958. Reply with tvals pushed by request callback

13959. Request callback should push values for reply to client onto valstack

13960. * DumpHeap command

13961. **comment:** NULL not needed here
label: code-design

13962. **comment:** XXX: this should be optimized to be a raw query and avoid valstack * operations if possible.
label: code-design

13963. push to stack

13964. Report next pc/line to be executed.

13965. DUK_USE_DEBUGGER_DUMPHEAP || DUK_USE_DEBUGGER_INSPECT

13966. tolerates NULL h_buf

13967. The actual detached_cb call is postponed to message loop so * we don't need any special precautions here (just skip to EOM * on the already closed connection).

13968. Avoid doing an actual write callback with length == 0, * because that's reserved for a write flush.

13969. As with all inspection code, we rely on the debug client providing * a valid, non-stale pointer: there's no portable way to safely * validate the pointer here.

13970. shared pop

13971. **comment:** unused
label: code-design

13972. [... func varmap enum key value this]

13973. [... result]

13974. side effects

13975. * Duktape debugger

13976. * Halt execution helper

13977. **comment:** XXX: several nice-to-have improvements here: * - Use direct reads avoiding value stack operations * - Avoid triggering getters, indicate getter values to debug client * - If side effects are possible, add error catching
label: code-design

13978. ptr may be NULL

13979. save stack top

13980. * NYF <int: 5> <int: fatal> <str: msg> <str: filename> <int: linenumber> EOM

13981. DUK_USE_DEBUGGER_INSPECT

13982. **comment:** Called on a read/write error: NULL all callbacks except the detached * callback so that we never accidentally call them after a read/write * error has been indicated. This is especially important for the transport * I/O callbacks to fulfill guaranteed callback semantics.
label: code-design

13983. flags

13984. For strings, special form for short lengths.

13985. XXX: Decrementing and restoring act->curr_pc works now, but if the * debugger message loop gains the ability to adjust the current PC * (e.g. a forced jump) restoring the PC here will break. Another * approach would be to use a state flag for the "decrement 1 from * topmost activation's PC" and take it into account whenever dealing * with PC values.

13986. unsigned

13987. Read tvals from the message and push them onto the valstack, * then call the request callback to process the request.

13988. **comment:** XXX: comes out as signed now
label: code-design

13989. Report thrown value to client coerced to string

13990. heap->dbg_udata: keep

13991. **comment:** Process debug messages until we are no longer paused.
label: code-design

13992. Caller must trigger recomputation of active breakpoint list. To * ensure stale values are not used if that doesn't happen, clear the * active breakpoint list here.

13993. switch initial byte

13994. may be NULL

13995. For anything other than an Error instance, we calculate the error * location directly from the current activation.

13996. Note: array dump will include elements beyond * 'length'.

13997. Halt execution and enter a debugger message loop until execution is resumed * by the client. PC for the current activation may be temporarily decremented * so that the "current" instruction will be shown by the client. This helper * is callable from anywhere, also outside bytecode executor.

13998. **comment:** SCANBUILD: warning about 'thr' potentially being NULL here, * warning is incorrect because thr != NULL always here.
label: code-design

13999. **comment:** XXX: differentiate null pointer from empty string?
label: code-design

14000. Native function pointer may be different from a void pointer, * and we serialize it from memory directly now (no byte swapping etc).

14001. To use the shared helper need the virtual index.

14002. Breakpoint entries above the used area are left as garbage.

14003. Skip fully.

14004. Skip dvalue.

14005. Note: peek cannot currently trigger a detach * so the dbg_detaching == 0 assert outside the * loop is correct.

14006. **comment:** prevent multiple in-progress detaches
label: requirement

14007. DUK_USE_DEBUGGER_THROW_NOTIFY

14008. processed one or more messages

14009. index

14010. no_block

14011. * Simple commands

14012. Safe to call multiple times.

14013. **comment:** Can be called multiple times with no harm. Mark the transport * bad (dbg_read_cb == NULL) and clear state except for the detached * callback and the udata field. The detached callback is delayed * to the message loop so that it can be called between messages; * this avoids corner cases related to immediate debugger reattach * inside the detached callback.
label: code-design

14014. The debugger protocol doesn't support a plain integer encoding for * the full 32-bit unsigned range (only 32-bit signed). For now, * unsigned 32-bit values simply written as signed ones. This is not * a concrete issue except for 32-bit heaphdr fields. Proper solutions * would be to (a) write such integers as IEEE doubles or (b)

add an * unsigned 32-bit dvalue.

14015. tentative, checked later

14016. Write signed 32-bit integer.

14017. Peek ahead in the stream one byte.

14018. * Breakpoint management

14019. **comment:** Numbers are normalized to big (network) endian. We can * (but are not required) to use integer dvalues when there's * no loss of precision. * * XXX: share check with other code; this check is slow but * reliable and doesn't require careful exponent/mantissa * mask tricks as in the fastint downgrade code.
label: code-design

14020. Decrement PC if that was requested, this requires a PC sync.

14021. **comment:** For short strings, use a fixed temp buffer.
label: code-design

14022. restore stack top

14023. Pretend like we got EOM

14024. always push some string

14025. * Detach handling

14026. duk_hnativefunction specific fields.

14027. [... eval "eval" eval_input level]

14028. Process one debug message. Automatically restore value stack top to its * entry value, so that individual message handlers don't need exact value * stack handling which is convenient.

14029. For now shared handler is fine.

14030. Signed integers always map to 4 bytes now.

14031. No debugger support.

14032. * Debug connection message write helpers

14033. terminator

14034. enum_flags

14035. this cannot initiate a detach

14036. true even with reattach

14037. XXX: Current putvar implementation doesn't have a success flag, * add one and send to debug client?

14038. keep heap->dbg_detached_cb

14039. num_stack_args

14040. **comment:** It is important not to call this if the last byte read was an EOM. * Reading ahead in this scenario would cause unnecessary blocking if * another message is not available.
label: code-design

14041. We're conceptually between two opcodes; act->pc indicates the next * instruction to be executed. This is usually the correct pc/line to * indicate in Status. (For the 'debugger' statement this now reports * the pc/line after the debugger statement because the debugger opcode * has already been executed.)

14042. Return 1: got EOM

14043. heap->dbg_processing: keep on purpose to avoid debugger re-entry in detaching state

14044. NOTE: length may be zero

14045. Read fully.

14046. No activation, no variable access. Could also pretend * we're in the global program context and read stuff off * the global object.

14047. get_value

14048. * Process incoming debug requests * * Individual request handlers can push temporaries on the value stack and * rely on duk_debug_process_message() to restore the value stack top * automatically.

14049. even after a detach and possible reattach

14050. avoid calling write callback in detach1()

14051. **comment:** XXX: Not needed for now, so not implemented. Note that * function pointers may have different size/layout than * a void pointer.
label: requirement

14052. **comment:** XXX: optimize to use direct reads, i.e. avoid * value stack operations.
label: code-design

14053. pruned

14054. switch cmd

14055. NOTE: act may be NULL if an error is thrown outside of any activation, * which may happen in the case of, e.g. syntax errors.

14056. Careful here: state must be wiped before the call * so that we can cleanly handle a re-attach from * inside the callback.

14057. **comment:** Ensure there are no stale active breakpoint pointers. * Breakpoint list is currently kept - we could empty it * here but we'd need to handle refcounts correctly, and * we'd need a 'thr' reference for that. * * XXX: clear breakpoint on either attach or detach?
label: code-design

14058. Short circuit if is safe: if act->curr_pc != NULL, 'fun' is * guaranteed to be a non-NULL Ecmascript function.

14059. **comment:** unused: not accepted in inbound messages
label: code-design

14060. Allow NULL 'msg'

14061. * Object inspection commands: GetHeapObjInfo, GetObjPropDesc, * GetObjPropDescRange

14062. detached

14063. XXX: Currently no inspection of threads, e.g. value stack, call * stack, catch stack, etc.

14064. * Debug connection skip primitives

14065. isAccessor

14066. * Helper structs

14067. At least: ... [err]

14068. Error instance, use augmented error data directly

14069. just in case callback is broken and won't write 'x'

14070. Accept any pointer-like value; for 'object' dvalue, read * and ignore the class number.

14071. DUK_USE_DEBUGGER_DUMPHEAP

14072. -> [... func varmap enum]

14073. duk_hobject specific fields.

14074. The index range space is conceptually the array part followed by the * entry part. Unlike normal enumeration all slots are exposed here as * is and return 'unused' if the slots are not in active use. In particular * the array part is included for the full a_size regardless of what the * array .length is.

14075. enum_index

14076. this is especially critical to avoid another write call in detach1()

14077. * Debug connection write primitives

14078. For errors a string coerced result is most informative * right now, as the debug client doesn't have the capability * to traverse the error object.

14079. As an initial implementation, write flush after every EOM (and the * version identifier). A better implementation would flush only when * Duktape is finished processing messages so that a flush only happens * after all outbound messages are finished on that occasion.

14080. Inspect a property using a virtual index into a conceptual property list * consisting of (1) all array part items from [0,a_size[(even when above * .length) and (2) all entry part items from [0,e_next[. Unused slots are * indicated using dvalue 'unused'.

14081. Use b[] to access the size of the union, which is strictly not * correct. Can't use fixed size unless there's feature detection * for pointer byte size.

14082. **comment:** XXX: this has some overlap with object inspection; remove this and make * DumpHeap return lists of heapptrs instead?
label: code-design

14083. Detach is pending; can be triggered from outside the * debugger loop (e.g. Status notify write error) or by * previous message handling. Call detached callback * here, in a controlled state, to ensure a possible * reattach inside the detached_cb is handled correctly. * * Recheck for detach in a while loop: an immediate *

reattach involves a call to duk_debugger_attach() * which writes a debugger handshake line immediately * inside the API call. If the transport write fails * for that handshake, we can immediately end up in a * "transport broken, detaching" case several times here. * Loop back until we're either cleanly attached or * fully detached. ** NOTE: Reset dbg_processing = 1 forcibly, in case we * re-attached; duk_debugger_attach() sets dbg_processing * to 0 at the moment.

14084. The eval code is executed within the lexical environment of a specified * activation. For now, use global object eval() function, with the eval * considered a 'direct call to eval'. ** Callstack level for debug commands only affects scope -- the callstack * as seen by, e.g. Duktape.act() will be the same regardless.

14085. not an error

14086. As an initial implementation, read flush after exiting the message * loop. If transport is broken, this is a no-op (with debug logs).

14087. **comment:** NOTE: This is a bit fragile. It's important to ensure that * duk_debug_process_messages() never throws an error or * act->curr_pc will never be reset.
label: code-design

14088. thr->heap->dbg_detaching may be != 0 if a debugger write outside * the message loop caused a transport error and detach1() to run.

14089. DUK_USE_STRTAB_PROBE

14090. Process messages until we're no longer paused or we peek * and see there's nothing to read right now.

14091. may be set to 0 by duk_debugger_attach() inside callback

14092. Variant for writing duk_tvals so that any heap allocated values are * written out as tagged heap pointers.

14093. DUK_USE_DEBUGGER_SUPPORT

14094. Easy to get wrong, so assert for it.

14095. optional callstack level

14096. restore PC

14097. w/o refcounting

14098. Write fully.

14099. * Debug message processing

14100. * Debug connection read primitives

14101. compensate for eval() call

14102. Since we already started writing the reply, just emit nothing.

14103. DUK_USE_STRTAB_CHAIN

14104. Skip dvalues to EOM.

14105. heap->dbg_detached_cb: keep

14106. nargs

14107. 'this' binding shouldn't matter here

14108. * Status message and helpers

14109. Ensure dirty state causes a Status even if never process any * messages. This is expected by the bytecode executor when in * the running state.

14110. **comment:** Direct eval requires that there's a current * activation and it is an Ecmascript function. * When Eval is executed from e.g. cooperate API * call we'll need to do an indirect eval instead.
label: code-design

14111. **comment:** XXX: exposed duk_debug_read_pointer
label: code-design

14112. Error codes.

14113. **comment:** unused
label: code-design

14114. Initial bytes for markers.

14115. Commands initiated by debug client.

14116. Commands and notifys initiated by Duktape.

14117. **comment:** XXX: exposed duk_debug_read_hbuffer
label: code-design

14118. Debugger protocol version is defined in the public API header.

14119. The short string/integer initial bytes starting from 0x60 don't have * defines now.

14120. The low 8 bits map directly to duk_hobject.h DUK_PROPDESC_FLAG_xxx. * The remaining flags are specific to the debugger.

14121. DUK_DEBUGGER_H_INCLUDED

14122. **comment:** XXX: exposed duk_debug_read_buffer
label: code-design

14123. Other initial bytes.

14124. **comment:** * Error handling macros, assertion macro, error codes. ** There are three level of 'errors': * * 1. Ordinary errors, relative to a thread, cause a longjmp, catchable. * 2. Fatal errors, relative to a heap, cause fatal handler to be called. * 3. Panic errors, unrelated to a heap and cause a process exit. ** Panics are used by the default fatal error handler and by debug code * such as assertions. By providing a proper fatal error handler, user * code can avoid panics in non-debug builds.
label: code-design

14125. DUK_ERR_URI_ERROR: no macros needed

14126. this is added to checks to allow for Duktape * API calls in addition to function's own use

14127. assertion omitted

14128. DUK_USE_PANIC_HANDLER

14129. * Assertion helpers

14130. this variant is used when an assert would generate a compile warning by * being always true (e.g. >= 0 comparison for an unsigned value

14131. no refcount check

14132. * Fatal error * * There are no fatal error macros at the moment. There are so few call * sites that the fatal error handler is called directly.

14133. DUK_ERR_ERROR: no macros needed

14134. Verbose errors with key/value summaries (non-paranoid) or without key/value * summaries (paranoid, for some security sensitive environments), the paranoid * vs. non-paranoid distinction affects only a few specific errors.

14135. DUK_USE_ASSERTIONS

14136. **comment:** * Panic error * * Panic errors are not relative to either a heap or a thread, and cause * DUK_PANIC() macro to be invoked. Unless a user provides DUK_USE_PANIC_HANDLER, * DUK_PANIC() calls a helper which prints out the error and causes a process * exit. ** The user can override the macro to provide custom handling. A macro is * used to allow the user to have inline panic handling if desired (without * causing a potentially risky function call). ** Panics are only used in debug code such as assertions, and by the default * fatal error handler.
label: code-design

14137. * Prototypes

14138. nop

14139. already defined, good

14140. DUK_ERR_ASSERTION_ERROR: no macros needed

14141. **comment:** * Error throwing helpers * * The goal is to provide verbose and configurable error messages. Call * sites should be clean in source code and compile to a small footprint. * Small footprint is also useful for performance because small cold paths * reduce code cache pressure. Adding macros here only makes sense if there * are enough call sites to get concrete benefits.
label: code-design

14142. **comment:** Because there are quite many call sites, pack error code (require at most * 8-bit) into a single argument.
label: code-design

14143. * Assert macro: failure causes panic.

14144. DUK_ERROR_H_INCLUDED

14145. DUK_USE_VERBOSE_ERRORS

14146. DUK_ERR_REFERENCE_ERROR: no macros needed

14147. **comment:** * Error codes: defined in duktape.h * * Error codes are used as a shorthand to throw exceptions from inside * the implementation. The appropriate Ecmascript object is constructed * based on the code. Ecmascript code throws objects directly. The error * codes are defined in the public API header because they

are also used * by calling code.

label: code-design

14148. **comment:** XXX: resolve macro definition issue or call through a helper function?

label: code-design

14149. Non-verbose errors for low memory targets: no file, line, or message.

14150. DUK_VERBOSE_ERRORS

14151. **comment:** the message should be a compile time constant without formatting (less risk); * we don't care about assertion text size because they're not used in production * builds.

label: code-design

14152. no valstack space check

14153. Assertion compatible inside a comma expression, evaluates to void. * Currently not compatible with DUK_USE_PANIC_HANDLER() which may have * a statement block.

14154. DUK_ERR_UNCAUGHT_ERROR: no macros needed

14155. assertion disabled

14156. DUK_USE_PARANOI_ERRORS

14157. DUK_ERR_EVAL: no macros needed

14158. **comment:** * Normal error * * Normal error is thrown with a longjmp() through the current setjmp() * catchpoint record in the duk_heap. The 'curr_thread' of the duk_heap * identifies the throwing thread. * * Error formatting is usually unnecessary. The error macros provide a * zero argument version (no formatting) and separate macros for small * argument counts. Variadic macros are not used to avoid portability * issues and avoid the need for stash-based workarounds when they're not * available. Vararg calls are avoided for non-formatted error calls * because vararg call sites are larger than normal, and there are a lot * of call sites with no formatting. * * Note that special formatting provided by debug macros is NOT available. * * The _RAW variants allow the caller to specify file and line. This makes * it easier to write checked calls which want to use the call site of the * checked function, not the error macro call inside the checked function.

label: code-design

14159. * Helper for valstack space * * Caller of DUK_ASSERT_VALSTACK_SPACE() estimates the number of free stack entries * required for its own use, and any child calls which are not (a) Duktape API calls * or (b) Duktape calls which involve extending the valstack (e.g. getter call).

14160. **comment:** * Augment an error at creation time with _Tracedata/fileName/lineNumber * and allow a user error handler (if defined) to process/replace the error. * The error to be augmented is at the stack top. * * thr: thread containing the error value * thr_callstack: thread which should be used for generating callstack etc. * c_filename: C __FILE__ related to the error * c_line: C __LINE__ related to the error * noblame_fileline: if true, don't fileName/line as error source, otherwise use traceback * (needed because user code filename/line are reported but internal ones * are not) * * XXX: rename noblame_fileline to flags field; combine it to some existing * field (there are only a few call sites so this may not be worth it).

label: code-design

14161. unsigned

14162. **comment:** * Augmenting errors at their creation site and their throw site. * * When errors are created, traceback data is added by built-in code * and a user error handler (if defined) can process or replace the * error. Similarly, when errors are thrown, a user error handler * (if defined) can process or replace the error. * * Augmentation and other processing at error creation time is nice * because an error is only created once, but it may be thrown and * rethrown multiple times. User error handler registered for processing * an error at its throw site must be careful to handle rethrowing in * a useful manner. * * Error augmentation may throw an internal error (e.g. alloc error). * * Ecmascript allows throwing any values, so all values cannot be * augmented. Currently, the built-in augmentation at error creation * only augments error values which are Error instances (= have the * built-in Error.prototype in their prototype chain) and are also * extensible. User error handlers have no limitations in this respect.

label: code-design

14163. [...] errval]

14164. 3 entries actually needed below

14165. no need to check now: both success and error are OK

14166. **comment:** XXX: optimize: allocate an array part to the necessary size (upwards * estimate) and fill in the values directly into the array part; finally * update 'length'.

label: code-design

14167. assume PC is at most 32 bits and non-negative

14168. No activation matches, use undefined for both .fileName and * .lineNumber (matches what we do with a _Tracedata based * no-match lookup).

14169. Append a "(line NNN)" to the "message" property of any error * thrown during compilation. Usually compilation errors are * SyntaxErrors but they can also be out-of-memory errors and * the like.

14170. DUK_USE_ERRTHROW

14171. num args

14172. If tracebacks are enabled, the '_Tracedata' property is the only * thing we need: 'fileName' and 'lineNumber' are virtual properties * which use '_Tracedata'.

14173. Lightfunc, not blamed now.

14174. * If tracebacks are disabled, 'fileName' and 'lineNumber' are added * as plain own properties. Since Error.prototype has accessors of * the same name, we need to define own properties directly (cannot * just use e.g. duk_put_prop_stridx). Existing properties are not * overwritten in case they already exist.

14175. -> [...] error]

14176. thr argument only used for thr->heap, so specific thread doesn't matter

14177. traceback depth doesn't take into account the filename/line * special handling above (intentional)

14178. -> [...] errhandler errval]

14179. DUK_USE_ERRTHROW || DUK_USE_ERRCREATE

14180. * DUK_CALL_FLAG_IGNORE_RECLIMIT causes duk_handle_call() to ignore C * recursion depth limit (and won't increase it either). This is * dangerous, but useful because it allows the error handler to run * even if the original error is caused by C recursion depth limit. * * The heap level DUK_HEAP_FLAG_ERRHANDLER_RUNNING is set for the * duration of the error handler and cleared afterwards. This flag * prevents the error handler from running recursively. The flag is * heap level so that the flag properly controls even coroutines * launched by an error handler. Since the flag is heap level, it is * critical to restore it correctly. * * We ignore errors now: a success return and an error value both * replace the original error value. (This would be easy to change.)

14181. Finally, blame the innermost callstack entry which has a * .fileName property.

14182. [...] error]

14183. ignore reclimit, not constructor

14184. Native function, no relevant lineNumber.

14185. !DUK_USE_TRACEBACKS

14186. Without tracebacks the concrete .fileName and .lineNumber need * to be added directly.

14187. PC points to next instruction, find offending PC, * PC == 0 for native code.

14188. [...] error func]

14189. ignore_loop

14190. callstack limits

14191. When creating built-ins, some of the built-ins may not be set * and we want to tolerate that when throwing errors.

14192. * The traceback format is pretty arcane in an attempt to keep it compact * and cheap to create. It may change arbitrarily from version to version. * It should be decoded/accessible through version specific accessors only. * * See doc/error-objects.rst.

14193. [...] error lineNumber fileName]

14194. * Add line number to a compiler error.

14195. DUK_USE_AUGMENT_ERROR_CREATE

14196. * Add .fileName and .lineNumber to an error on the stack top.

14197. * Criteria for augmenting: * * - augmentation enabled in build (naturally) * - error value internal prototype chain contains the built-in * Error prototype object (i.e. 'val instanceof Error') * * Additional criteria for built-in augmenting: * * - error value is an extensible object

14198. since no recursive error handler calls

14199. DUK_USE_PC2LINE

14200. **comment:** * Check whether or not we have an error handler. ** We must be careful of not triggering an error when looking up the * property. For instance, if the property is a getter, we don't want * to call it, only plain values are allowed. The value, if it exists, * is not checked. If the value is not a function, a TypeError happens * when it is called and that error replaces the original one.

label: code-design

14201. invalidated by pushes, so get out of the way

14202. Compiler SyntaxError (or other error) gets the primary blame. * Currently no flag to prevent blaming.

14203. [... errhandler undefined errval]

14204. [... errval errhandler]

14205. -> [... arr num]

14206. [... error func fileName]

14207. DUK_USE_TRACEBACKS

14208. If the value has a prototype loop, it's critical not to * throw here. Instead, assume the value is not to be * augmented.

14209. DUK_USE_AUGMENT_ERROR_THROW

14210. [... error arr]

14211. [... error func fileName lineNumber]

14212. call_flags

14213. -> [... errhandler undefined(= this) errval]

14214. Compiler SyntaxErrors (and other errors) come first, and are * blamed by default (not flagged "noblame").

14215. [... arr]

14216. Add function object.

14217. **comment:** * Helper for calling a user error handler. ** 'thr' must be the currently active thread; the error handler is called * in its context. The valstack of 'thr' must have the error value on * top, and will be replaced by another error value based on the return * value of the error handler. ** The helper calls duk_handle_call() recursively in protected mode. * Before that call happens, no longjmps should happen; as a consequence, * we must assume that the valstack contains enough temporary space for * arguments and such. * While the error handler runs, any errors thrown will not trigger a * recursive error handler call (this is implemented using a heap level * flag which will "follow" through any coroutines resumed inside the * error handler). If the error handler is not callable or throws an * error, the resulting error replaces the original error (for Duktape * internal errors, duk_error_throw.c further substitutes this error with * a DoubleError which is not ideal). This would be easy to change and * even signal to the caller. * The user error handler is stored in 'Duktape.errCreate' or * 'Duktape.errThrow' depending on whether we're augmenting the error at * creation or throw time. There are several alternatives to this approach, * see doc/error-objects.rst for discussion. ** Note: since further longjmp(s) may occur while calling the error handler * (for many reasons, e.g. a labeled 'break' inside the handler), the * caller can make no assumptions on the thr->heap->lj state after the * call (this affects especially duk_error_throw.c). This is not an issue * as long as the caller writes to the lj state only after the error handler * finishes.

label: code-design

14218. * Add ._Tracedata to an error on the stack top.

14219. XXX: specify array size, as we know it

14220. (flags<<32) + (line), flags = 0

14221. * Augment an error being created using Duktape specific properties * like _Tracedata or .fileName/.lineNumber.

14222. * Augment an error at throw time; allow a user error handler (if defined) * to process/replace the error. The error to be augmented is at the * stack top.

14223. unreferenced w/o tracebacks

14224. Filename/line from C macros (_FILE_, _LINE_) are added as an * entry with a special format: (string, number). The number contains * the line and flags.

14225. **comment:** * Note: each API operation potentially resizes the callstack, * so be careful to re-lookup after every operation. Currently * these is no issue because we don't store a temporary 'act' * pointer at all. (This would be a non-issue if we operated * directly on the array part.)

label: code-design

14226. XXX: set with duk_hobject_set_length() when tracedata is filled directly

14227. unreferenced w/o asserts

14228. **comment:** XXX: using duk_put_prop_index() would cause obscure error cases when Array.prototype * has write-protected array index named properties. This was seen as DoubleErrors * in e.g. some test262 test cases. Using duk_xdef_prop_index() is better but heavier. * The best fix is to fill in the tracedata directly into the array part. There are * no side effect concerns if the array part is allocated directly and only INCREFS * happen after that.

label: code-design

14229. Add a number containing: pc, activation flags. ** PC points to next instruction, find offending PC. Note that * PC == 0 for native code.

14230. C call site gets blamed next, unless flagged not to do so. * XXX: file/line is disabled in minimal builds, so disable this * too when appropriate.

14231. **comment:** If we don't have a jmpbuf_ptr, there is little we can do * except panic. The caller's expectation is that we never * return. ** With C++ exceptions we now just propagate an uncaught error * instead of invoking the fatal error handler. Because there's * a dummy jmpbuf for C++ exceptions now, this could be changed.

label: code-design

14232. DUK_USE_CPP_EXCEPTIONS

14233. dummy

14234. * Do a longjmp call, calling the fatal error handler if no * catchpoint exists.

14235. * Error throwing helpers

14236. The file/line arguments are NULL and 0, they're ignored by DUK_ERROR_RAW() * when non-verbose errors are used.

14237. DUK_USE_VERBOSE_ERRORS

14238. **comment:** size for formatting buffers

label: code-design

14239. * Default fatal error handler

14240. !DUK_USE_PANIC_HANDLER

14241. omit print

14242. **comment:** dead code, but ensures portability (see Linux man page notes)

label: code-design

14243. SCANBUILD: "Dereference of null pointer", normal

14244. * Default panic handler

14245. * Error, fatal, and panic handling.

14246. exit() afterwards to satisfy "noreturn"

14247. DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT

14248. Don't intercept a DoubleError, we may have caused the initial double * fault and attempting to intercept it will cause us to be called * recursively and exhaust the C stack.

14249. Report it to the debug client

14250. * Helper to walk the thread chain and see if there is an active error * catcher. Protected calls or finally blocks aren't considered catching.

14251. * Get prototype object for an integer error code.

14252. **comment:** XXX: more specific error classes?

label: code-design

14253. DUK_USE_DEBUGGER_SUPPORT

14254. side effects

14255. XXX: Allow customizing the pause and notify behavior at runtime * using debugger runtime flags. For now the behavior is fixed using * config options.

14256. all we need to know

14257. **comment:** * XXX: As noted above, a protected API call won't be counted as a * catcher. This is usually convenient, e.g. in the case of a top- * level duk_pcall(), but may not always be desirable. Perhaps add an * argument to treat them as catchers?

label: code-design

14258. * Error helpers

14259. * Exposed helper for setting up heap longjmp state.

14260. use_prev_pc
14261. DUK_USE_DEBUGGER_SUPPORT && (DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT)
14262. If something is thrown with the debugger attached and nobody will * catch it, execution is paused before the longjmp, turning over * control to the debug client. This allows local state to be examined * before the stack is unwound. Errors are not intercepted when debug * message loop is active (e.g. for Eval).
14263. **comment:** * The __FILE__ and __LINE__ information is intentionally not used in the * creation of the error object, as it isn't useful in the tracedata. The * tracedata still contains the function which returned the negative return * code, and having the file/line of this function isn't very useful.
 label: code-design
14264. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
14265. **comment:** * Create and throw an error (originating from Duktape internally) * * Push an error object on top of the stack, possibly throw augmenting * the error, and finally longjmp. * * If an error occurs while we're dealing with the current error, we might * enter an infinite recursion loop. This is prevented by detecting a * "double fault" through the heap->handling_error flag; the recursion * then stops at the second level.
 label: code-design
14266. * Create and throw an Ecmascript error object based on a code and a message. * * Used when we throw errors internally. Ecmascript generated error objects * are created by Ecmascript code, and the throwing is handled by the bytecode * executor.
14267. **comment:** XXX: if attempt to push beyond allocated valstack, this double fault * handling fails miserably. We should really write the double error * directly to thr->heap->lj.value1 and avoid valstack use entirely.
 label: code-design
14268. * Create and push an error object onto the top of stack. * If a "double error" occurs, use a fixed error instance * to avoid further trouble.
14269. **comment:** XXX: unnecessary '%s' formatting here, but cannot use * 'msg' as a format string directly.
 label: code-design
14270. reset callstack limit
14271. **comment:** XXX: this generates quite large code - perhaps select the error * class based on the code and then just use the error 'name'?
 label: code-design
14272. just making sure
14273. * Finally, longjmp
14274. **comment:** Allow headroom for calls during error handling (see GH-191). * We allow space for 10 additional recursions, with one extra * for, e.g. a print() call at the deepest level.
 label: code-design
14275. **comment:** * Augment error (throw time), unless alloc/double error
 label: code-design
14276. **comment:** XXX: shared strings
 label: code-design
14277. Error object is augmented at its creation here.
14278. * Helper for C function call negative return values.
14279. * Exception for Duktape internal throws when C++ exceptions are used * for long control transfers. * * Doesn't inherit from any exception base class to minimize the chance * that user code would accidentally catch this exception.
14280. intentionally empty
14281. DUK_EXCEPTION_H_INCLUDED
14282. duk_tval intentionally skipped
14283. * Forward declarations
14284. * Forward declarations for all Duktape structures.
14285. DUK_FORWDECL_H_INCLUDED
14286. no typedef
14287. Current size (not counting a dynamic buffer's "spare").
14288. External buffer with 'curr_alloc' managed by user code and pointing to an * arbitrary address. When heap pointer compression is not used, this struct * has the same layout as duk_hbuffer_dynamic.
14289. No pointer compression because pointer is potentially outside of * Duktape heap.
14290. **comment:** It's not strictly necessary to track the current size, but * it is useful for writing robust native code.
 label: code-design
14291. Stored in duk_heaphdr h_extra16.
14292. * Flags * * Fixed buffer: 0 * Dynamic buffer: DUK_HBUFFER_FLAG_DYNAMIC * External buffer: DUK_HBUFFER_FLAG_DYNAMIC | DUK_HBUFFER_FLAG_EXTERNAL
14293. * Data following the header depends on the DUK_HBUFFER_FLAG_DYNAMIC * flag. * * If the flag is clear (the buffer is a fixed size one), the buffer * data follows the header directly, consisting of 'size' bytes. * * If the flag is set, the actual buffer is allocated separately, and * a few control fields follow the header. Specifically: * * - a "void *" pointing to the current allocation * - a duk_size_t indicating the full allocated size (always >= 'size') * * If DUK_HBUFFER_FLAG_EXTERNAL is set, the buffer has been allocated * by user code, so that Duktape won't be able to resize it and won't * free it. This allows buffers to point to e.g. an externally * allocated structure such as a frame buffer. * * Unlike strings, no terminator byte (NUL) is guaranteed after the * data. This would be convenient, but would pad aligned user buffers * unnecessarily upwards in size. For instance, if user code requested * a 64-byte dynamic buffer, 65 bytes would actually be allocated which * would then potentially round upwards to perhaps 68 or 72 bytes.
14294. **comment:** * Data follows the struct header. The struct size is padded by the * compiler based on the struct members. This guarantees that the * buffer data will be aligned-by-4 but not necessarily aligned-by-8. * * On platforms where alignment does not matter, the struct padding * could be removed (if there is any). On platforms where alignment * by 8 is required, the struct size must be forced to be a multiple * of 8 by some means. Without it, some user code may break, and also * Duktape itself breaks (e.g. the compiler stores duk_tvals in a * dynamic buffer).
 label: code-design
14295. size stored in duk_heaphdr unused flag bits
14296. * Misc defines
14297. indirect allocs
14298. may be NULL if alloc_size == 0
14299. * Prototypes
14300. **comment:** * Allocation size for 'curr_alloc' is alloc_size. There is no * automatic NUL terminator for buffers (see above for rationale). * * 'curr_alloc' is explicitly allocated with heap allocation * primitives and will thus always have alignment suitable for * e.g. duk_tval and an IEEE double.
 label: code-design
14301. Dynamic buffer with 'curr_alloc' pointing to a dynamic area allocated using * heap allocation primitives. Also used for external buffers when low memory * options are not used.
14302. * Heap buffer representation. * * Heap allocated user data buffer which is either: * * 1. A fixed size buffer (data follows header statically) * 2. A dynamic size buffer (data pointer follows header) * * The data pointer for a variable size buffer of zero size may be NULL.
14303. Get/set the current user visible size, without accounting for a dynamic * buffer's "spare" (= usable size).
14304. Intentionally not 0x7fffffffUL; at least JSON code expects that * 2*len + 2 fits in 32 bits.
14305. buffer pointer is to an externally allocated buffer
14306. assume 0 <=> NULL
14307. dynamic buffer ops
14308. DUK_HBUFFER_H_INCLUDED
14309. **comment:** A union is used here as a portable struct size / alignment trick: * by adding a 32-bit or a 64-bit (unused) union member, the size of * the struct is effectively forced to be a multiple of 4 or 8 bytes * (respectively) without increasing the size of the struct unless * necessary.
 label: code-design
14310. no extra padding
14311. * Field access
14312. Cannot be compressed as a heap pointer because may point to * an arbitrary address.

14313. Without heap pointer compression `duk_hbuffer_dynamic` and `duk_hbuffer_external *` have the same layout so checking for fixed vs. dynamic (or external) is enough.
14314. * Structs
14315. Get a pointer to the current buffer contents (matching current allocation * size). May be NULL for zero size dynamic/external buffer.
14316. Stored in `duk_heapheader` unused flags.
14317. Shared prefix for all buffer types.
14318. buffer is behind a pointer, dynamic or external
14319. Fixed buffer; data follows struct, with proper alignment guaranteed by * struct size.
14320. Impose a maximum buffer length for now. Restricted artificially to * ensure resize computations or adding a heap header length won't * overflow `size_t` and that a signed `duk_int_t` can hold a buffer * length. The limit should be synchronized with `DUK_HSTRING_MAX_BYTELEN`.
14321. Because size > 0, NULL check is correct
14322. **comment:** Size sanity check. Should not be necessary because caller is * required to check this, but we don't want to cause a segfault * if the size wraps either in `duk_size_t` computation or when * storing the size in a 16-bit field.
- label:** code-design
14323. alloc external with size zero
14324. the compressed pointer is zeroed which maps to NULL, so nothing to do.
14325. For indirect allocs.
14326. no need to write 'out_bufdata'
14327. zero everything unless requested not to do so
14328. * `duk_hbuffer` allocation and freeing.
14329. Allocate a new `duk_hbuffer` of a certain type and return a pointer to it * (NULL on error). Write buffer data pointer to 'out_bufdata' (only if * allocation successful).
14330. no wrapping
14331. **comment:** * `duk_hbuffer` operations such as resizing and inserting/appending data to * a dynamic buffer. * * Append operations append to the end of the buffer and they are relatively * efficient: the buffer is grown with a "spare" part relative to the buffer * size to minimize reallocations. Insert operations need to move existing * data forward in the buffer with `memmove()` and are not very efficient. * They are used e.g. by the regexp compiler to "backpatch" regexp bytecode.
- label:** code-design
14332. * Resizing
14333. 'res' may be NULL if new allocation size is 0.
14334. **comment:** * Note: use indirect realloc variant just in case mark-and-sweep * (finalizers) might resize this same buffer during garbage * collection.
- label:** code-design
14335. * Maximum size check
14336. * The entire allocated buffer area, regardless of actual used * size, is kept zeroed in resizes for simplicity. If the buffer * is grown, zero the new part.
14337. All element accessors are host endian now (driven by `TypedArray` spec).
14338. Validate that the whole slice [0,length[is contained in the underlying * buffer. Caller must ensure 'buf' != NULL.
14339. `DUK_HBUFFEROBJECT_H_INCLUDED`
14340. byte index limit for element access, exclusive
14341. **comment:** Slice and accessor information. * * Because the underlying buffer may be dynamic, these may be * invalidated by the buffer being modified so that both offset * and length should be validated before every access. Behavior * when the underlying buffer has changed doesn't need to be clean: * virtual 'length' doesn't need to be affected, reads can return * zero/NaN, and writes can be ignored. * * Note that a data pointer cannot be precomputed because 'buf' may * be dynamic and its pointer unstable.
- label:** code-design
14342. Validate byte read/write for virtual 'offset', i.e. check that the * offset, taking into account `h->offset`, is within the underlying * buffer size. This is a safety check which is needed to ensure * that even a misconfigured `duk_hbufferobject` never causes memory * unsafe behavior (e.g. if an underlying dynamic buffer changes * after being setup). Caller must ensure 'buf' != NULL.
14343. Shared object part.
14344. element type
14345. Underlying buffer (refcounted), may be NULL.
14346. Get the current data pointer (caller must ensure buf != NULL) as a * `duk_uint8_t` ptr.
14347. True if slice is full, i.e. offset is zero and length covers the entire * buffer. This status may change independently of the `duk_hbufferobject` if * the underlying buffer is dynamic and changes without the `hbufferobject` * being changed.
14348. element size shift: * 0 = u8/i8 * 1 = u16/i16 * 2 = u32/i32/float * 3 = double
14349. No assertions for offset or length; in particular, \ * it's OK for length to be longer than underlying \ * buffer. Just ensure they don't wrap when added. \
14350. byte offset to buf
14351. * Heap Buffer object representation. Used for all Buffer variants.
14352. Clamp an input byte length (already assumed to be within the nominal * `duk_hbufferobject` 'length') to the current dynamic buffer limits to * yield a byte length limit that's safe for memory accesses. This value * can be invalidated by any side effect because it may trigger a user * callback that resizes the underlying buffer.
14353. Slice starting point is beyond current length.
14354. `DUK_USE_BUFFEROBJECT_SUPPORT`
14355. Line number range for function. Needed during debugging to * determine active breakpoints.
14356. number of arguments allocated to regs
14357. * Additional control information is placed into the object itself * as internal properties to avoid unnecessary fields for the * majority of functions. The compiler tries to omit internal * control fields when possible. * * Function templates: * * { * name: "func", // declaration, named function expressions * `fileName`: <debug info for creating nice errors> * `_Varmap`: { "arg1": 0, "arg2": 1, "varname": 2 }, * `_Formals`: ["arg1", "arg2"], * `_Source`: "function func(arg1, arg2) { ... }", * `_Pc2line`: <debug info for pc-to-line mapping>, * } * * Function instances: * * { * length: 2, * prototype: { constructor: <func> }, * caller: <thrower>, * arguments: <thrower>, * name: "func", // declaration, named function expressions * `fileName`: <debug info for creating nice errors> * `_Varmap`: { "arg1": 0, "arg2": 1, "varname": 2 }, * `_Formals`: ["arg1", "arg2"], * `_Source`: "function func(arg1, arg2) { ... }", * `_Pc2line`: <debug info for pc-to-line mapping>, * `_Varenv`: <variable environment of closure>, * `_Lexenv`: <lexical environment of closure (if differs from `_Varenv`)> * } * * More detailed description of these properties can be found * in the documentation.
14358. **comment:** XXX: casts could be improved, especially for GET/SET DATA
- label:** code-design
14359. * Heap compiled function (EcmaScript function) representation. * * There is a single data buffer containing the EcmaScript function's * bytecode, constants, and inner functions.
14360. shared object part
14361. **comment:** * 'nregs' registers are allocated on function entry, at most 'nargs' * are initialized to arguments, and the rest to undefined. Arguments * above 'nregs' are not mapped to registers. All registers in the * active stack range must be initialized because they are GC reachable. * 'nargs' is needed so that if the function is given more than 'nargs' * arguments, the additional arguments do not 'clobber' registers * beyond 'nregs' which must be consistently initialized to undefined. * * Usually there is no need to know which registers are mapped to * local variables. Registers may be allocated to variable in any * way (even including gaps). However, a register-variable mapping * must be the same for the duration of the function execution and * the register cannot be used for anything else. * * When looking up variables by name, the '`_Varmap`' map is used. * When an activation closes, registers mapped to arguments are * copied into the environment record based on the same map. The * reverse map (from register to variable) is not currently needed * at run time, except for debugging, so it is not maintained.
- label:** code-design
14362. * Field accessor macros
14363. `DUK_HCOMPILEDFUNCTION_H_INCLUDED`
14364. regs to allocate
14365. **comment:** No need for constants pointer (= same as data). * * When using 16-bit packing alignment to 4 is nice. 'funcs' will be * 4-byte aligned because 'constants' are `duk_tvvals`. For now the * inner function pointers are not compressed, so that 'bytecode' will * also be 4-byte aligned.
- label:** code-design

14366. Data area, fixed allocation, stable data ptrs.
14367. Note: assumes 'data' is always a fixed buffer
14368. * Main struct
14369. * Pointers to function data area for faster access. Function * data is a buffer shared between all closures of the same * "template" function. The data buffer is always fixed (non- * dynamic, hence stable), with a layout as follows: * * constants (duk_tval) * inner functions (duk_hobject *) * bytecode (duk_instr_t) * * * Note: bytecode end address can be computed from 'data' buffer * size. It is not strictly necessary functionally, assuming * bytecode never jumps outside its allocated area. However, * it's a safety/robustness feature for avoiding the chance of * executing random data as bytecode due to a compiler error. * * Note: values in the data buffer must be incref'd (they will * be decref'd on release) for every compiledfunction referring * to the 'data' element.
14370. * Accessor macros for function specific data areas
14371. **comment:** XXX: double evaluation of DUK_HCOMPILEDFUNCTION_GET_DATA()
 label: code-design
14372. callback for indirect reallocs, request for current pointer
14373. Retry allocation after mark-and-sweep for this * many times. A single mark-and-sweep round is * not guaranteed to free all unreferenced memory * because of finalization (in fact, ANY number of * rounds is strictly not enough).
14374. **comment:** XXX: static alloc is OK until separate chaining stringtable * resizing is implemented.
 label: code-design
14375. heap destruction ongoing, finalizer rescue no longer possible
14376. resend state next time executor is about to run
14377. No field needed when strings are in ROM.
14378. indicates a deleted string; any fixed non-NULL, non-hstring pointer works
14379. **comment:** alloc size in elements
 label: code-design
14380. value1 -> label number, pseudo-type to indicate a break continuation (for ENDFIN)
14381. don't run finalizers; queue finalizable objects back to heap_allocated
14382. **comment:** unused
 label: code-design
14383. DUK_USE_ROM_STRINGS
14384. don't compact objects; needed during object property allocation resize
14385. borrowed; NULL if no step state (NULLED in unwind)
14386. breakpoints: [0,breakpoint_count[gc reachable
14387. value1 -> resume value, value2 -> resume thread, iserror -> error/normal
14388. * Other heap related defines
14389. Allocator functions.
14390. don't run finalizers; leave finalizable objects in finalize_list for next round
14391. Step types
14392. used entries + approx 100% -> reset load to 50%
14393. current thread
14394. * Mark-and-sweep flags * * These are separate from heap level flags now but could be merged. * The heap structure only contains a 'base mark-and-sweep flags' * field and the GC caller can impose further flags.
14395. * Memory constants
14396. force executor restart to recheck breakpoints; used to handle function returns (see GH-303)
14397. required, NULL implies detached
14398. no value, pseudo-type to indicate a normal continuation (for ENDFIN)
14399. heap thread, used internally and for finalization
14400. * Thread switching * * To switch heap->curr_thread, use the macro below so that interrupt counters * get updated correctly. The macro allows a NULL target thread because that * happens e.g. in call handling.
14401. * Longjmp types, also double as identifying continuation type for a rethrow (in 'finally')
14402. * Longjmp state, contains the information needed to perform a longjmp. * Longjmp related values are written to value1, value2, and iserror.
14403. **comment:** * String cache should ideally be at duk_hthread level, but that would * cause string finalization to slow down relative to the number of * threads; string finalization must check the string cache for "weak" * references to the string being finalized to avoid dead pointers. * * Thus, string caches are now at the heap level now.
 label: code-design
14404. Stringcache is used for speeding up char-offset-to-byte-offset * translations for non-ASCII strings.
14405. For manual debugging: instruction count based on executor and * interrupt counter book-keeping. Inspect debug logs to see how * they match up.
14406. Heap udata, used for allocator functions but also for other heap * level callbacks like pointer compression, etc.
14407. load factor min 25%
14408. helper to insert a (non-string) heap object into heap allocated list
14409. initial stringtable size, must be prime and higher than DUK_UTIL_MIN_HASH_PRIME
14410. used elements (includes DELETED)
14411. marker for detecting internal "double faults", see duk_error_throw.c
14412. callbacks and udata; dbg_read_cb != NULL is used to indicate attached state
14413. debugger state, only relevant when attached
14414. allocated heap objects
14415. 4'294'967'295
14416. longjmp type
14417. Mark-and-sweep interval is relative to combined count of objects and * strings kept in the heap during the latest mark-and-sweep pass. * Fixed point .8 multiplier and .0 adder. Trigger count (interval) is * decreased by each (re)allocation attempt (regardless of size), and each * refzero processed object. * * 'SKIP' indicates how many (re)allocations to wait until a retry if * GC is skipped because there is no thread do it with yet (happens * only during init phases).
14418. Precomputed pointers when using 16-bit heap pointer packing.
14419. Maximum number of breakpoints. Only breakpoints that are set are * consulted so increasing this has no performance impact.
14420. Milliseconds between status notify and transport peeks.
14421. value1 -> error object
14422. mark-and-sweep is currently running
14423. step type: none, step into, step over, step out
14424. refcount code is processing refzero list
14425. emergency mode: try extra hard
14426. time when status/peek was last done (Date-based rate limit)
14427. Opcode interval for a Date-based status/peek rate limit check. Only * relevant when debugger is attached. Requesting a timestamp may be a * slow operation on some platforms so this shouldn't be too low. On the * other hand a high value makes Duktape react to a pause request slowly.
14428. mark-and-sweep control
14429. if 0, 'str16' used, if > 0, 'strlist16' used
14430. * Stringtable entry for fixed size stringtable
14431. no refcounting
14432. currently processing messages or breakpoints: don't enter message processing recursively (e.g. no breakpoints when processing debugger eval)
14433. probe sequence (open addressing)
14434. * Heap structure. * * Heap contains allocated heap objects, interned strings, and built-in * strings for one or more threads.
14435. starting line number

14436. Fatal error handling, called e.g. when a longjmp() is needed but * ljjmpbuf_ptr is NULL. fatal_func must never return; it's not * declared as "noreturn" because doing that for typedefs is a bit * challenging portability-wise.

14437. if 0, 'str' used, if > 0, 'strlist' used

14438. heap level "stash" object (e.g., various reachability roots)

14439. * Heap flags

14440. value1 -> return value, pseudo-type to indicate a return continuation (for ENDFIN)

14441. * Raw memory calls: relative to heap, but no GC interaction

14442. mark-and-sweep flags automatically active (used for critical sections)

14443. heap string indices are autogenerated in duk_strings.h

14444. * Prototypes

14445. 50x heap size

14446. string access cache (codepoint offset -> byte offset) for fast string * character looping; 'weak' reference which needs special handling in GC.

14447. currently running thread

14448. fixed top level hashtable size (separate chaining)

14449. longjmp state

14450. debugger detaching; used to avoid calling detach handler recursively

14451. required

14452. **comment:** A 16-bit listlen makes sense with 16-bit heap pointers: there * won't be space for 64k strings anyway.
label: code-design

14453. built-in strings

14454. executor interrupt running (used to avoid nested interrupts)

14455. **comment:** * Memory calls: relative to heap, GC interaction, but no error throwing. * * XXX: Currently a mark-and-sweep triggered by memory allocation will run * using the heap->heap_thread. This thread is also used for running * mark-and-sweep finalization; this is not ideal because it breaks the * isolation between multiple global environments. * * Notes: * * - DUK_FREE() is required to ignore NULL and any other possible return * value of a zero-sized alloc/realloc (same as ANSI C free()). * * - There is no DUK_REALLOC_ZEROED because we don't assume to know the * old size. Caller must zero the reallocated memory. * * - DUK_REALLOC_INDIRECT() must be used when a mark-and-sweep triggered * by an allocation failure might invalidate the original 'ptr', thus * causing a realloc retry to use an invalid pointer. Example: we're * reallocating the value stack and a finalizer resizes the same value * stack during mark-and-sweep. The indirect variant requests for the * current location of the pointer being reallocated using a callback * right before every realloc attempt; this circuitous approach is used * to avoid strict aliasing issues in a more straightforward indirect * pointer (void ***) approach. Note: the pointer in the storage * location is read but is NOT updated; the caller must do that.
label: code-design

14456. current setjmp() catchpoint

14457. isError flag for yield

14458. **comment:** XXX: make active breakpoints actual copies instead of pointers?
label: code-design

14459. value1 -> label number, pseudo-type to indicate a continue continuation (for ENDFIN)

14460. These are for rate limiting Status notifications and transport peeking.

14461. Used to support single-byte stream lookahead.

14462. duk_handle_call / duk_handle_safe_call recursion depth limiting

14463. **comment:** unused
label: code-design

14464. * Main heap structure

14465. currently active breakpoints: NULL term, borrowed pointers

14466. 2nd related value (type specific)

14467. resizing parameters

14468. 1st related value (type specific)

14469. an error handler (user callback to augment/replace error) is running

14470. mix-in value for computing string hashes; should be reasonably unpredictable

14471. Starting from this round, use emergency mode * for mark-and-sweep.

14472. work list for objects to be finalized (by mark-and-sweep)

14473. value of dbg_exec_counter when we last did a Date-based check

14474. * Stringtable

14475. DUK_USE_DEBUGGER_SUPPORT

14476. mark-and-sweep marking reached a recursion limit and must use multi-pass marking

14477. callstack index

14478. debugger

14479. DUK_HEAP_H_INCLUDED

14480. cumulative opcode execution count (overflows are OK)

14481. **comment:** alloc function typedefs in duktape.h
label: code-design

14482. value1 -> yield value, iserror -> error / normal

14483. work list for objects whose refcounts are zero but which have not been * "finalized"; avoids recursive C calls when refcounts go to zero in a * chain of objects.

14484. strings up to the this length are not cached

14485. * Built-in strings

14486. * Debugger support

14487. 1x heap size

14488. currently paused: talk with debug client until step/resume

14489. don't resize stringtable (but may sweep it); needed during stringtable resize

14490. rnd_state for duk_util_tinyrandom.c

14491. string intern table (weak refs)

14492. load factor max 75%

14493. Currently nothing to free

14494. No str[] pointer.

14495. explicit NULL inits

14496. some derived types

14497. **comment:** * Allocate heap struct * * Use a raw call, all macros expect the heap to be initialized
label: code-design

14498. **comment:** suppress warning, not used
label: code-design

14499. Bernstein hash init value is normally 5381; XOR it in in case pointer low bits are 0

14500. Run mark-and-sweep a few times just in case (unreachable object * finalizers run already here). The last round must rescue objects * from the previous round without running any more finalizers. This * ensures rescued objects get their FINALIZED flag cleared so that * their finalizer is called once more in forced finalization to * satisfy finalizer guarantees. However, we don't want to run any * more finalizer because that'd required one more loop, and so on.

14501. **comment:** Similar workaround for INFINITY.
label: code-design

14502. **comment:** XXX: make a common DUK_USE_option, and allow custom fixed seed?
label: code-design

14503. Note: heap->heap_thread, heap->curr_thread, and heap->heap_object * are on the heap allocated list.

14504. * All done
14505. 'thr' is now reachable
14506. **comment:** These is not 100% because format would need to be non-portable "long long". * Also print out as doubles to catch cases where the "long" type is not wide * enough; the limits will then not be printed accurately but the magnitude * will be correct.
 label: code-design
14507. **comment:** Silence a few global unused warnings here.
 label: code-design
14508. Internal keys are prefixed with 0xFF in the stringtable * (which makes them invalid UTF-8 on purpose).
14509. **comment:** XXX: here again finalizer thread is the heap_thread which needs * to be coordinated with finalizer thread fixes.
 label: code-design
14510. DUK_USE_ROM_STRINGS
14511. DUK_USE_HEAPTR16
14512. assumes that allocated pointers and alloc funcs are valid * if res exists
14513. **comment:** No need to length check string: it will never exceed even * the 16-bit length maximum.
 label: code-design
14514. basic platform types
14515. Currently nothing to free; 'data' is a heap object
14516. mark-and-sweep not running -> must be empty
14517. The casts through duk_intr_pt is to avoid the following GCC warning: * * warning: cast from pointer to integer of different size [-Wpointer-to-int-cast] * * This still generates a /Wp64 warning on VS2010 when compiling for x86.
14518. Only objects in heap_allocated may have finalizers. Check that * the object itself has a _Finalizer property (own or inherited) * so that we don't execute finalizers for e.g. Proxy objects.
14519. * Init the heap object
14520. **comment:** XXX: zero assumption
 label: code-design
14521. unsigned
14522. **comment:** XXX: Add a flag to reject an attempt to re-attach? Otherwise * the detached callback may immediately reattach.
 label: code-design
14523. Cannot wrap: each object is at least 8 bytes so count is * at most 1/8 of that.
14524. * Init stringtable: fixed variant
14525. Fixed seed value used with ROM strings.
14526. **comment:** Workaround for some exotic platforms where NAN is missing * and the expression (0.0 / 0.0) does NOT result in a NaN. * Such platforms use the global 'duk_computed_nan' which must * be initialized at runtime. Use 'volatile' to ensure that * the compiler will actually do the computation and not try * to do constant folding which might result in the original * problem.
 label: code-design
14527. refzero not running -> must be empty
14528. We don't log or warn about freeing zero refcount objects * because they may happen with finalizer processing.
14529. * Init the heap thread
14530. Constants for built-in string data depacking.
14531. structs from duk_forwdecl.h
14532. * Zero the struct, and start initializing roughly in order
14533. * duk_heap allocation and freeing.
14534. convenience
14535. just one 'int' for C++ exceptions
14536. **comment:** XXX: use the pointer as a seed for now: mix in time at least
 label: code-design
14537. 0 = uppercase, 32 = lowercase (= 'a' - 'A')
14538. * Init built-in strings
14539. **comment:** XXX: with 'caller' property the callstack would need * to be unwound to update the 'caller' properties of * functions in the callstack.
 label: code-design
14540. **comment:** Note: first argument not really used
 label: code-design
14541. * Free the heap. * * Frees heap-related non-heap-tracked allocations such as the * string intern table; then frees the heap allocated objects; * and finally frees the heap structure itself. Reference counts * and GC markers are ignored (and not updated) in this process, * and finalizers won't be called. * * The heap pointer and heap object pointers must not be used * after this call.
14542. rescue no longer supported
14543. Special flags checks. Since these strings are always * reachable and a string cannot appear twice in the string * table, there's no need to check/set these flags elsewhere. * The 'internal' flag is set by string intern code.
14544. res->mark_and_sweep_trigger_counter == 0 -> now causes immediate GC; which is OK
14545. important chosen base types
14546. * Allocate a heap. * * String table is initialized with built-in strings from genbuiltins.py, * either by dynamically creating the strings or by referring to ROM strings.
14547. **comment:** Execute finalizers before freeing the heap, even for reachable * objects, and regardless of whether or not mark-and-sweep is * enabled. This gives finalizers the chance to free any native * resources like file handles, allocations made outside Duktape, * etc. This is quite tricky to get right, so that all finalizer * guarantees are honored. * * XXX: this perhaps requires an execution time limit.
 label: code-design
14548. * Computed values (e.g. INFINITY)
14549. skip finalizers; queue finalizable objects to heap_allocated
14550. tval
14551. DUK_USE_DEBUG
14552. basic types from duk_features.h
14553. * Init stringcache
14554. no res->strs[]
14555. DUK_USE_EXPLICIT_NULL_INIT
14556. * Free a heap object. * * Free heap object and its internal (non-heap) pointers. Assumes that * caller has removed the object from heap allocated list or the string * intern table, and any weak references (which strings may have) have * been already dealt with.
14557. Detach a debugger if attached (can be called multiple times) * safely.
14558. res->strs16[] is zeroed and zero decodes to NULL, so no NULL inits.
14559. Decrease by 25% every round
14560. Each round of finalizer execution may spawn new finalizable objects * which is normal behavior for some applications. Allow multiple * rounds of finalization, but use a shrinking limit based on the * first round to detect the case where a runaway finalizer creates * an unbounded amount of new finalizable objects. Finalizer rescue * is not supported: the semantics are unclear because most of the * objects being finalized here are already reachable. The finalizer * is given a boolean to indicate that rescue is not possible. * * See discussion in: <https://github.com/svaarala/duktape/pull/473>
14561. nothing to NULL
14562. derived types
14563. strings are only tracked by stringtable
14564. Don't free h->resumer because it exists in the heap. * Callstack entries also contain function pointers which * are not freed for the same reason.
14565. DUK_USE_STRTAB_PROBE

14566. maps to finalizer 2nd argument
14567. * If selftests enabled, run them as early as possible
14568. zero
14569. **comment:** XXX: The incref macro takes a thread pointer but doesn't * use it right now.
label: code-design
14570. **comment:** XXX: error handling is incomplete. It would be cleanest if * there was a setjmp catchpoint, so that all init code could * freely throw errors. If that were the case, the return code * passing here could be removed.
label: code-design
14571. DUK_USE_STRTAB_CHAIN
14572. Prevent mark-and-sweep for the pending finalizers, also prevents * refzero handling from moving objects away from the heap_allocated * list. (The flag meaning is slightly abused here.)
14573. default prototype (Note: 'thr' must be reachable)
14574. basic types
14575. **comment:** XXX: this may now fail, and is not handled correctly
label: code-design
14576. * Debug dump type sizes
14577. With ROM-based strings, heap->strs[] and thr->strs[] are omitted * so nothing to initialize for strs[].
14578. * Init stringtable: probe variant
14579. **comment:** XXX: inefficient loop
label: code-design
14580. note: mixing len into seed improves hashing when skipping
14581. **comment:** Use Murmurhash2 directly for short strings, and use "block skipping" * for long strings: hash an initial part and then sample the rest of * the string with reasonably sized chunks. An initial offset for the * sampling is computed based on a hash of the initial part of the string; * this is done to (usually) avoid the case where all long strings have * certain offset ranges which are never sampled. * * Skip should depend on length and bound the total time to roughly * logarithmic. With current values: * * 1M string => 256 * 241 = 61696 bytes (0.06M) of hashing * 1G string => 256 * 16321 = 4178176 bytes (3.98M) of hashing * * XXX: It would be better to compute the skip offset more "smoothly" * instead of having a few boundary values.
label: code-design
14582. Bernstein hash init value is normally 5381
14583. Constants for duk_hashstring().
14584. DUK_USE_STRHASH_DENSE
14585. **comment:** Truncate to 16 bits here, so that a computed hash can be compared * against a hash stored in a 16-bit field.
label: code-design
14586. **comment:** * String hash computation (interning). * * String hashing is performance critical because a string hash is computed * for all new strings which are candidates to be added to the string table. * However, strings actually added to the string table go through a codepoint * length calculation which dominates performance because it goes through * every byte of the input string (but only for strings added). * * The string hash algorithm should be fast, but on the other hand provide * good enough hashes to ensure both string table and object property table * hash tables work reasonably well (i.e., there aren't too many collisions * with real world inputs). Unless the hash is cryptographic, it's always * possible to craft inputs with maximal hash collisions. * * NOTE: The hash algorithms must match src/dukutil.py:duk_heap_hashstring() * for ROM string support!
label: code-design
14587. off >= step, and step >= 1
14588. Slightly modified "Bernstein hash" from: * * http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx * * Modifications: string skipping and reverse direction similar to * Lua 5.1.5, and different hash initializer. * * The reverse direction ensures last byte it always included in the * hash which is a good default as changing parts of the string are * more often in the suffix than in the prefix.
14589. may be NULL, too
14590. last element that was left in the heap
14591. weak refs should be handled here, but no weak refs for * any non-string objects exist right now.
14592. Note: object cannot be a finalizable unreachable object, as * they have been marked temporarily reachable for this round, * and are handled above.
14593. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize or a forced relocating realloc?
label: code-design
14594. * Finish
14595. * If object has been marked finalizable, move it to the * "to be finalized" work list. It will be collected on * the next mark-and-sweep if it is still unreachable * after running the finalizer.
14596. * String table resize check. * * Note: this may silently (and safely) fail if GC is caused by an * allocation call in stringtable resize_hash(). Resize_hash() * will prevent a recursive call to itself by setting the * DUK_MS_FLAG_NO_STRINGTABLE_RESIZE in heap->mark_and_sweep_base_flags.
14597. heapdr: * - is not reachable * - is an object * - is not a finalized object * - has a finalizer
14598. finally free the struct itself
14599. deal with weak references first
14600. **comment:** XXX: disable error handlers for duration of compaction?
label: code-design
14601. * Fallback marking handler if recursion limit is reached. * * Iterates 'temproots' until recursion limit is no longer hit. Note * that temproots may reside either in heap allocated list or the * refzero work list. This is a slow scan, but guarantees that we * finish with a bounded C stack. * * Note that nodes may have been marked as temproots before this * scan begun, OR they may have been marked during the scan (as * we process nodes recursively also during the scan). This is * intended behavior.
14602. XXX: for threads, compact value stack, call stack, catch stack?
14603. only objects have finalizers
14604. **comment:** * Sweep garbage and remove marking flags, and move objects with * finalizers to the finalizer work list. * * Objects to be swept need to get their refcounts finalized before * they are swept. In other words, their target object refcounts * need to be decreased. This has to be done before freeing any * objects to avoid decreffing dangling pointers (which may happen * even without bugs, e.g. with reference loops) * * Because strings don't point to other heap objects, similar * finalization is not necessary for strings.
label: code-design
14605. this case can no longer occur because refcount is unsigned
14606. DUK_USE_HEAPPTR16
14607. free object and all auxiliary (non-heap) allocs
14608. **comment:** XXX: more emergency behavior, e.g. find smaller hash sizes etc
label: code-design
14609. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
14610. Strings and ROM objects are never placed on the heap allocated list.
14611. queue back to heap_allocated
14612. * Assertions after
14613. 'data' is reachable through every compiled function which * contains a reference.
14614. XXX: skip count_free w/o debug?
14615. * Object will be kept; queue object back to heap_allocated (to tail)
14616. * Finalizer torture. Do one fake finalizer call which causes side effects * similar to one or more finalizers on actual objects.
14617. XXX: will need a force flag if garbage collection is triggered * explicitly during paused state.
14618. DUK_USE_ASSERTIONS
14619. DUK_USE_MARK_AND_SWEEP
14620. * Assertion helpers.

14621. must never longjmp
14622. * Object compaction. ** Compaction is assumed to never throw an error.
14623. Cannot simulate individual finalizers because finalize_list only * contains objects with actual finalizers. But simulate side effects * from finalization by doing a bogus function call and resizing the * stacks.
14624. nothing to mark
14625. If debugger is paused, garbage collection is disabled by default.
14626. main reachability roots
14627. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
14628. **comment:** remove the string (mark DELETED), could also call * duk_heap_string_remove() but that would be slow and * pointless because we already know the slot.
label: code-design
14629. * Plain, boring reachable object.
14630. XXX: stringtable emergency compaction?
14631. recursion tracking happens here only
14632. reset voluntary gc trigger count
14633. * Main mark-and-sweep function. ** 'flags' represents the features requested by the caller. The current * heap->mark_and_sweep_base_flags is ORed automatically into the flags; * the base flags mask typically prevents certain mark-and-sweep operations * to avoid trouble.
14634. **comment:** If the thr != NULL, the thr may still be in the middle of * initialization. * XXX: Improve the thread viability test.
label: code-design
14635. Note: DUK_HEAP_HAS_REFZERO_FREE_RUNNING(heap) may be true; a refcount * finalizer may trigger a mark-and-sweep.
14636. * Run (object) finalizers in the "to be finalized" work list.
14637. **comment:** XXX: use advancing pointers instead of index macros -> faster and smaller?
label: code-design
14638. * Object's finalizer was executed on last round, and * object has been happily rescued.
14639. DUK_USE_MARKANDSWEEP_FINALIZER_TORTURE
14640. * Assertions before
14641. * Unreachable object, free
14642. OK
14643. done so that duk_mark_heaphdr() works correctly
14644. must also check refzero_list
14645. * Mark objects on finalize_list. *
14646. DUK_USE_REFERENCE_COUNTING
14647. XXX: remove this feature entirely? it would only matter for * emergency GC. Disable for lowest memory builds.
14648. free inner references (these exist e.g. when external * strings are enabled)
14649. ignored
14650. Non-zero refcounts should not happen because we refcount * finalize all unreachable objects which should cancel out * refcounts (even for cycles).
14651. * Mark-and-sweep garbage collection.
14652. * Mark refzero_list objects. ** Objects on the refzero_list have no inbound references. They might have * outbound references to objects that we might free, which would invalidate * any references held by the refzero objects. A refzero object might also * be rescued by refcount finalization. Refzero objects are treated as * reachability roots to ensure they (or anything they point to) are not * freed in mark-and-sweep.
14653. * Mark the heap.
14654. **comment:** * Object compaction (emergency only). ** Object compaction is a separate step after sweeping, as there is * more free memory for it to work with. Also, currently compaction * may insert new objects into the heap allocated list and the string * table which we don't want to do during a sweep (the reachability * flags of such objects would be incorrect). The objects inserted * are currently: ** - a temporary duk_hbuffer for a new properties allocation * - if array part is abandoned, string keys are interned ** The object insertions go to the front of the list, so they do not * cause an infinite loop (they are not compacted).
label: code-design
14655. mark finalizer work list as reachability roots
14656. **comment:** Require a lot of stack to force a value stack grow/shrink. * Recursive mark-and-sweep is prevented by allocation macros * so this won't trigger another mark-and-sweep.
label: code-design
14657. * Mark unreachable, finalizable objects. ** Such objects will be moved aside and their finalizers run later. They have * to be treated as reachability roots for their properties etc to remain * allocated. This marking is only done for unreachable values which would * be swept later (refzero_list is thus excluded). ** Objects are first marked FINALIZABLE and only then marked as reachability * roots; otherwise circular references might be handled inconsistently.
14658. temproots
14659. refzero_list treated as reachability roots
14660. nothing now
14661. * Finalize refcounts for heap elements just about to be freed. * This must be done for all objects before freeing to avoid any * stale pointer dereferences. ** Note that this must deduce the set of objects to be freed * identically to duk_sweep_heap().
14662. flags have been already cleared
14663. * Reset trigger counter
14664. **comment:** XXX: duk_small_uint_t would be enough for this loop
label: code-design
14665. **comment:** Run fake finalizer. Avoid creating unnecessary garbage.
label: code-design
14666. * Unreachable object about to be swept. Finalize target refcounts * (objects which the unreachable object points to) without doing * refzero processing. Recursive decrefs are also prevented when * refzero processing is disabled. ** Value cannot be a finalizable object, as they have been made * temporarily reachable for this round.
14667. Caller will finish the marking process if we hit a recursion limit.
14668. no mark-and-sweep gc
14669. may have FINALIZED
14670. log this with a normal debug level because this should be relatively rare
14671. * Marking functions for heap types: mark children recursively
14672. **comment:** * Finalize objects in the finalization work list. Finalized * objects are queued back to heap_allocated with FINALIZED set. ** Since finalizers may cause arbitrary side effects, they are * prevented during string table and object property allocation * resizing using the DUK_MS_FLAG_NO_FINALIZERS flag in * heap->mark_and_sweep_base_flags. In this case the objects * remain in the finalization work list after mark-and-sweep * exits and they may be finalized on the next pass. ** Finalization currently happens inside "MARKANDSWEEP_RUNNING" * protection (no mark-and-sweep may be triggered by the * finalizers). As a side effect: ** 1) an out-of-memory error inside a finalizer will not * cause a mark-and-sweep and may cause the finalizer * to fail unnecessarily ** 2) any temporary objects whose refcount decreases to zero * during finalization will not be put into refzero_list; * they can only be collected by another mark-and-sweep ** This is not optimal, but since the sweep for this phase has * already happened, this is probably good enough for now.
label: code-design
14673. hash part is a 'weak reference' and does not contribute
14674. **comment:** Checking this here rather than in memory alloc primitives * reduces checking code there but means a failed allocation * will go through a few retries before giving up. That's * fine because this only happens during debugging.
label: code-design
14675. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers). A prototype loop must not cause * an error to be thrown here; duk_hobject_hasprop_raw() will ignore a * prototype loop silently and indicate that the property doesn't exist.
14676. nothing to process

14677. * Clear (reachable) flags of refzero work list.
14678. * Begin
14679. * Mark roots, hoping that recursion limit is not normally hit. * If recursion limit is hit, run additional reachability rounds * starting from "temproots" until marking is complete. ** Marking happens in two phases: first we mark actual reachability * roots (and run "temproots" to complete the process). Then we * check which objects are unreachable and are finalizable; such * objects are marked as FINALIZABLE and marked as reachability * (and "temproots" is run again to complete the process). ** The heap finalize_list must also be marked as a reachability root. * There may be objects on the list from a previous round if the * previous run had finalizer skip flag.
14680. Run the finalizer, duk_hobject_run_finalizer() sets FINALIZED. * Next mark-and-sweep will collect the object unless it has * become reachable (i.e. rescued). FINALIZED prevents the * finalizer from being executed again before that.
14681. nop
14682. DUK_USE_STRTAB_PROBE
14683. * Reachable object, keep
14684. May happen in some out-of-memory corner cases.
14685. * Sweep stringtable
14686. An object may be in heap_allocated list with a zero * refcount also if it is a temporary object created by * a finalizer; because finalization now runs inside * mark-and-sweep, such objects will not be queued to * refzero_list and will thus appear here with refcount * zero.
14687. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed.
14688. **comment:** Select a thread for mark-and-sweep use. ** XXX: This needs to change later.
 label: code-design
14689. An object may be in heap_allocated list with a zero * refcount if it has just been finalized and is waiting * to be collected by the next cycle.
14690. * Sweep heap
14691. DUK_USE_STRTAB_CHAIN
14692. finalize_list will always be processed completely
14693. approximate
14694. **comment:** XXX: thread selection for mark-and-sweep is currently a hack. * If we don't have a thread, the entire mark-and-sweep is now * skipped (although we could just skip finalizations).
 label: code-design
14695. **comment:** XXX: which lists should participate? to be finalized?
 label: code-design
14696. nargs
14697. mark finalizable as reachability roots
14698. * Clear (reachable) flags of finalize_list ** We could mostly do in the sweep phase when we move objects from the * heap into the finalize_list. However, if a finalizer run is skipped * during a mark-and-sweep, the objects on the finalize_list will be marked * reachable during the next mark-and-sweep. Since they're already on the * finalize_list, no-one will be clearing their REACHABLE flag so we do it * here. (This now overlaps with the sweep handling in a harmless way.)
14699. Used during heap destruction: don't actually run finalizers * because we're heading into forced finalization. Instead, * queue finalizable objects back to the heap_allocated list.
14700. No finalizers for ROM objects
14701. * Misc
14702. * Memory allocation handling.
14703. Note: key issue here is to re-lookup the base pointer on every attempt. * The pointer being reallocated may change after every mark-and-sweep.
14704. for zero size allocations NULL is allowed
14705. **comment:** * Compared to a direct macro expansion this wrapper saves a few * instructions because no heap dereferencing is required.
 label: code-design
14706. Note: must behave like a no-op with NULL and any pointer * returned from malloc/realloc with zero size.
14707. simulate alloc failure on every realloc (except when mark-and-sweep is running)
14708. * Helpers * * The fast path checks are done within a macro to ensure "inlining" * while the slow path actions use a helper (which won't typically be * inlined in size optimized builds).
14709. * Avoid a GC if GC is already running. This can happen at a late * stage in a GC when we try to e.g. resize the stringtable * or compact objects.
14710. * Reallocate memory with garbage collection, using a callback to provide * the current allocated pointer. This variant is used when a mark-and-sweep * (e.g. finalizers) might change the original pointer.
14711. no voluntary gc
14712. * First attempt
14713. simulate alloc failure on every alloc (except when mark-and-sweep is running)
14714. DUK_USE_MARK_AND_SWEEP
14715. assume memset with zero size is OK
14716. * Allocate memory with garbage collection
14717. ptr before mark-and-sweep
14718. Must behave like a no-op with NULL and any pointer returned from * malloc/realloc with zero size.
14719. saves a few instructions to have this wrapper (see comment on duk_heap_mem_alloc)
14720. Count free operations toward triggering a GC but never actually trigger * a GC from a free. Otherwise code which frees internal structures would * need to put in NULLs at every turn to ensure the object is always in * consistent state for a mark-and-sweep.
14721. useful for debugging
14722. ptr may be NULL
14723. * Reallocate memory with garbage collection
14724. * Retry with several GC attempts. Initial attempts are made without * emergency mode; later attempts use emergency mode which minimizes * memory allocations forcibly.
14725. * Free memory
14726. * Voluntary periodic GC (if enabled)
14727. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC
14728. * Avoid a GC if GC is already running. See duk_heap_mem_alloc().
14729. For initial entry use default value; zero forces an * interrupt before executing the first instruction.
14730. DUK_USE_INTERRUPT_COUNTER
14731. * Support functions for duk_heap.
14732. may be NULL
14733. arbitrary remove only works with double linked heap, and is only required by * reference counting so far.
14734. **comment:** The prev/next pointers of the removed duk_heaphdr are left as garbage. * It's up to the caller to ensure they're written before inserting the * object back.
 label: code-design
14735. Copy interrupt counter/init value state to new thread (if any). * It's OK for new_thr to be the same as curr_thr.
14736. yes -> move back to heap allocated
14737. Require a lot of stack to force a value stack grow/shrink.
14738. * Refcount memory freeing loop. ** Frees objects in the refzero_pending list until the list becomes * empty. When an object is freed, its references get decref'd and * may cause further objects to be queued for freeing. ** This could be expanded to allow incremental freeing: just bail out * early and resume at a future alloc/decref/refzero.
14739. 'count' is more or less comparable to normal trigger counter update * which happens in memory block (re)allocation.
14740. * Buffers have no internal references. However, a dynamic * buffer has a separate allocation for the buffer. This is * freed by duk_heap_free_heaphdr_raw().
14741. DUK_USE_REFERENCE_COUNTING
14742. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed, * and we're in an infinite loop.
14743. DUK_USE_REFZERO_FINALIZER_TORTURE

14744. no -> decref members, then free
14745. Run fake finalizer. Avoid creating new refzero queue entries * so that we are not forced into a forever loop.
14746. ignored
14747. no refcounting
14748. **comment:** Avoid fake finalization for the duk__refcount_fake_finalizer function * itself, otherwise we're in infinite recursion.
 label: code-design
14749. * Finalizer check. * * Note: running a finalizer may have arbitrary side effects, e.g. * queue more objects on refzero_list (tail), or even trigger a * mark-and-sweep.
 * * Note: quick reject check should match vast majority of * objects and must be safe (not throw any errors, ever).
14750. An object may have FINALIZED here if it was finalized by mark-and-sweep * on a previous run and refcount then decreased to zero. We won't run the * finalizer again here.
14751. refcount is unsigned, so always true
14752. currently, always the case
14753. not emergency
14754. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers).
14755. must never longjmp
14756. Torture option to shake out finalizer side effect issues: * make a bogus function call for every finalizable object, * essentially simulating the case where everything has a * finalizer.
14757. * Rescue or free.
14758. May happen in some out-of-memory corner cases.
14759. * Once the whole refzero cascade has been freed, check for * a voluntary mark-and-sweep.
14760. tail insert: don't disturb head in case refzero is running
14761. * Detect recursive invocation
14762. nothing to finalize
14763. **comment:** unused
 label: code-design
14764. nothing now
14765. Refzero head is still the same. This is the case even if finalizer * inserted more refzero objects; they are inserted to the tail.
14766. not strictly necessary
14767. * Strings have no internal references but do have "weak" * references in the string cache. Also note that strings * are not on the heap_allocated list like other heap * elements.
14768. duk_object_run_finalizer() sets
14769. * Incref and decref functions. * * Decref may trigger immediate refzero handling, which may free and finalize * an arbitrary number of objects. *
14770. * Heap object refcount finalization. * * When an object is about to be freed, all other objects it refers to must * be decref'd. Refcount finalization does NOT free the object or its inner * allocations (mark-and-sweep shares these helpers), it just manipulates * the refcounts. * * Note that any of the decref's may cause a refcount to drop to zero, BUT * it will not be processed inline; instead, because refzero is already * running, the objects will just be queued to refzero list and processed * later. This eliminates C recursion.
14771. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize?
 label: code-design
14772. bump refcount to prevent refzero during finalizer processing
14773. * Objects have internal references. Must finalize through * the "refzero" work list.
14774. * Pick an object from the head (don't remove yet).
14775. **comment:** XXX: ideally this would be just one flag (maybe a derived one) so * that a single bit test is sufficient to check the condition.
 label: test
14776. approximate
14777. * Remove the object from the refzero list. This cannot be done * before a possible finalizer has been executed; the finalizer * may trigger a mark-and-sweep, and mark-and-sweep must be able * to traverse a complete refzero_list.
14778. **comment:** XXX: better to get base and walk forwards?
 label: code-design
14779. nargs
14780. * Churn refzero_list until empty
14781. remove artificial bump
14782. cannot happen: strings are not put into refzero list (they don't even have the next/prev pointers)
14783. hash part is a 'weak reference' and does not contribute
14784. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC
14785. * Reference counting implementation.
14786. **comment:** * Refzero handling is skipped entirely if (1) mark-and-sweep is * running or (2) execution is paused in the debugger. The objects * are left in the heap, and will be freed by mark-and-sweep or * eventual heap destruction. * * This is necessary during mark-and-sweep because refcounts are also * updated during the sweep phase (otherwise objects referenced by a * swept object would have incorrect refcounts) which then calls here. * This could be avoided by using separate decref macros in * mark-and-sweep; however, mark-and-sweep also calls finalizers which * would use the ordinary decref macros anyway and still call this * function. * * This check must be enabled also when mark-and-sweep support has been * disabled: the flag is also used in heap destruction when running * finalizers for remaining objects, and the flag prevents objects from * being moved around in heap linked lists.
 label: code-design
14787. * Misc
14788. **comment:** * Delete references to given hstring from the heap string cache. * * String cache references are 'weak': they are not counted towards * reference counts, nor serve as roots for mark-and-sweep. When an * object is about to be freed, such references need to be removed.
 label: code-design
14789. update entry, allocating if necessary
14790. may be equal
14791. * String scanning helpers * * All bytes other than UTF-8 continuation bytes ([0x80,0xbf]) are * considered to contribute a character. This must match how string * character length is computed.
14792. take last entry
14793. * Scan from shortest distance: * - start of string * - end of string * - cache entry (if exists)
14794. LRU: move our entry to first
14795. **comment:** * Convert char offset to byte offset * * Avoid using the string cache if possible: for ASCII strings byte and * char offsets are equal and for short strings direct scanning may be * better than using the string cache (which may evict a more important * entry). * * Typing now assumes 32-bit string byte/char offsets (duk_uint_fast32_t). * Better typing might be to use duk_size_t.
 label: code-design
14796. no sce, or sce scan not best
14797. * Update cache entry (allocating if necessary), and move the * cache entry to the first place (in an "LRU" policy).
14798. clen == blen -> pure ascii
14799. **comment:** * For non-ASCII strings, we need to scan forwards or backwards * from some starting point. The starting point may be the start * or end of the string, or some cached midpoint in the string * cache. * * For "short" strings we simply scan without checking or updating * the cache. For longer strings we check and update the cache as * necessary, inserting a new cache entry if none exists.
 label: code-design
14800. **comment:** 'sce' points to the wrong entry here, but is no longer used
 label: code-design
14801. Scan error: this shouldn't normally happen; it could happen if * string is not valid UTF-8 data, and clen/blen are not consistent * with the scanning algorithm.
14802. **comment:** XXX: the string shouldn't appear twice, but we now loop to the * end anyway; if fixed, add a looping assertion to ensure there * is no duplicate.

label: code-design

14803. * For ASCII strings, the answer is simple.
14804. * String cache. * * Provides a cache to optimize indexed string lookups. The cache keeps * track of (byte offset, char offset) states for a fixed number of strings. * Otherwise we'd need to scan from either end of the string, as we store * strings in (extended) UTF-8.
14805. initialize for debug prints, needed if sce==NULL
14806. * A C * B A * C <- sce ==> B * D D
14807. may be less, since DELETED entries are NULLED by rehash
14808. **comment:** load factor too low or high, count actually used entries and resize
- label:** code-design
14809. Count actually used (non-NULL, non-DELETED) entries.
14810. OK
14811. avoid array abandoning which interns strings
14812. unsigned intentionally
14813. dummy
14814. checking for DUK__DELETED_MARKER is not necessary here, but helper does it now
14815. may be NULL
14816. NUL terminate for convenient C access
14817. Undefine local defines
14818. If blen <= 0xffffUL, clen is also guaranteed to be <= 0xffffUL.
14819. Overflow, relevant mainly when listlen is 16 bits.
14820. find and remove string from stringtable; caller must free the string itself
14821. **comment:** * String table algorithm: fixed size string table with array chaining * * The top level string table has a fixed size, with each slot holding * either NULL, string pointer, or pointer to a separately allocated * string pointer list. * * This is good for low memory environments using a pool allocator: the * top level allocation has a fixed size and the pointer lists have quite * small allocation size, which further matches the typical pool sizes * needed by objects, strings, property tables, etc.
- label:** code-design
14822. char: use int cast
14823. guaranteed to succeed
14824. * Create a hstring and insert into the heap. The created object * is directly garbage collectable with reference count zero. * * The caller must place the interned string into the stringtable * immediately (without chance of a longjmp); otherwise the string * is lost.
14825. fail
14826. new_used / size <= 1 / DIV <=> new_used <= size / DIV
14827. DUK_USE_STRTAB_PROBE
14828. DUK_USE_DEBUG
14829. caller is responsible for ensuring this
14830. strings may have inner refs (extdata) in some cases
14831. grow by at most one
14832. Note: hstring is in heap but has refcount zero and is not strongly reachable. * Caller should increase refcount and make the hstring reachable before any * operations which require allocation (and possible gc).
14833. looping should never happen
14834. not strictly necessary
14835. Using an explicit 'ASCII' flag has larger footprint (one call site * only) but is quite useful for the case when there's no explicit * 'clen' in duk_hstring.
14836. **comment:** avoid pressure to add/remove strings, invalidation of call data argument, etc.
- label:** code-design
14837. rehash even if old and new sizes are the same to get rid of * DELETED entries.
14838. DUK_USE_ROM_STRINGS
14839. * Heap stringtable handling, string interning.
14840. Force a resize so that DELETED entries are eliminated. * Another option would be duk_recheck_strtab_size_probe(); * but since that happens on every intern anyway, this whole * check can now be disabled.
14841. **comment:** unused with some debug level combinations
- label:** code-design
14842. * The attempt to allocate may cause a GC. Such a GC must not attempt to resize * the stringtable (though it can be swept); finalizer execution and object * compaction must also be postponed to avoid the pressure to add strings to the * string table. Call site must prevent these.
14843. Because new_size > duk_count_used_probe(heap), guaranteed to work
14844. DUK_USE_HEAPPTR16
14845. failed
14846. avoid recursive string table call
14847. formatted result limited
14848. Free strings in the stringtable and any allocations needed * by the stringtable itself.
14849. * String table algorithm: closed hashing with a probe sequence * * This is the default algorithm and works fine for environments with * minimal memory constraints.
14850. DUK_USE_STRTAB_CHAIN
14851. Now two entries in the same slot, alloc list
14852. **comment:** XXX: could check for e16 == 0 because NULL is guaranteed to * encode to zero.
- label:** code-design
14853. required for rehash to succeed, equality not that useful
14854. * Exposed calls
14855. **comment:** XXX: This is VERY inefficient now, and should be e.g. a * binary search or perfect hash, to be fixed.
- label:** code-design
14856. All strings beginning with 0xff are treated as "internal", * even strings interned by the user. This allows user code to * create internal properties too, and makes behavior consistent * in case user code happens to use a string also used by Duktape * (such as string has already been interned and has the 'internal' * flag set).
14857. st_used remains the same, DELETED is counted as used
14858. **comment:** Prevent any side effects on the string table and the caller provided * str/blen arguments while interning is in progress. For example, if * the caller provided str/blen from a dynamic buffer, a finalizer might * resize that dynamic buffer, invalidating the call arguments.
- label:** code-design
14859. For manual testing only.
14860. **comment:** unused
- label:** code-design
14861. Relies on NULL encoding to zero.
14862. FAIL
14863. * Raw intern and lookup
14864. new_free / size <= 1 / DIV <=> new_free <= size / DIV
14865. Without ROM objects "needs refcount update" == is heap allocated.
14866. **comment:** * Reference counting helper macros. The macros take a thread argument * and must thus always be executed in a specific thread context. The * thread argument is needed for features like finalization. Currently * it is not required for INCREF, but it is included just in case. * * Note that 'raw' macros such as DUK_HEAPHDR_GET_REFCOUNT() are not * defined without DUK_USE_REFERENCE_COUNTING, so caller must #ifdef * around them.
- label:** code-design
14867. **comment:** Slow variants, call to a helper to reduce code size. * Can be used explicitly when size is always more important than speed.

label: code-design

14868. DUK_USE_REFERENCE_COUNTING

14869. refcount macros not defined without refcounting, caller must #ifdef now

14870. round out to 8 bytes

14871. currently nop

14872. result: updated refcount

14873. * Note: type is treated as a field separate from flags, so some masking is * involved in the macros below.

14874. refcounting requires direct heap frees, which in turn requires a dual linked heap

14875. 5 heap flags

14876. mark-and-sweep: finalized (on previous pass)

14877. Original idiom used, minimal code size.

14878. **comment:** XXX: fast int-to-double**label:** code-design

14879. * Assert helpers

14880. Optimized for size.

14881. DUK_USE_ROM_OBJECTS

14882. nop

14883. Optimized for speed.

14884. **comment:** XXX: double evaluation for 'tv' argument.**label:** code-design14885. **comment:** * Common heap header * * All heap objects share the same flags and refcount fields. Objects other * than strings also need to have a single or double linked list pointers * for insertion into the "heap allocated" list. Strings are held in the * heap-wide string table so they don't need link pointers. * * Technically, 'h_refcount' must be wide enough to guarantee that it cannot * wrap (otherwise objects might be freed incorrectly after wrapping). This * means essentially that the refcount field must be as wide as data pointers. * On 64-bit platforms this means that the refcount needs to be 64 bits even * if an 'int' is 32 bits. This is a bit unfortunate, and compromising on * this might be reasonable in the future. * * Heap header size on 32-bit platforms: 8 bytes without reference counting, * 16 bytes with reference counting.**label:** code-design

14886. init pointer fields to null

14887. Check that prev/next links are consistent: if e.g. h->prev is != NULL, * h->prev->next should point back to h.

14888. DUK_USE_FASTINT

14889. **comment:** XXX: fast-int-to-double**label:** code-design

14890. read-only object, in code section

14891. 2 bits for heap type

14892. side effects

14893. mark-and-sweep: children not processed

14894. mark-and-sweep: finalizable (on current pass)

14895. Casting convenience.

14896. **comment:** Convenience for some situations; the above macros don't allow NULLs * for performance reasons.**label:** code-design14897. **comment:** 16 bits would be enough for shared heaphdr flags and duk_hstring * flags. The initial parts of duk_heaphdr_string and duk_heaphdr * must match so changing the flags field size here would be quite * awkward. However, to minimize struct size, we can pack at least * 16 bits of duk_hstring data into the flags field.**label:** code-design

14898. * Heap header definition and assorted macros, including ref counting. * Access all fields through the accessor macros.

14899. Default variants. Selection depends on speed/size preference. * Concretely: with gcc 4.8.1 -Os x64 the difference in final binary * is about +1kB for _FAST variants.

14900. get or set a range of flags; m=first bit number, n=number of bits

14901. **comment:** * Macros to set a duk_tval and update refcount of the target (decref the * old value and incref the new value if necessary). This is both performance * and footprint critical; any changes made should be measured for size/speed.**label:** code-design14902. **comment:** When DUK_USE_HEAPPTR16 (and DUK_USE_REFCOUNT16) is in use, the * struct won't align nicely to 4 bytes. This 16-bit extra field * is added to make the alignment clean; the field can be used by * heap objects when 16-bit packing is used. This field is now * conditional to DUK_USE_HEAPPTR16 only, but it is intended to be * used with DUK_USE_REFCOUNT16 and DUK_USE_DOUBLE_LINKED_HEAP; * this only matter to low memory environments anyway.**label:** code-design14903. **comment:** Fast variants, inline refcount operations except for refzero handling. * Can be used explicitly when speed is always more important than size. * For a good compiler and a single file build, these are basically the * same as a forced inline.**label:** code-design

14904. 25 user flags

14905. DUK_HEAPHDR_H_INCLUDED

14906. With ROM objects "needs refcount update" is true when the value is * heap allocated and is not a ROM object.

14907. **comment:** XXX: no optimized variants yet**label:** requirement14908. **comment:** Faster alternative: avoid making a temporary copy of tvptr_dst and use * fast incref/decref macros.**label:** code-design14909. **comment:** DUK_TVAL_SET_TVAL_UPDREF() is used a lot in executor, property lookups, * etc, so it's very important for performance. Measure when changing. * * NOTE: the source and destination duk_tval pointers may be the same, and * the macros MUST deal with that correctly.**label:** code-design

14910. mark-and-sweep: reachable

14911. * Heap native function representation.

14912. **comment:** The 'magic' field allows an opaque 16-bit field to be accessed by the * Duktape/C function. This allows, for instance, the same native function * to be used for a set of very similar functions, with the 'magic' field * providing the necessary non-argument flags / values to guide the behavior * of the native function. The value is signed on purpose: it is easier to * convert a signed value to unsigned (simply AND with 0xffff) than vice * versa. * * Note: cannot place nargs/magic into the heaphdr flags, because * duk_hobject takes almost all flags already (and needs the spare).**label:** code-design

14913. DUK_HNATIVEFUNCTION_H_INCLUDED

14914. shared object part

14915. hobject management functions

14916. !DUK_SINGLE_FILE

14917. LAYOUT 2

14918. misc

14919. * Macros to access the 'props' allocation.

14920. object has an array part (a_size may still be 0)

14921. * Exposed data

14922. **comment:** * Heap object representation. * * Heap objects are used for Ecmascript objects, arrays, and functions, * but also for internal control like declarative and object environment * records. Compiled functions, native functions, and threads are also * objects but with an extended C struct. * * Objects provide the required Ecmascript semantics and exotic behaviors * especially for property access. * * Properties are stored in three conceptual parts: * * 1. A linear 'entry part' contains

ordered key-value-attributes triples * and is the main method of string properties. ** 2. An optional linear 'array part' is used for array objects to store a * (dense) range of [0,N[array indexed entries with default attributes * (writable, enumerable, configurable). If the array part would become * sparse or non-default attributes are required, the array part is * abandoned and moved to the 'entry part'. ** 3. An optional 'hash part' is used to optimize lookups of the entry * part; it is used only for objects with sufficiently many properties * and can be abandoned without loss of information. ** These three conceptual parts are stored in a single memory allocated area. * This minimizes memory allocation overhead but also means that all three * parts are resized together, and makes property access a bit complicated.

label: code-design

14923. E5 Section 8.6.1

14924. push value to stack

14925. object is extensible

14926. 'Arguments' object and has arguments exotic behavior (non-strict callee)

14927. hash size approx. 1.25 times entries size

14928. * Struct defs

14929. **comment:** unused

label: code-design

14930. maximum length for a SKIP-1 diffstream: 35 bits per entry, rounded up to bytes

14931. **comment:** XXX: when optimizing for guaranteed property slots, use a guaranteed * slot for internal value; this call can then access it directly.

label: code-design

14932. **comment:** * 'props' contains {key,value,flags} entries, optional array entries, and * an optional hash lookup table for non-array entries in a single 'sliced' * allocation. There are several layout options, which differ slightly in * generated code size/speed and alignment/padding; duk_features.h selects * the layout used. * * Layout 1 (DUK_USE_HOBJECT_LAYOUT_1): ** e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * * Layout 2 (DUK_USE_HOBJECT_LAYOUT_2): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_uint8_t) + pad bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * * Layout 3 (DUK_USE_HOBJECT_LAYOUT_3): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * * In layout 1, the 'e_next' count is rounded to 4 or 8 on platforms * requiring 4 or 8 byte alignment. This ensures proper alignment * for the entries, at the cost of memory footprint. However, it's * probably preferable to use another layout on such platforms instead. * * In layout 2, the key and value parts are swapped to avoid padding * the key array on platforms requiring alignment by 8. The flags part * is padded to get alignment for array entries. The 'e_next' count does * not need to be rounded as in layout 1. * * In layout 3, entry values and array values are always aligned properly, * and assuming pointers are at most 8 bytes, so are the entry keys. Hash * indices will be properly aligned (assuming pointers are at least 4 bytes). * Finally, flags don't need additional alignment. This layout provides * compact allocations without padding (even on platforms with alignment * requirements) at the cost of a bit slower lookups. * * Objects with few keys don't have a hash index; keys are looked up linearly, * which is cache efficient because the keys are consecutive. Larger objects * have a hash index part which contains integer indexes to the entries part. * * A single allocation reduces memory allocation overhead but requires more * work when any part needs to be resized. A sliced allocation for entries * makes linear key matching faster on most platforms (more locality) and * skimps on flags size (which would be followed by 3 bytes of padding in * most architectures if entries were placed in a struct). * * 'props' also contains internal properties distinguished with a non-BMP * prefix. Often used properties should be placed early in 'props' whenever * possible to make accessing them as fast as possible.

label: code-design

14933. Object flag. There are currently 26 flag bits available. Make sure * this stays in sync with debugger object inspection code.

14934. finalization

14935. probe sequence

14936. **comment:** function: function must not be tail called

label: code-design

14937. function: create binding for func name (function templates only, used for named function expressions)

14938. Maximum traversal depth for "bound function" chains.

14939. entry part size

14940. additional flags which are passed in the same flags argument as property * flags but are not stored in object properties.

14941. * PC-to-line constants

14942. internal property functions

14943. object is a buffer object (duk_hbufferobject) (always exotic)

14944. * Macro for object validity check * * Assert for currently guaranteed relations between flags, for instance.

14945. internal define property: skip write silently if exists

14946. flags used for property attributes in duk_propdesc and packed flags

14947. object is a thread (duk_hthread)

14948. prop index in 'entry part', < 0 if not there

14949. **comment:** Macro for creating flag initializer from a class number. * Unsigned type cast is needed to avoid warnings about coercing * a signed integer to an unsigned one; the largest class values * have the highest bit (bit 31) set which causes this.

label: code-design

14950. **comment:** hash part size or 0 if unused

label: code-design

14951. Object built-in methods

14952. LAYOUT 1

14953. property is virtual: used in duk_propdesc, never stored * (used by e.g. buffer virtual properties)

14954. implies DUK_HOBJECT_IS_BUFFEROBJECT

14955. * Resizing and hash behavior

14956. custom; implies DUK_HOBJECT_IS_BUFFEROBJECT

14957. function: create new environment when called (see duk_hcompiledfunction)

14958. class masks

14959. 2/3 -> 1/8 = 12.5% min growth

14960. convenience

14961. custom; implies DUK_HOBJECT_IS_THREAD

14962. for updating (all are set to < 0 for virtual properties)

14963. 'Proxy' object

14964. index for next new key ([0,e_next[are gc reachable)

14965. note: this updates refcounts

14966. alias for above

14967. 'Array' object, array length and index exotic behavior

14968. if density < L, abandon array part, L = 3-bit fixed point, e.g. 2 -> 2/8 = 25%

14969. 25%, i.e. less than 25% used -> abandon

14970. Located in duk_heapheader h_extra16. Subclasses of duk_hobject (like * duk_hcompiledfunction) are not free to use h_extra16 for this reason.

14971. hobject property layout

14972. core property functions

14973. DUK_HOBJECT_H_INCLUDED

14974. prop index in 'array part', < 0 if not there

14975. limit is quite low: one array entry is 8 bytes, one normal entry is 4+1+8+4 = 17 bytes (with hash entry)

14976. flags for duk_hobject_get_own_propdesc() and variants
14977. The get/set pointers could be 16-bit pointer compressed but it * would make no difference on 32-bit platforms because duk_tval is * 8 bytes or more anyway.
14978. * Prototypes
14979. * Macros for accessing size fields
14980. object is a compiled function (duk_hcompiledfunction)
14981. macros
14982. Duktape/C (nativefunction) object, exotic 'length'
14983. internal properties
14984. custom
14985. higher value conserves memory; also note that linear scan is cache friendly
14986. read-only values 'lifted' for ease of use
14987. Maximum prototype traversal depth. Sanity limit which handles e.g. * prototype loops (even complex ones like 1->2->3->4->2->3->4->2->3->4).
14988. envrec: (declarative) record is closed
14989. hash size relative to entries size: for value X, approx. hash_prime(e_size + e_size / X)
14990. LAYOUT 3
14991. internal align target for props allocation, must be 2*n for some n
14992. Sanity limit on max number of properties (allocated, not necessarily used). * This is somewhat arbitrary, but if we're close to 2**32 properties some * algorithms will fail (e.g. hash size selection, next prime selection). * Also, we use negative array/entry table indices to indicate 'not found', * so anything above 0x80000000 will cause trouble now.
14993. object is constructable
14994. prototype: the only internal property lifted outside 'e' as it is so central
14995. controls for minimum array part growth
14996. * Macros for property handling
14997. pc2line
14998. ES6 proxy
14999. 2**31-1 ~= 2G properties
15000. function: create an arguments object on function call
15001. accessor
15002. controls for minimum entry part growth
15003. range check not necessary because all 4-bit values are mapped
15004. XXX: duk_uarridx_t?
15005. object has any exotic behavior(s)
15006. E5 Section 8.6.2 + custom classes
15007. prop index in 'hash part', < 0 if not there
15008. don't throw for prototype loop
15009. low-level property functions
15010. object established using Function.prototype.bind()
15011. * Ecmascript [[Class]]
15012. 112.5%, i.e. new size less than 12.5% higher -> fast resize
15013. helpers for defineProperty() and defineProperties()
15014. object is a native function (duk_hnativefunction)
15015. if new_size < L * old_size, resize without abandon check; L = 3-bit fixed point, e.g. 9 -> 9/8 = 112.5%
15016. function: function object is strict
15017. enumeration
15018. array part size (entirely gc reachable)
15019. 'String' object, array index exotic behavior
15020. alloc and init
15021. * Misc
15022. * Allocate an duk_hobject. ** The allocated object has no allocation for properties; the caller may * want to force a resize if a desired size is known. ** The allocated object has zero reference count and is not reachable. * The caller MUST make the object reachable and increase its reference * count before invoking any operation that might require memory allocation.
15023. * Allocate a new thread. ** Leaves the built-ins array uninitialized. The caller must either * initialize a new global context or share existing built-ins from * another thread.
15024. **comment:** * obj->props is intentionally left as NULL, and duk_hobject_props.c must deal * with this properly. This is intentional: empty objects consume a minimum * amount of memory. Further, an initial allocation might fail and cause * 'obj' to "leak" (require a mark-and-sweep) since it is not reachable yet.
 label: code-design
15025. also goes into flags
15026. different memory layout, alloc size, and init
15027. **comment:** XXX: macro? sets both heaphdr and object flags
 label: code-design
15028. * Hobject allocation. ** Provides primitive allocation functions for all object types (plain object, * compiled function, native function, thread). The object return is not yet * in "heap allocated" list and has a refcount of zero, so caller must careful.
15029. Zero encoded pointer is required to match NULL
15030. **comment:** unused now
 label: code-design
15031. when nothing is running, API calls are in non-strict mode
15032. NULL pointer is required to encode to zero, so memset is enough.
15033. **comment:** UNUSED, intentionally empty
 label: code-design
15034. Note: assumes that these string indexes are 8-bit, genstrings.py must ensure that
15035. * Hobject Ecmascript [[Class]].
15036. recheck that the property still exists
15037. * Virtual properties. ** String and buffer indices are virtual and always enumerable, * 'length' is virtual and non-enumerable. Array and arguments * object props have special behavior but are concrete.
15038. 'length' and other virtual properties are not * enumerable, but are included if non-enumerable * properties are requested.
15039. **comment:** XXX: identify enumeration target with an object index (not top of stack)
 label: code-design
15040. * Entries part
15041. [... enum]
15042. * Returns non-zero if a key and/or value was enumerated, and: * * [enum] -> [key] (get_value == 0) * [enum] -> [key value] (get_value == 1) ** Returns zero without pushing anything on the stack otherwise.
15043. still reachable
15044. compact; no need to seal because object is internal
15045. [... res]
15046. -> [... enum key enum_target val]
15047. * Get enumerated keys in an Ecmascript array. Matches Object.keys() behavior * described in E5 Section 15.2.3.14.
15048. return 1 to allow callers to tail call
15049. used for array, stack, and entry indices

15050. [enum_target res]
15051. enumerator must have no keys deleted
15052. [... enum_target res trap_result val true]
15053. * Some E5/E5.1 algorithms require that array indices are iterated * in a strictly ascending order. This is the case for e.g. * Array.prototype.forEach() and JSON.stringify() PropertyList * handling. * * To ensure this property for arrays with an array part (and * arbitrary objects too, since e.g. forEach() can be applied * to an array), the caller can request that we sort the keys * here.
15054. string objects must not created without internal value
15055. control props
15056. **comment:** XXX: not sure what the correct semantic details are here, * e.g. handling of missing values (gaps), handling of non-array * trap results, etc. * * For keys, we simply skip non-string keys which seems to be * consistent with how e.g. Object.keys() will process proxy trap * results (ES6, Section 19.1.2.14).
label: code-design
15057. Enumeration keys are checked against the enumeration target (to see * that they still exist). In the proxy enumeration case _Target will * be the proxy, and checking key existence against the proxy is not * required (or sensible, as the keys may be fully virtual).
15058. -> [... enum_target res]
15059. * Array part * * Note: ordering between array and entry part must match 'abandon array' * behavior in duk_hobject_props.c: key order after an array is abandoned * must be the same.
15060. The internal _Target property is kept pointing to the original * enumeration target (the proxy object), so that the enumerator * 'next' operation can read property values if so requested. The * fact that the _Target is a proxy disables key existence check * during enumeration.
15061. **comment:** * Create an internal enumerator object E, which has its keys ordered * to match desired enumeration ordering. Also initialize internal control * properties for enumeration. * * Note: if an array was used to hold enumeration keys instead, an array * scan would be needed to eliminate duplicates found in the prototype chain.
label: code-design
15062. [enum_target res key true]
15063. -> [... enum key val]
15064. There's intentionally no check for * current underlying buffer length.
15065. -- p_insert -- p_curr * v v * | ... | insert | ... | curr
15066. [... enum_target res trap_result]
15067. must match exactly the number of internal properties inserted to enumerator
15068. **comment:** XXX: avoid this at least when enum_target is an Array, it has an * array part, and no ancestor properties were included? Not worth * it for JSON, but maybe worth it for forEach().
label: code-design
15069. Initialize index so that we skip internal control keys.
15070. -> [... enum key enum_target key]
15071. -> [...]
15072. * Proxy object handling
15073. **comment:** Create a temporary enumerator to get the (non-duplicated) key list; * the enumerator state is initialized without being needed, but that * has little impact.
label: code-design
15074. **comment:** * Helper to sort array index keys. The keys are in the enumeration object * entry part, starting from DUK__ENUM_START_INDEX, and the entry part is dense. * * We use insertion sort because it is simple (leading to compact code,) * works nicely in-place, and minimizes operations if data is already sorted * or nearly sorted (which is a very common case here). It also minimizes * the use of element comparisons in general. This is nice because element * comparisons here involve re-parsing the string keys into numbers each * time, which is naturally very expensive. * * Note that the entry part values are all "true", e.g. * * "1" -> true, "3" -> true, "2" -> true * * so it suffices to only work in the key part without exchanging any keys, * simplifying the sort. * *
http://en.wikipedia.org/wiki/Insertion_sort * * (Compiles to about 160 bytes now as a stand-alone function.)
label: code-design
15075. [enum_target enum res]
15076. **comment:** * Hobject enumeration support. * * Creates an internal enumeration state object to be used e.g. with for-in * enumeration. The state object contains a snapshot of target object keys * and internal control state for enumeration. Enumerator flags allow caller * to e.g. request internal/non-enumerable properties, and to enumerate only * "own" properties. * * Also creates the result value for e.g. Object.keys() based on the same * internal structure. * * This snapshot-based enumeration approach is used to simplify enumeration: * non-snapshot-based approaches are difficult to reconcile with mutating * the enumeration target, running multiple long-lived enumerators at the * same time, garbage collection details, etc. The downside is that the * enumerator object is memory inefficient especially for iterating arrays.
label: code-design
15077. No need to replace the 'enum_target' value in stack, only the * enum_target reference. This also ensures that the original * enum target is reachable, which keeps the proxy and the proxy * target reachable. We do need to replace the internal _Target.
15078. **comment:** XXX: may need a 'length' filter for forEach()
label: code-design
15079. **comment:** XXX use get tval ptr, more efficient
label: code-design
15080. Needs to be inserted; scan backwards, since we optimize * for the case where elements are nearly in order.
15081. -> [... key val]
15082. Neutered buffer, zero length seems * like good behavior here.
15083. String and buffer enumeration behavior is identical now, * so use shared handler.
15084. [res]
15085. Target must be stored so that we can recheck whether or not * keys still exist when we enumerate. This is not done if the * enumeration result comes from a proxy trap as there is no * real object to check against.
15086. -> [... key]
15087. nargs
15088. [... enum_target res trap_result val]
15089. DUK_USE_ES6_PROXY
15090. keep val_highest
15091. -> [... enum_target res trap_result]
15092. we know these because enum objects are internally created
15093. Copy trap result keys into the enumerator object.
15094. no array part
15095. [... enum_target res handler trap]
15096. -> [... enum_target res handler undefined target]
15097. only 'length'
15098. -> [... enum_target res trap handler target]
15099. ensure never re-entered until rescue cycle complete
15100. This shouldn't happen; call sites should avoid looking up * _Finalizer "through" a Proxy, but ignore if we come here * with a Proxy to avoid finalizer re-entry.
15101. -> [... obj finalizer]
15102. **comment:** XXX: Finalizer lookup should traverse the prototype chain (to allow * inherited finalizers) but should not invoke accessors or proxy object * behavior. At the moment this lookup will invoke proxy behavior, so * caller must ensure that this function is not called if the target is * a Proxy.
label: code-design
15103. Note: we rely on duk_safe_call() to fix up the stack for the caller, * so we don't need to pop stuff here. There is no return value; * caller determines rescued status based on object refcount.
15104. [... obj]

15105. nargs

15106. **comment:** * Run an duk_hobject finalizer. Used for both reference counting * and mark-and-sweep algorithms. Must never throw an error. ** There is no return value. Any return value or error thrown by * the finalizer is ignored (although errors are debug logged). ** Notes: ** - The thread used for calling the finalizer is the same as the * 'thr' argument. This may need to change later. ** - The finalizer thread 'top' assertions are there because it is * critical that strict stack policy is observed (i.e. no cruft * left on the finalizer stack).

label: code-design

15107. -> [... obj retval/error]

15108. **comment:** XXX: use a NULL error handler for the finalizer call?

label: code-design

15109. **comment:** * Get and call the finalizer. All of this must be wrapped * in a protected call, because even getting the finalizer * may trigger an error (getter may throw one, for instance).

label: code-design

15110. nrets

15111. [... obj finalizer obj heapDestruct] -> [... obj retval]

15112. this also increases refcount by one

15113. duk_safe_call discipline

15114. -> [...]

15115. Note: we ask for one return value from duk_safe_call to get this * error debugging here.

15116. False if the object is NULL or the prototype 'p' is NULL. * In particular, false if both are NULL (don't compare equal).

15117. * Misc support functions

15118. avoid problems if p == h->prototype

15119. * Iterate the bitstream (line diffs) until PC is reached

15120. 0: no change

15121. 1 0 <2 bits>

15122. 1 1 1 <32 bits>

15123. 0

15124. PC is unsigned. If caller does PC arithmetic and gets a negative result, * it will map to a large PC which is out of bounds and causes a zero to be * returned.

15125. Note: pc is unsigned and cannot be negative

15126. **comment:** workaround: max nbis = 24 now

label: code-design

15127. 1 1 0 <8 bits>

15128. **comment:** XXX: now that pc2line is used by the debugger quite heavily in * checked execution, this should be optimized to avoid value stack * and perhaps also implement some form of pc2line caching (see * future work in debugger.rst).

label: code-design

15129. * Use the index in the header to find the right starting point

15130. **comment:** XXX: add proper spare handling to dynamic buffer, to minimize * reallocs; currently there is no spare at all.

label: code-design

15131. end of diff run

15132. DUK_USE_PC2LINE

15133. Generate pc2line data for an instruction sequence, leaving a buffer on stack top.

15134. end of bytecode

15135. compact

15136. **comment:** * Helpers for creating and querying pc2line debug data, which * converts a bytecode program counter to a source line number. ** The run-time pc2line data is bit-packed, and documented in: ** doc/function-objects.rst

label: documentation

15137. be_ctx->offset == length of encoded bitstream

15138. 1 1 1 <32 bits> * Encode in two parts to avoid bitencode 24-bit limitation

15139. valid pc range is [0, length[

15140. Although there are writable virtual properties (e.g. plain buffer * and buffer object number indices), they are handled before we come * here.

15141. should never happen

15142. [key getter this] -> [key retval]

15143. Note: no key on stack

15144. avoid attempt to compact the current object

15145. key is now reachable in the valstack

15146. no need to pop, nothing was pushed

15147. NULL obj->p is OK

15148. **comment:** XXX: top of stack must contain value, which helper doesn't touch, * rework to use tv_val directly?

label: code-design

15149. all virtual properties are non-configurable and non-writable

15150. not found

15151. resume lookup from target

15152. **comment:** Note: reuse 'curr'

label: code-design

15153. * Get old and new length

15154. * Note: although arguments object variable mappings are only established * for non-strict functions (and a call to a non-strict function created * the arguments object in question), an inner strict function may be doing * the actual property write. Hence the throw_flag applied here comes from * the property write call.

15155. **comment:** XXX: consolidated algorithm step 15.f -> redundant?

label: code-design

15156. **comment:** XXX: very basic optimization -> duk_get_prop_stridx_top

label: code-design

15157. * Check whether we need to abandon an array part (if it exists)

15158. no need to decref, as previous value is 'undefined'

15159. **comment:** * The 'in' operator requires an object as its right hand side, * throwing a TypeError unconditionally if this is not the case. ** However, lightfuncs need to behave like fully fledged objects * here to be maximally transparent, so we need to handle them * here.

label: code-design

15160. USE_PROP_HASH_PART

15161. Get default hash part size for a certain entry part size.

15162. Must coerce key: if key is an object, it may coerce to e.g. 'length'.

15163. key is 'length', cannot match argument exotic behavior

15164. * First check whether property exists; if not, simple case. This covers * steps 1-4.

15165. String is an own (virtual) property of a lightfunc.

15166. Input value should be on stack top and will be coerced and * popped. Refuse to update an Array's 'length' to a value * outside the 32-bit range. Negative zero is accepted as zero.

15167. If neutered must return 0; offset is zeroed during * neutering.

15168. * Ecmascript compliant [[GetProperty]](P), for internal use only. ** If property is found: * - Fills descriptor fields to 'out_desc' * - If

DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero *

* If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with

DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * May cause arbitrary side effects and invalidate (most) duk_tval * pointers.

15169. setter and/or getter may be NULL
15170. Not possible because array object 'length' is present * from its creation and cannot be deleted, and is thus * caught as an existing property above.
15171. replaces top of stack with new value if necessary
15172. **comment:** prev value must be unused, no decref
 label: code-design
15173. * Update an existing property of the base object.
15174. late lookup, avoid side effects
15175. * Debug logging after adjustment.
15176. Notes: * - only numbered indices are relevant, so arr_idx fast reject is good * (this is valid unless there are more than 4**32-1 arguments). * - since variable lookup has no side effects, this can be skipped if * DUK_GETDESC_FLAG_PUSH_VALUE is not set.
15177. **comment:** XXX: option to pretend property doesn't exist if sanity limit is * hit might be useful.
 label: code-design
15178. update length (curr points to length, and we assume it's still valid)
15179. * Post resize assertions.
15180. flags
15181. Abandon array part, moving array entries into entries part. * This requires a props resize, which is a heavy operation. * We also compact the entries part while we're at it, although * this is not strictly required.
15182. IsGenericDescriptor(desc) == true; this means in practice that 'desc' * only has [[Enumerable]] or [[Configurable]] flag updates, which are * allowed at this point.
15183. No resize has occurred so temp_desc->e_idx is still OK
15184. Lookup 'key' from arguments internal 'map', perform a variable write if mapped. * Used in E5 Section 10.6 algorithm for [[DefineOwnProperty]] (used by [[Put]]).
 * Assumes stack top contains 'put' value (which is NOT popped).
15185. **comment:** XXX: this is a major target for size optimization
 label: code-design
15186. implicit attributes
15187. [in_val in_val]
15188. leave result on stack top
15189. if fails, e_size will be zero = not an issue, except performance-wise
15190. Note: no recursion issue, we can trust 'map' to behave
15191. non-object base, no offending virtual property
15192. caller ensures; rom objects are never bufferobjects now
15193. * Some change(s) need to be made. Steps 7-11.
15194. For internal use: get non-accessor entry value and attributes
15195. * Write to entry part
15196. * All done, switch properties ('p') allocation to new one.
15197. * Slow delete, but we don't care as we're already in a very slow path. * The delete always succeeds: key has no exotic behavior, property * is configurable, and no resize occurs.
15198. hash probe sequence
15199. must have been, since in array part
15200. array entries are all plain values
15201. **comment:** XXX: the helper currently assumes stack top contains new * 'length' value and the whole calling convention is not very * compatible with what we need.
 label: code-design
15202. Coerce to number before validating pointers etc so that the * number coercions in duk_hbufferobject_validated_write() are * guaranteed to be side effect free and not invalidate the * pointer checks we do here.
15203. Buffer writes are often integers.
15204. XXX: duk_to_int() ensures we'll get 8 lowest bits as * as input is within duk_int_t range (capped outside it).
15205. [obj key value desc]
15206. resume delete to target
15207. stage 2: delete configurable entries above target length
15208. Lightfunc virtual properties are non-configurable, so * reject if match any of them.
15209. success
15210. [] -> [res]
15211. keep key reachable for GC etc; guaranteed not to fail
15212. Special behavior for 'caller' property of (non-bound) function objects * and non-strict Arguments objects: if 'caller' -value- (!) is a strict * mode function, throw a TypeError (E5 Sections 15.3.5.4, 10.6). * Quite interestingly, a non-strict function with no formal arguments * will get an arguments object -without- special 'caller' behavior! * * The E5.1 spec is a bit ambiguous if this special behavior applies when * a bound function is the base value (not the 'caller' value): Section * 15.3.4.5 (describing bind()) states that [[Get]] for bound functions * matches that of Section 15.3.5.4 ([[Get]] for Function instances). * However, Section 13.3.5.4 has "NOTE: Function objects created using * Function.prototype.bind use the default [[Get]] internal method." * The current implementation assumes this means that bound functions * should not have the special [[Get]] behavior. * * The E5.1 spec is also a bit unclear if the TypeError throwing is * applied if the 'caller' value is a strict bound function. The * current implementation will throw even for both strict non-bound * and strict bound functions. * * See test-dev-strict-func-as-caller-prop-value.js for quite extensive * tests. * * This exotic behavior is disabled when the non-standard 'caller' property * is enabled, as it conflicts with the free use of 'caller'.
15213. !DUK_USE_NONSTD_FUNC_CALLER_PROPERTY
15214. assume array part is comprehensive (contains all array indexed elements * or none of them); hence no need to check the entries part here.
15215. index is above internal buffer length -> property is fully normal
15216. write_to_array_part:
15217. E5 Section 15.5.5.1
15218. ignore retval -> [key]
15219. Resolve Proxy targets until Proxy chain ends. No explicit check for * a Proxy loop: user code cannot create such a loop without tweaking * internal properties directly.
15220. * Because buffer values are often looped over, a number fast path * is important.
15221. * E5 Section 15.4.5.1, steps 3.k - 3.n. The order at the end combines * the error case 3.l.iii and the success case 3.m-3.n. * * Note: 'length' is always in entries part, so no array abandon issues for * 'writable' update.
15222. don't push value
15223. Convert a duk_tval number (caller checks) to a 32-bit index. Returns * DUK_NO_ARRAY_INDEX if the number is not whole or not a valid array * index.
15224. * Intern key via the valstack to ensure reachability behaves * properly. We must avoid longjmp's here so use non-checked * primitives. * * Note: duk_check_stack() potentially reallocs the valstack, * invalidating any duk_tval pointers to valstack. Callers * must be careful.
15225. **comment:** never shrinks; auto-adds DUK_VALSTACK_INTERNAL_EXTRA, which is generous
 label: code-design
15226. increases key refcount
15227. Reject attempt to change virtual properties: not part of the * standard algorithm, applies currently to e.g. virtual index * properties of buffer objects (which are virtual but writable). * (Cannot "force" modification of a virtual property.)
15228. avoid attempts to add/remove object keys
15229. curr is accessor, desc is data
15230. Set up pointers to the new property area: this is hidden behind a macro * because it is memory layout specific.
15231. * Local defines
15232. E5 Section 15.4.5.1, step 4

15233. Lookup 'key' from arguments internal 'map', perform a variable lookup * if mapped, and leave the result on top of stack (and return non-zero). * Used in E5 Section 10.6 algorithms [[Get]] and [[GetOwnProperty]].

15234. Note: intentionally use approximations to shave a few instructions: * a_used = old_used (accurate: old_used + 1) * a_size = arr_idx (accurate: arr_idx + 1)

15235. [... old_result result] -> [... result]

15236. * Detach actual buffer from dynamic buffer in valstack, and * pop it from the stack. ** XXX: the buffer object is certainly not reachable at this point, * so it would be nice to free it forcibly even with only * mark-and-sweep enabled. Not a big issue though.

15237. Must avoid duk_pop() in exit path

15238. DUK_USE_FASTINT

15239. no decref needed for a number

15240. Leave 'getter' on stack

15241. [key] -> [undefined] (default value)

15242. **comment:** XXX: There is currently no support for writing buffer object * indexed elements here. Attempt to do so will succeed and * write a concrete property into the buffer object. This should * be fixed at some point but because buffers are a custom feature * anyway, this is relatively unimportant.
label: code-design

15243. **comment:** array part must not contain any non-unused properties, as they would * be configurable and writable.
label: code-design

15244. force_flag

15245. If not found, resume existence check from Function.prototype. * We can just substitute the value in this case; nothing will * need the original base value (as would be the case with e.g. * setters/getters).

15246. * Object internal value * * Returned value is guaranteed to be reachable / incref'd, caller does not need * to incref OR decref. No proxies or accessors are invoked, no prototype walk.

15247. curr value

15248. [key] (coerced)

15249. Note: no allocation pressure, no need to check refcounts etc

15250. This function is only called for objects with array exotic behavior. * The [[DefineOwnProperty]] algorithm for arrays requires that * 'length' can never have a value outside the unsigned 32-bit range, * attempt to write such a value is a RangeError. Here we can thus * assert for this. When Duktape internals go around the official * property write interface (doesn't happen often) this assumption is * easy to accidentally break, so such code must be written carefully. * See test-bi-array-push maxlen.js.

15251. [key setter this val key] -> [key retval]

15252. * Found existing accessor property (own or inherited). * Call setter with 'this' set to orig, and value as the only argument. * Setter calls are OK even for ROM objects. * * Note: no exotic arguments object behavior, because [[Put]] never * calls [[DefineOwnProperty]] (E5 Section 8.12.5, step 5.b).

15253. guaranteed to find an empty slot

15254. Lookup 'key' from arguments internal 'map', and leave replacement value * on stack top if mapped (and return non-zero). * Used in E5 Section 10.6 algorithm for [[GetOwnProperty]] (used by [[Get]]).

15255. [key trap_result] -> []

15256. **comment:** XXX: investigate whether write protect can be handled above, if we * just update length here while ignoring its protected status
label: code-design

15257. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small.
label: code-design

15258. remove value

15259. * Object.isSealed() and Object.isFrozen() (E5 Sections 15.2.3.11, 15.2.3.13) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: all virtual (non-concrete) properties are currently non-configurable * and non-writable (and there are no accessor virtual properties), so they don't * need to be considered here now.

15260. * Fast path for defining array indexed values without interning the key. * This is used by e.g. code for Array prototype and traceback creation so * must avoid interning.

15261. * Object.getOwnPropertyDescriptor() (E5 Sections 15.2.3.3, 8.10.4) * * This is an actual function call.

15262. * Not found

15263. * Proxy helpers

15264. steps 3.h and 3.i

15265. * Found; post-processing (Function and arguments objects)

15266. [key val]

15267. valstack space that suffices for all local calls, including recursion * of other than Duktape calls (getters etc)

15268. the entries [new_e_next, new_e_size_adjusted[are left uninitialized on purpose (ok, not gc reachable)

15269. cannot be arguments exotic

15270. actually used, non-NULL keys

15271. avoid double coercion

15272. Note: assume array part is comprehensive, so that either * the write goes to the array part, or we've abandoned the * array above (and will not come here).

15273. Magically bound variable cannot be an accessor. However, * there may be an accessor property (or a plain property) in * place with magic behavior removed. This happens e.g. when * a magic property is redefined with defineProperty(). * Cannot assert for "not accessor" here.

15274. * Local prototypes

15275. Note: buffer is dynamic so that we can 'steal' the actual * allocation later.

15276. **comment:** XXX: post-checks (such as no duplicate keys)
label: code-design

15277. **comment:** The handler is looked up with a normal property lookup; it may be an * accessor or the handler object itself may be a proxy object. If the * handler is a proxy, we need to extend the valstack as we make a * recursive proxy check without a function call in between (in fact * there is no limit to the potential recursion here). * * (For sanity, proxy creation rejects another proxy object as either * the handler or the target at the moment so recursive proxy cases * are not realized now.)
label: code-design

15278. Check array density and indicate whether or not the array part should be abandoned.

15279. [key value] or [key undefined]

15280. descriptor type specific checks

15281. avoid multiple computations of flags address; bypasses macros

15282. * Found existing own (non-inherited) plain property. * Do an access control check and update in place.

15283. * Ecmascript compliant [[GetOwnProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * Notes: * * - Getting a property descriptor may cause an allocation (and hence * GC) to take place, hence reachability and refcount of all related * values matter. Reallocation of value stack, properties, etc may * invalidate many duk_tval pointers (concretely, those which reside * in memory areas subject to reallocation). However, heap object * pointers are never affected (heap objects have stable pointers). * * - The value of a plain property is always reachable and has a non-zero * reference count. * * - The value of a virtual property is not necessarily reachable from * elsewhere and may have a refcount of zero. Hence we push it onto * the valstack for the caller, which ensures it remains reachable * while it is needed. * * - There are no virtual accessor properties. Hence, all getters and * setters are always related to concretely stored properties, which * ensures that the get/set functions in the resulting descriptor are * reachable and have non-zero refcounts. Should there be virtual * accessor properties later, this would need to change.

15284. **comment:** * XXX: array indices are mostly typed as duk_uint32_t here; duk_uarridx_t * might be more appropriate.
label: code-design

15285. not enumerable

15286. * Allocate and initialize a new entry, resizing the properties allocation * if necessary. Returns entry index (e_idx) or throws an error if alloc fails. * * Sets the key of the entry (increasing the key's refcount), and updates * the hash part if it exists. Caller must set value and flags, and update * the entry value refcount. A decref

for the previous value is not necessary.

15287. should not happen

15288. previous value is assumed to be garbage, so don't touch it

15289. * Abandon array failed, need to decref keys already inserted * into the beginning of new_e_k before unwinding valstack.

15290. standard behavior, step 3.f.i

15291. index is above internal string length -> property is fully normal

15292. * Rebuild the hash part always from scratch (guaranteed to finish). * * Any resize of hash part requires rehashing. In addition, by rehashing * get rid of any elements marked deleted (DUK_HASH_DELETED) which is critical * to ensuring the hash part never fills up.

15293. must not be extensible

15294. **comment:** fill new_h with u32 0xff = UNUSED

label: code-design

15295. * Misc helpers

15296. side effects

15297. NULL if tv_obj is primitive

15298. actual update happens once write has been completed without * error below.

15299. IsDataDescriptor(desc) == true

15300. Target object must be checked for a conflicting * non-configurable property.

15301. guaranteed when building arguments

15302. With this check in place fast paths won't need read-only * object checks. This is technically incorrect if there are * setters that cause no writes to ROM objects, but current * built-ins don't have such setters.

15303. * Copy array elements to new array part.

15304. property exists, either 'desc' is empty, or all values * match (SameValue)

15305. we currently assume virtual properties are not configurable (as none of them are)

15306. * Property lookup

15307. Stage 1: find highest preventing non-configurable entry (if any). * When forcing, ignore non-configurability.

15308. Count actually used entry part entries (non-NULL keys).

15309. Note: use 'curr' as a temp propdesc

15310. Fast check for extending array: check whether or not a slow density check is required.

15311. * GETPROP: Ecmascript property read.

15312. * Property not found in prototype chain.

15313. unsigned

15314. **comment:** * In a fast check we assume old_size equals old_used (i.e., existing * array is fully dense). * * Slow check if: * * (new_size - old_size) / old_size > limit * new_size - old_size > limit * old_size * new_size > (1 + limit) * old_size || limit' is 3 bits fixed point * new_size > (1 + (limit' / 8)) * old_size || * 8 * 8 * new_size > (8 + limit') * old_size || : 8 * new_size > (8 + limit') * (old_size / 8) * new_size > limit" * (old_size / 8) || limit' = 9 -> max 25% increase * arr_idx + 1 > limit" * (old_size / 8) * * This check doesn't work well for small values, so old_size is rounded * up for the check (and the '+ 1' of arr_idx can be ignored in practice): * * arr_idx > limit" * ((old_size + 7) / 8)

label: code-design

15315. * Property already exists. Steps 5-6 detect whether any changes need * to be made.

15316. since new_alloc_size > 0

15317. Must have array part and no conflicting exotic behaviors. * Doesn't need to have array special behavior, e.g. Arguments * object has array part.

15318. Delete elements required by a smaller length, taking into account * potentially non-configurable elements. Returns non-zero if all * elements could be deleted, and zero if all or some elements could * not be deleted. Also writes final "target length" to 'out_result_len'. * This is the length value that should go into the 'length' property * (must be set by the caller). Never throws an error.

15319. * Compact an object. Minimizes allocation size for objects which are * not likely to be extended. This is useful for internal and non- * extensible objects, but can also be called for non-extensible objects. * May abandon the array part if it is computed to be too sparse. * * This call is relatively expensive, as it needs to scan both the * entries and the array part. * * The call may fail due to allocation error.

15320. XXX: fast path for arrays?

15321. force the property to 'undefined' to create a slot for it

15322. This may trigger mark-and-sweep with arbitrary side effects, * including an attempted resize of the object we're resizing, * executing a finalizer which may add or remove properties of * the object we're resizing etc.

15323. This is a pretty awkward control flow, but we need to recheck the * key coercion here.

15324. [obj key undefined]

15325. **comment:** * XXX: duk_uint_fast32_t should probably be used in many places here.

label: code-design

15326. [obj key desc value get set curr_value]

15327. valstack space allocated especially for proxy lookup which does a * recursive property lookup

15328. DUK_USE_HOBJECT_HASH_PART

15329. **comment:** XXX: pre-checks (such as no duplicate keys)

label: code-design

15330. coercion order matters

15331. XXX: The TypeError is currently not applied to bound * functions because the 'strict' flag is not copied by * bind(). This may or may not be correct, the specification * only refers to the value being a "strict mode Function * object" which is ambiguous.

15332. Undefine local defines

15333. [obj key value desc value]

15334. Array properties have exotic behavior but they are concrete, * so no special handling here. * * Arguments exotic behavior (E5 Section 10.6, [[GetOwnProperty]]) * is only relevant as a post-check implemented below; hence no * check here.

15335. since duk_abandon_array_checked() causes a resize, there should be no gaps in keys

15336. If the value happens to be 0xFFFFFFFF, it's not a valid array index * but will then match DUK_NO_ARRAY_INDEX.

15337. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. * * XXX: this is now an overkill for many fast paths. Rework this * to be faster (although switching to a valstack discipline might * be a better solution overall).

label: code-design

15338. default value

15339. write to entry part

15340. For internal use: get array part value

15341. treat like property not found

15342. [... put_value]

15343. **comment:** * Relocate property allocation, moving properties to the new allocation. * * Includes key compaction, rehashing, and can also optionally abandoning * the array part, 'migrating' array entries into the beginning of the * new entry part. Arguments are not validated here, so e.g. new_h_size * MUST be a valid prime. * * There is no support for in-place reallocation or just compacting keys * without resizing the property allocation. This is intentional to keep * code size minimal. * * The implementation is relatively straightforward, except for the array * abandonment process. Array abandonment requires that new string keys * are interned, which may trigger GC. All keys interned so far must be * reachable for GC at all times; valstack is used for that now. * * Also, a GC triggered during this reallocation process must not interfere * with the object being resized. This is currently controlled by using * heap->mark_and_sweep_base_flags to indicate that no finalizers will be * executed (as they can affect ANY object) and no objects are compacted * (it would suffice to protect this particular object only, though). * * Note: a non-checked variant would be nice but is a bit tricky to * implement for the array abandonment process. It's easy for * everything else. * * Note: because we need to potentially resize the valstack (as part * of abandoning the array part), any tval pointers to the valstack * will become invalid after this call.

label: code-design

15344. **comment:** Note: reuse 'curr' as a temp propdesc

label: code-design

15345. XXX: share final setting code for value and flags? difficult because * refcount code is different. Share entry allocation? But can't allocate * until array index checked.

15346. [key] -> []

15347. if abandon_array, new_a_size must be 0

15348. * Helpers to resize properties allocation on specific needs.

15349. for resizing of array part, use slow path

15350. -> [... trap handler]

15351. Note: proxy handling must happen before key is string coerced.

15352. guaranteed to finish, as hash is never full

15353. Push an arbitrary duk_tval to the stack, coerce it to string, and return * both a duk_hstring pointer and an array index (or DUK_NO_ARRAY_INDEX).

15354. * Migrate array to start of entries if requested. ** Note: from an enumeration perspective the order of entry keys matters. * Array keys should appear wherever they appeared before the array abandon * operation.

15355. Note: unconditional throw

15356. If a setter/getter is missing (undefined), the descriptor must * still have the property present with the value 'undefined'.

15357. Avoid side effects that might disturb curr.e_idx until * we're done editing the slot.

15358. **comment:** actually used
label: code-design

15359. **comment:** XXX: any chance of unifying this with the 'length' key handling?
label: code-design

15360. * Array part

15361. -0 is accepted here as index 0 because ToString(-0) == "0" which is * in canonical form and thus an array index.

15362. see below

15363. hash lookup

15364. **comment:** * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written. ** Note: must re-lookup because calls above (e.g. duk_alloc_entry_checked()) * may realloc and compact properties and hence change e_idx.
label: code-design

15365. * NormalizePropertyDescriptor() related helper. ** Internal helper which validates and normalizes a property descriptor * represented as an Ecmascript object (e.g. argument to defineProperty()). * The output of this conversion is a set of defprop_flags and possibly * some values pushed on the value stack; some subset of: property value, * getter, setter. Caller must manage stack top carefully because the * number of values pushed depends on the input property descriptor. ** The original descriptor object must not be altered in the process.

15366. success: leave varname in stack

15367. Note: changing from writable to non-writable is OK

15368. don't touch property attributes or hash part

15369. stage 1 guarantees

15370. current assertion is quite strong: decref's and set to unused

15371. Length in elements: take into account shift, but * intentionally don't check the underlying buffer here.

15372. [key val] -> [key]

15373. * Argument exotic [[Delete]] behavior (E5 Section 10.6) is * a post-check, keeping arguments internal 'map' in sync with * any successful deletes (note that property does not need to * exist for delete to 'succeed'). ** Delete key from 'map'. Since 'map' only contains array index * keys, we can use arr_idx for a fast skip.

15374. [... key trap handler]

15375. * Get old and new length

15376. abandoning requires a props allocation resize and * 'rechecks' the valstack, invalidating any existing * valstack value pointers!

15377. * PUTPROP: Ecmascript property write. ** Unlike Ecmascript primitive which returns nothing, returns 1 to indicate * success and 0 to indicate failure (assuming throw is not set). ** This is an extremely tricky function. Some examples: *** Currently a decref may trigger a GC, which may compact an object's * property allocation. Consequently, any entry indices (e_idx) will * be potentially invalidated by a decref. *** Exotic behaviors (strings, arrays, arguments object) require, * among other things: ** - Preprocessing before and postprocessing after an actual property * write. For example, array index write requires pre-checking the * array 'length' property for access control, and may require an * array 'length' update after the actual write has succeeded (but * not if it fails). ** - Deletion of multiple entries, as a result of array 'length' write. *** Input values are taken as pointers which may point to the valstack. * If valstack is resized because of the put (this may happen at least * when the array part is abandoned), the pointers can be invalidated. * (We currently make a copy of all of the input values to avoid issues.)

15378. * Found existing own or inherited plain property, but original * base is a primitive value.

15379. * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written.

15380. V

15381. ... target is extensible

15382. duk_get_min_grow_e() is always >= 1

15383. * Nice debug log.

15384. * DELPROP: Ecmascript property deletion.

15385. [... varname]

15386. * Ecmascript compliant [[Delete]](P, Throw).

15387. **comment:** XXX: for fastints, could use a variant which assumes a double duk_tval * (and doesn't need to check for fastint again).
label: code-design

15388. **comment:** XXX: the key in 'key in obj' is string coerced before we're called * (which is the required behavior in E5/E5.1/E6) so the key is a string * here already.
label: code-design

15389. note: original, uncoerced base

15390. != 0 => post-update for array 'length' (used when key is an array index)

15391. only need to guarantee 1 more slot, but allocation growth is in chunks

15392. IsAccessorDescriptor(desc) == true

15393. no wrap assuming h_bufobj->length is valid

15394. avoid issues with relocation

15395. [key] -> []

15396. contains value

15397. Receiver: Proxy object

15398. **comment:** XXX: inline into a prototype walking loop?
label: code-design

15399. new entry: previous value is garbage; set to undefined to share write_value

15400. nargs

15401. * Found ** Arguments object has exotic post-processing, see E5 Section 10.6, * description of [[GetOwnProperty]] variant for arguments.

15402. happens when hash part dropped

15403. **comment:** Duktape 0.11.0 and prior tried to optimize the resize by not * counting the number of actually used keys prior to the resize. * This worked mostly well but also caused weird leak-like behavior * as in: test-bug-object-prop-alloc-unbounded.js. So, now we count * the keys explicitly to compute the new entry part size.
label: code-design

15404. no need for decref/incref because value is a number

15405. [key undefined] -> [key]

15406. shared exit path now

15407. attempt to change from accessor to data property

15408. Read value pushed on stack.

15409. curr and desc are data

15410. accessor with defined getter
15411. key coercion (unless already coerced above)
15412. covered by comparison
15413. * Must guarantee all actually used array entries will fit into * new entry part. Add one growth step to ensure we don't run out * of space right away.
15414. [key]
15415. * Object.prototype.hasOwnProperty() and Object.prototype.propertyIsEnumerable().
15416. -> [in_val]
15417. **comment:** Note: for an accessor without getter, falling through to * check for "caller" exotic behavior is unnecessary as * "undefined" will never activate the behavior. But it does * no harm, so we'll do it anyway.
label: code-design
15418. steps 4.a and 4.b are tricky
15419. defaults, E5 Section 8.6.1, Table 7
15420. * For property layout 1, tweak e_size to ensure that the whole entry * part (key + val + flags) is a suitable multiple for alignment * (platform specific). * * Property layout 2 does not require this tweaking and is preferred * on low RAM platforms requiring alignment.
15421. flag for later write
15422. Get minimum array part growth for a certain size.
15423. arrays MUST have a 'length' property
15424. * Note: assuming new_a_size == 0, and that entry part contains * no conflicting keys, refcounts do not need to be adjusted for * the values, as they remain exactly the same. * * The keys, however, need to be interned, incref'd, and be * reachable for GC. Any intern attempt may trigger a GC and * claim any non-reachable strings, so every key must be reachable * at all times. * * A longjmp must not occur here, as the new_p allocation would * be freed without these keys being decref'd, hence the messy * decref handling if intern fails.
15425. * Helpers for managing property storage size
15426. * Arguments handling helpers (argument map mainly). * * An arguments object has exotic behavior for some numeric indices. * Accesses may translate to identifier operations which may have * arbitrary side effects (potentially invalidating any duk_tval * pointers).
15427. NULL if not an object
15428. Catches >0x100000000 and negative values.
15429. currently required
15430. **comment:** XXX: Extensibility check for target uses IsExtensible(). If we * implemented the isExtensible trap and didn't reject proxies as * proxy targets, it should be respected here.
label: code-design
15431. Currently there are no deletable virtual properties, but * with force_flag we might attempt to delete one.
15432. * Start doing property attributes updates. Steps 12-13. * * Start by computing new attribute flags without writing yet. * Property type conversion is done above if necessary.
15433. **comment:** Note: array entries are always writable, so the writability check * above is pointless for them. The check could be avoided with some * refactoring but is probably not worth it.
label: code-design
15434. **comment:** XXX: for array instances we could take a shortcut here and assume * Array.prototype doesn't contain an array index property.
label: code-design
15435. retval indicates delete failed
15436. E5 Section 15.4.5.1, steps 4.e.i - 4.e.ii
15437. **comment:** XXX: handling for array part missing now; this doesn't affect * compliance but causes array entry writes using defineProperty() * to always abandon array part.
label: code-design
15438. Caller doesn't need to check exotic proxy behavior (but does so for * some fast paths).
15439. fall through
15440. * HASPROP: Ecmascript property existence check ("in" operator). * * Interestingly, the 'in' operator does not do any coercion of * the target object.
15441. **comment:** XXX: write protect after flag? -> any chance of handling it here?
label: code-design
15442. If neutered must return 0; length is zeroed during * neutering.
15443. array case is handled comprehensively above
15444. Note: array part values are [[Writable]], [[Enumerable]], * and [[Configurable]] which matches the required attributes * here.
15445. [key setter this val] -> [key retval]
15446. stage 3: update length (done by caller), decide return code
15447. * Not found as concrete or virtual
15448. minimum new length is highest_arr_idx + 1
15449. property is configurable and
15450. consistency requires this
15451. * Coercion and fast path processing
15452. **comment:** XXX: Refactor key coercion so that it's only called once. It can't * be trivially lifted here because the object must be type checked * first.
label: code-design
15453. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside object 'a_size'.
15454. The coercion order must match the ToPropertyDescriptor() algorithm * so that side effects in coercion happen in the correct order. * (This order also happens to be compatible with duk_def_prop(), * although it doesn't matter in practice.)
15455. relevant array index is non-configurable, blocks write
15456. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.
15457. **comment:** Note: we could return the hash index here too, but it's not * needed right now.
label: code-design
15458. P
15459. * Internal helper for defining an accessor property, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes. This is called * very rarely, so the implementation first sets a value to undefined * and then changes the entry to an accessor (this is to save code space).
15460. index/length check guarantees
15461. **comment:** XXX: At the moment Duktape accesses internal keys like _Finalizer using a * normal property set/get which would allow a proxy handler to interfere with * such behavior and to get access to internal key strings. This is not a problem * as such because internal key strings can be created in other ways too (e.g. * through buffers). The best fix is to change Duktape internal lookups to * skip proxy behavior. Until that, internal property accesses bypass the * proxy and are applied to the target (as if the handler did not exist). * This has some side effects, see test-bi-proxy-internal-keys.js.
label: code-design
15462. new value
15463. actually used, non-NULL entries
15464. Do decrefs only with safe pointers to avoid side effects * disturbing e_idx.
15465. shared checks for all descriptor types
15466. strict flag for putvar comes from our caller (currently: fixed)
15467. * Coercion and fast path processing.
15468. Not strictly necessary because if key == NULL, flag MUST be ignored.
15469. caller ensures
15470. traits are separate; in particular, arguments not an array
15471. 0 = don't push current value
15472. Note: new_e_next matches pushed temp key count, and nothing can * fail above between the push and this point.

15473. fill new entries with -unused- (required, gc reachable)
15474. duk_get_min_grow_a() is always ≥ 1
15475. [key getter this key] -> [key retval]
15476. * New length not lower than old length => no changes needed * (not even array allocation).
15477. **comment:** * Write to 'length' of an array is a very complex case * handled in a helper which updates both the array elements * and writes the new 'length'. The write may result in an * unconditional RangeError or a partial write (indicated * by a return code). ** Note: the helper has an unnecessary writability check * for 'length', we already know it is writable.
label: code-design
15478. important for callers
15479. * Arguments exotic behavior not possible for new properties: all * magically bound properties are initially present in the arguments * object, and if they are deleted, the binding is also removed from * parameter map.
15480. clear array part flag only after switching
15481. * Writability check
15482. tv1 points to value storage
15483. caller must have decref'd values above new_a_size (if that is necessary)
15484. * XXX: shrink array allocation or entries compaction here?
15485. not found in 'curr', next in prototype chain; impose max depth
15486. FOUND
15487. Buffer has virtual properties similar to string, but indexed values * are numbers, not 1-byte buffers/strings which would perform badly.
15488. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small; some overlap with string * handling.
label: code-design
15489. entry part must not contain any configurable properties, or * writable properties (if is_frozen).
15490. t ≤ 0 always true, unsigned
15491. Storing the entry top is cheaper here to ensure stack is correct at exit, * as there are several paths out.
15492. Lookup 'key' from arguments internal 'map', delete mapping if found. * Used in E5 Section 10.6 algorithm for [[Delete]]. Note that the * variable/argument itself (where the map points) is not deleted.
15493. Set property slot to an empty state. Careful not to invoke * any side effects while using desc.e_idx so that it doesn't * get invalidated by a finalizer mutating our object.
15494. target
15495. fall thru
15496. [key setter this val]
15497. **comment:** Note: we can reuse 'desc' here
label: code-design
15498. * Array exotic behaviors can be implemented at this point. The local variables * are essentially a 'value copy' of the input descriptor (Desc), which is modified * by the Array [[DefineOwnProperty]] (E5 Section 15.4.5.1).
15499. data property or accessor without getter
15500. **comment:** XXX: fastint
label: code-design
15501. required to guarantee success of rehashing, * intentionally use unadjusted new_e_size
15502. Regardless of whether property is found in entry or array part, * it may have arguments exotic behavior (array indices may reside * in entry part for abandoned / non-existent array parts).
15503. Careful with wrapping (left shifting idx would be unsafe).
15504. [...] -> [...]
15505. currently used -> new size
15506. mask out flags not actually stored
15507. varenv remains reachable through 'obj'
15508. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.
15509. * New length is smaller than old length, need to delete properties above * the new length. ** If array part exists, this is straightforward: array entries cannot * be non-configurable so this is guaranteed to work. ** If array part does not exist, array-indexed values are scattered * in the entry part, and some may not be configurable (preventing length * from becoming lower than their index + 1). To handle the algorithm * in E5 Section 15.4.5.1, step 1 correctly, we scan the entire property * set twice.
15510. **comment:** XXX: optimize this filling behavior later
label: code-design
15511. **comment:** * Property count check. This is the only point where we ensure that * we don't get more (allocated) property space that we can handle. * There aren't hard limits as such, but some algorithms fail (e.g. * finding next higher prime, selecting hash part size) if we get too * close to the 4G property limit. ** Since this works based on allocation size (not actually used size), * the limit is a bit approximate but good enough in practice.
label: code-design
15512. All lightfunc own properties are non-writable and the lightfunc * is considered non-extensible. However, the write may be captured * by an inherited setter which means we can't stop the lookup here.
15513. Convert a duk_tval fastint (caller checks) to a 32-bit index.
15514. [key result] -> [result]
15515. * Standard algorithm succeeded without errors, check for exotic post-behaviors. ** Arguments exotic behavior in E5 Section 10.6 occurs after the standard * [[DefineOwnProperty]] has completed successfully. ** Array exotic behavior in E5 Section 15.4.5.1 is implemented partly * prior to the default [[DefineOwnProperty]], but: * - for an array index key (e.g. "10") the final 'length' update occurs here * - for 'length' key the element deletion and 'length' update occurs here
15516. map is reachable through obj
15517. 0 if no used entries
15518. * All defined array-indexed properties are in the array part * (we assume the array part is comprehensive), and all array * entries are writable, configurable, and enumerable. Thus, * nothing can prevent array entries from being deleted.
15519. resume check from proxy target
15520. **comment:** Count actually used array part entries and array minimum size. * NOTE: 'out_min_size' can be computed much faster by starting from the * end and breaking out early when finding first used entry, but this is * not needed now.
label: code-design
15521. arguments MUST have an initialized lexical environment reference
15522. if new_p == NULL, all of these pointers are NULL
15523. If index is not valid, idx will be DUK_NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside bufferobject length.
15524. automatic length update
15525. always grow the array, no sparse / abandon support here
15526. **comment:** remove old value
label: code-design
15527. [...] -> [...]
15528. always in entry part, no need to look up parents etc
15529. * Copy keys and values in the entry part (compacting them at the same time).
15530. E5 Section 15.4.5.1, step 3, steps a - i are implemented here, j - n at the end
15531. no need to compact since we already did that in duk_abandon_array_checked() * (regardless of whether an array part existed or not).
15532. XXX: C recursion limit if proxies are allowed as handler/target values
15533. **comment:** XXX: assertion that entries \geq old_len are already unused

label: code-design

15534. Arrays never have other exotic behaviors.
15535. * Create a new property in the original object. ** Exotic properties need to be reconsidered here from a write * perspective (not just property attributes perspective). * However, the property does not exist in the object already, * so this limits the kind of exotic properties that apply.
15536. Initial value for highest_idx is -1 coerced to unsigned. This * is a bit odd, but (highest_idx + 1) will then wrap to 0 below * for out_min_size as intended.
15537. * Write to array part? ** Note: array abandoning requires a property resize which uses * 'rechecks' valstack for temporaries and may cause any existing * valstack pointers to be invalidated. To protect against this, * tv_obj, tv_key, and tv_val are copies of the original inputs.
15538. copy existing entries as is
15539. value from hook
15540. **comment:** * Shallow fast path checks for accessing array elements with numeric * indices. The goal is to try to avoid coercing an array index to an * (interned) string for the most common lookups, in particular, for * standard Array objects. ** Interning is avoided but only for a very narrow set of cases: * - Object has array part, index is within array allocation, and * value is not unused (= key exists) * - Object has no interfering exotic behavior (e.g. arguments or * string object exotic behaviors interfere, array exotic * behavior does not). ** Current shortcoming: if key does not exist (even if it is within * the array allocation range) a slow path lookup with interning is * always required. This can probably be fixed so that there is a * quick fast path for non-existent elements as well, at least for * standard Array objects.
- label:** code-design
15541. [... put_value varname]
15542. number
15543. Get minimum entry part growth for a certain size.
15544. if accessor without getter, return value is undefined
15545. undefined is accepted
15546. [obj key desc value get set curr_value varname]
15547. key must not already exist in entry part
15548. * Note: currently no fast path for array index writes. * They won't be possible anyway as strings are immutable.
15549. * Object.seal() and Object.freeze() (E5 Sections 15.2.3.8 and 15.2.3.9) ** Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. ** Note: virtual (non-concrete) properties which are non-configurable but * writable would pose some problems, but such properties do not currently * exist (all virtual properties are non-configurable and non-writable). * If they did exist, the non-configurability does NOT prevent them from * becoming non-writable. However, this change should be recorded somehow * so that it would turn up (e.g. when getting the property descriptor), * requiring some additional flags in the object.
15550. **comment:** * HASPROP variant used internally. ** This primitive must never throw an error, callers rely on this. * In particular, don't throw an error for prototype loops; instead, * pretend like the property doesn't exist if a prototype sanity limit * is reached. ** Does not implement proxy behavior: if applied to a proxy object, * returns key existence on the proxy object itself.
- label:** code-design
15551. **comment:** The fast path for array property put is not fully compliant: * If one places conflicting number-indexed properties into * Array.prototype (for example, a non-writable Array.prototype[7]) * the fast path will incorrectly ignore them. ** This fast path could be made compliant by falling through * to the slow path if the previous value was UNUSED. This would * also remove the need to check for extensibility. Right now a * non-extensible array is slower than an extensible one as far * as writes are concerned. ** The fast path behavior is documented in more detail here: * tests/ecmascript/test-misc-array-fast-write.js
- label:** code-design
15552. * Because buffer values may be looped over and read/written * from, an array index fast path is important.
15553. Note: only numbered indices are relevant, so arr_idx fast reject * is good (this is valid unless there are more than 4**32-1 arguments).
15554. entry allocation updates hash part and increases the key * refcount; may need a props allocation resize but doesn't * 'recheck' the valstack.
15555. * Actual Object.defineProperty() default algorithm.
15556. **comment:** Linear scan: more likely because most objects are small. * This is an important fast path. ** XXX: this might be worth inlining for property lookups.
- label:** code-design
15557. [... value? getter? setter?]
15558. may become sparse...
15559. Get Proxy target object. If the argument is not a Proxy, return it as is. * If a Proxy is revoked, an error is thrown.
15560. **comment:** * Object.defineProperty() related helper (E5 Section 15.2.3.6) ** Inlines all [[DefineOwnProperty]] exotic behaviors. ** Note: Ecmascript compliant [[DefineOwnProperty]](P, Desc, Throw) is not * implemented directly, but Object.defineProperty() serves its purpose. * We don't need the [[DefineOwnProperty]] internally and we don't have a * property descriptor with 'missing values' so it's easier to avoid it * entirely. ** Note: this is only called for actual objects, not primitive values. * This must support virtual properties for full objects (e.g. Strings) * but not for plain values (e.g. strings). Lightfuncs, even though * primitive in a sense, are treated like objects and accepted as target * values.
- label:** code-design
15561. **comment:** * Array needs to grow, but we don't want it becoming too sparse. * If it were to become sparse, abandon array part, moving all * array entries into the entries part (for good). ** Since we don't keep track of actual density (used vs. size) of * the array part, we need to estimate somehow. The check is made * in two parts: * * - Check whether the resize need is small compared to the * current size (relatively); if so, resize without further * checking (essentially we assume that the original part is * "dense" so that the result would be dense enough). * * - Otherwise, compute the resize using an actual density * measurement based on counting the used array entries.
- label:** code-design
15562. * Compute new alloc size and alloc new area. ** The new area is allocated as a dynamic buffer and placed into the * valstack for reachability. The actual buffer is then detached at * the end. ** Note: heap_mark_and_sweep_base_flags are altered here to ensure * no-one touches this object while we're resizing and rehashing it. * The flags must be reset on every exit path after it. Finalizers * and compaction is prevented currently for all objects while it * would be enough to restrict it only for the current object.
15563. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. ** XXX: this is an overkill for some paths, so optimize this later * (or maybe switch to a stack arguments model entirely).
- label:** code-design
15564. **comment:** re-lookup to update curr.flags * XXX: would be faster to update directly
- label:** code-design
15565. This should not happen because DUK_TAG_OBJECT case checks * for this already, but check just in case.
15566. * Hobject property set/get functionality. ** This is very central functionality for size, performance, and compliance. * It is also rather intricate; see hobject-algorithms.rst for discussion on * the algorithms and memory-management.rst for discussion on refcounts and * side effect issues. ** Notes: * * - It might be tempting to assert "refcount nonzero" for objects * being operated on, but that's not always correct: objects with * a zero refcount may be operated on by the refcount implementation * (finalization) for instance. Hence, no refcount assertions are made. * * - Many operations (memory allocation, identifier operations, etc) * may cause arbitrary side effects (e.g. through GC and finalization). * These side effects may invalidate duk_tval pointers which point to * areas subject to reallocation (like value stack). Heap objects * themselves have stable pointers. Holding heap object pointers or * duk_tval copies is not problematic with respect to side effects; * care must be taken when holding and using argument duk_tval pointers. * * - If a finalizer is executed, it may operate on the the same object * we're currently dealing with. For instance, the finalizer might * delete a certain property which has already been looked up and * confirmed to exist. Ideally finalizers would be disabled if GC * happens during property access. At the moment property table realloc * disables finalizers, and all DECREFs may cause arbitrary changes so * handle DECREF carefully. * * - The order of operations for a DECREF matters. When DECREF is executed, * the entire object graph must be consistent; note that a refzero may * lead to a mark-and-sweep through a refcount finalizer.
15567. **comment:** * Entries part is a bit more complex
- label:** code-design
15568. remove key
15569. array 'length' is always a number, as we coerce it
15570. = IsAccessorDescriptor(Desc)
15571. Reject attempt to change a read-only object.
15572. idx_value may be < 0 (no value), set and get may be NULL
15573. Is whole and within 32 bit range. If the value happens to be 0xFFFFFFFF, * it's not a valid array index but will then match DUK_NO_ARRAY_INDEX.

15574. * Find an existing key from entry part either by linear scan or by * using the hash index (if it exists). ** Sets entry index (and possibly the hash index) to output variables, * which allows the caller to update the entry and hash entries in-place. * If entry is not found, both values are set to -1. If entry is found * but there is no hash part, h_idx is set to -1.

15575. -> [... handler trap]

15576. DUK_USE_ES6_PROXY

15577. not reachable

15578. marker values for hash part

15579. 'varname' is in stack in this else branch, leaving an unbalanced stack below, * but this doesn't matter now.

15580. no need for 'caller' post-check, because 'key' must be an array index

15581. remove in_val

15582. Grow entry part allocation for one additional entry.

15583. * Internal helper to define a property with specific flags, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes unless caller requests * that value only be updated if it doesn't already exists. ** Does not support: * - virtual properties (error if write attempted) * - getter/setter properties (error if write attempted) * - non-default (!= WEC) attributes for array entries (error if attempted) * - array abandoning: if array part exists, it is always extended * - array 'length' updating ** Stack: [... in_val] -> [] ** Used for e.g. built-in initialization and environment record * operations.

15584. for zero size, don't push anything on valstack

15585. [key result]

15586. remaining actual steps are carried out if standard DefineOwnProperty succeeds

15587. curr is accessor -> cannot be in array part

15588. E5 Section 15.5.5.2

15589. equality not actually possible

15590. This is not strictly necessary, but avoids compiler warnings; e.g. * gcc won't reliably detect that no uninitialized data is read below.

15591. Avoid zero copy with an invalid pointer. If obj->p is NULL, * the 'new_a' pointer will be invalid which is not allowed even * when copy size is zero.

15592. ignore result

15593. cannot be e.g. arguments exotic, since exotic 'traits' are mutually exclusive

15594. * Arguments objects have exotic [[DefineOwnProperty]] which updates * the internal 'map' of arguments for writes to currently mapped * arguments. More concretely, writes to mapped arguments generate * a write to a bound variable. ** The [[Put]] algorithm invokes [[DefineOwnProperty]] for existing * data properties and new properties, but not for existing accessors. * Hence, in E5 Section 10.6 ([[DefinedOwnProperty]] algorithm), we * have a Desc with 'Value' (and possibly other properties too), and * we end up in step 5.b.i.

15595. * Helper: handle Array object 'length' write which automatically * deletes properties, see E5 Section 15.4.5.1, step 3. This is * quite tricky to get right. ** Used by duk_hobject_putprop().

15596. **comment:** XXX: is valstack top best place for argument?

label: code-design

15597. XXX: sufficient to check 'strict', assert for 'is function'

15598. Leave 'value' on stack

15599. return value

15600. [... val key] -> [... key val]

15601. a dummy undefined value is pushed to make valstack * behavior uniform for caller

15602. * Found existing inherited plain property. * Do an access control check, and if OK, write * new property to 'orig'.

15603. result: hash_prime(floor(1.2 * e_size))

15604. errors out if out of memory

15605. stack contains value (if requested), 'out_desc' is set

15606. curr and desc are accessors

15607. Must be an object, otherwise TypeError (E5.1 Section 8.10.5, step 1).

15608. **comment:** Downgrade checks are not made everywhere, so 'length' is not always * a fastint (it is a number though). This can be removed once length * is always guaranteed to be a fastint.

label: code-design

15609. * Entries part

15610. second incref for the entry reference

15611. * Not found as a concrete property, check whether a String object * virtual property matches.

15612. out_desc is left untouched (possibly garbage), caller must use return * value to determine whether out_desc can be looked up

15613. **comment:** * Array abandon check; abandon if: * * new_used / new_size < limit * new_used < limit * new_size || limit is 3 bits fixed point * new_used < limit' / 8 * new_size || *8 * 8*new_used < limit' * new_size || :8 * new_used < limit' * (new_size / 8) * * Here, new_used = a_used, new_size = a_size. * * Note: some callers use approximate values for a_used and/or a_size * (e.g. dropping a '+1' term). This doesn't affect the usefulness * of the check, but may confuse debugging.

label: code-design

15614. prev value can be garbage, no decref

15615. * Check whether the property already exists in the prototype chain. * Note that the actual write goes into the original base object * (except if an accessor property captures the write).

15616. step 3.e: replace 'Desc.[[Value]]'

15617. **comment:** NOTE: lightfuncs are coerced to full functions because * lightfuncs don't fit into a property value slot. This * has some side effects, see test-dev-lightfunc-accessor.js.

label: code-design

15618. **comment:** * Object.preventExtensions() and Object.isExtensible() (E5 Sections 15.2.3.10, 15.2.3.13) ** Not needed, implemented by macros DUK_HOBJECT_{HAS,CLEAR,SET}_EXTENSIBLE * and the Object built-in bindings.

label: code-design

15619. [str] -> [substr]

15620. value from target

15621. * New length lower than old length => delete elements, then * update length. ** Note: even though a bunch of elements have been deleted, the 'desc' is * still valid as properties haven't been resized (and entries compacted).

15622. Note: this order matters (final value before deleting map entry must be done)

15623. unconditional

15624. Object.defineProperty() calls [[DefineOwnProperty]] with Throw=true

15625. * Fast path for bufferobject getprop/putprop

15626. XXX: shrink array allocation or entries compaction here?

15627. Value is required to be a number in the fast path so there * are no side effects in write coercion.

15628. * Possible pending array length update, which must only be done * if the actual entry write succeeded.

15629. 0 = no update

15630. Grow array part for a new highest array index.

15631. **comment:** XXX: awkward helpers

label: code-design

15632. * Pre resize assertions.

15633. curr is data, desc is accessor

15634. stack prepped for func call: [... trap handler]

15635. remove hash entry (no decref)

15636. resume write to target

15637. For internal use: get non-accessor entry value

15638. * Internal helpers for managing object 'length'

15639. All the flags fit in 16 bits, so will fit into duk_bool_t.

15640. throw

15641. **comment:** XXX: macro checks for array index flag, which is unnecessary here
label: code-design

15642. [... value this_binding]

15643. Note: actual update happens once write has been completed * without error below. The write should always succeed * from a specification viewpoint, but we may e.g. run out * of memory. It's safer in this order.

15644. Note: 'curr' refers to 'length' propdesc

15645. **comment:** * Abandon array part because all properties must become non-configurable. * Note that this is now done regardless of whether this is always the case * (skips check, but performance problem if caller would do this many times * for the same object; not likely).
label: code-design

15646. varname is still reachable

15647. get varenv for varname (callee's declarative lexical environment)

15648. guaranteed to finish

15649. Leave 'setter' on stack

15650. If 16-bit hash is in use, stuff it into duk_heapheader_string flags.

15651. string is 'eval' or 'arguments'

15652. Slightly smaller code without explicit flag, but explicit flag * is very useful when 'clen' is dropped.

15653. computed live

15654. **comment:** XXX: could add flags for "is valid CESU-8" (EcmaScript compatible strings), "is valid UTF-8", "is valid extended UTF-8" (internal strings are not, * regexp bytecode is), and "contains non-BMP characters". These are not * needed right now.
label: code-design

15655. * For an external string, the NUL-terminated string data is stored * externally. The user must guarantee that data behind this pointer * doesn't change while it's used.

15656. get array index related to string (or return DUK_HSTRING_NO_ARRAY_INDEX); * avoids helper call if string has no array index value.

15657. * Prototypes

15658. string hash

15659. The external string struct is defined even when the feature is inactive.

15660. string is a reserved word (strict)

15661. slower but more compact variant

15662. marker value; in E5 2^32-1 is not a valid array index (2^32-2 is highest valid)

15663. Impose a maximum string length for now. Restricted artificially to * ensure adding a heap header length won't overflow size_t. The limit * should be synchronized with DUK_HBUFFER_MAX_BYTELEN. * * E5.1 makes provisions to support strings longer than 4G characters. * This limit should be eliminated on 64-bit platforms (and increased * closer to maximum support on 32-bit platforms).

15664. string is a reserved word (non-strict)

15665. string is ASCII, clen == blen

15666. * String value of 'blen+1' bytes follows (+1 for NUL termination * convenience for C API). No alignment needs to be guaranteed * for strings, but fields above should guarantee alignment-by-4 * (but not alignment-by-8).

15667. **comment:** should never be called
label: code-design

15668. placed in duk_heapheader_string

15669. length in codepoints (must be E5 compatible)

15670. Note: we could try to stuff a partial hash (e.g. 16 bits) into the * shared heap header. Good hashing needs more hash bits though.

15671. * Heap string representation. * * Strings are byte sequences ordinarily stored in extended UTF-8 format, * allowing values larger than the official UTF-8 range (used internally) * and also allowing UTF-8 encoding of surrogate pairs (CESU-8 format). * Strings may also be invalid UTF-8 altogether which is the case e.g. with * strings used as internal property names and raw buffers converted to * strings. In such cases the 'cflen' field contains an inaccurate value. * * EcmaScript requires support for 32-bit long strings. However, since each * 16-bit codepoint can take 3 bytes in CESU-8, this representation can only * support about 1.4G codepoint long strings in extreme cases. This is not * really a practical issue.

15672. string is a valid array index

15673. string is internal

15674. string data is external (duk_hstring_external)

15675. **comment:** Smaller heapheader than for other objects, because strings are held * in string intern table which requires no link pointers. Much of * the 32-bit flags field is unused by flags, so we can stuff a 16-bit * field in there.
label: code-design

15676. length in bytes (not counting NUL term)

15677. DUK_HSTRING_H_INCLUDED

15678. * Misc

15679. Caller must check character offset to be inside the string.

15680. unsigned

15681. This may throw an error though not for valid E5 strings.

15682. * Misc support functions

15683. !DUK_USE_HSTRING_CLEN

15684. **comment:** Most practical strings will go here.
label: code-design

15685. **comment:** Convenience copies from heap/vm for faster access.
label: code-design

15686. allocation size

15687. idx_bottom and idx_retval are only used for book-keeping of * EcmaScript-initiated calls, to allow returning to an EcmaScript * function properly. They are duk_size_t to match the convention * that value stack sizes are duk_size_t and local frame indices * are duk_idx_t.

15688. next to use, highest used is top - 1

15689. flags field: LLLLLLFT, L = label (24 bits), F = flags (4 bits), T = type (4 bits)

15690. current lexical environment (may be NULL if delayed)

15691. Current compiler state (if any), used for augmenting SyntaxErrors.

15692. No field needed when strings are in ROM.

15693. * Catcher defines

15694. end of valstack allocation (exclusive)

15695. function executes as a constructor (called via "new")

15696. cached: valstack_end - valstack (in entries, not bytes)

15697. internal extra elements assumed on function entry, * always added to user-defined 'extra' for e.g. the * duk_check_stack() call.

15698. **comment:** Built-in-objects; may or may not be shared with other threads, * threads existing in different "compartments" will have different * built-ins. Must be stored on a per-thread basis because there * is no intermediate structure for a thread group / compartment. * This takes quite a lot of space, currently 43x4 = 172 bytes on * 32-bit platforms. * * In some cases the builtins array could be ROM based, but it's * sometimes edited (e.g. for sandboxing) so it's better to keep * this array in RAM.
label: code-design

15699. catch part will catch

15700. Note: DUK_VALSTACK_INITIAL_SIZE must be >= DUK_VALSTACK_API_ENTRY_MINIMUM * + DUK_VALSTACK_INTERNAL_EXTRA so that the initial stack conforms to spare * requirements.

15701. roughly 512 bytes

15702. DUK_USE_ROM_STRINGS

15703. thread has yielded
15704. roughly 256 bytes
15705. who resumed us (if any)
15706. callstack index of related activation
15707. * Flags for `__FILE__` / `__LINE__` registered into tracedata
15708. roughly 1 kB
15709. **comment:** Pointer to bytecode executor's `'curr_pc'` variable. Used to copy * the current PC back into the topmost activation when activation * state is about to change (or "syncing" is otherwise needed). This * is rather awkward but important for performance, see `execution.rst`.
label: code-design
15710. activation prevents yield (native call or "new")
15711. function executes in strict mode
15712. Note: it's nice if size is 2^N (at least for 32-bit platforms).
15713. Sanity limits for stack sizes.
15714. borrowed: full `duk_tval` for function being executed; for lightfuncs
15715. indirect allocs
15716. * Assert context is valid: non-NULL pointer, fields look sane. ** This is used by public API call entrypoints to catch invalid 'ctx' pointers * as early as possible; invalid 'ctx' pointers cause very odd and difficult to * diagnose behavior so it's worth checking even when the check is not 100%.
15717. number of activation records in callstack preventing a yield
15718. activation has active breakpoint(s)
15719. Call stack. `[0,callstack_top[` is GC reachable.
15720. thread not currently running
15721. roughly 1.0 kB -> but rounds up to `DUK_VALSTACK_GROW_STEP` in practice
15722. roughly 128 bytes
15723. (reference is valid as long activation exists)
15724. don't report `__FILE__` / `__LINE__` as fileName/lineNumber
15725. `DUK_HTHREAD_H_INCLUDED`
15726. borrowed reference to catch variable name (or NULL if none)
15727. roughly 0.5 kB
15728. bottom of current frame
15729. Note: it's nice if size is 2^N (not $4 \times 4 = 16$ bytes on 32 bit)
15730. Return value when returning to this activation (points to caller * reg, not callee reg); index is absolute (only set if activation is * not topmost). ** Note: `idx_bottom` is always set, while `idx_retval` is only applicable * for activations below the topmost one. Currently `idx_retval` for * the topmost activation is considered garbage (and it not initialized * on entry or cleared on return; may contain previous or garbage * values).
15731. thread currently running (only one at a time)
15732. top of current frame (exclusive)
15733. number of elements guaranteed to be user accessible * (in addition to call arguments) on Duktape/C function entry.
15734. * Prototypes
15735. activation has tail called one or more times
15736. Current strictness flag: affects API calls.
15737. Backpointers.
15738. roughly 2 kB
15739. values for the state field
15740. needed for stepping
15741. Yield/resume book-keeping.
15742. **comment:** XXX: Valstack, callstack, and catchstack are currently assumed * to have non-NULL pointers. Relaxing this would not lead to big * benefits (except perhaps for terminated threads).
label: code-design
15743. finally part will catch
15744. `idx_base` and `idx_base+1` get completion value and type
15745. Executor interrupt default interval when nothing else requires a * smaller value. The default interval must be small enough to allow * for reasonable execution timeout checking but large enough to keep * impact on execution performance low.
15746. Thread state.
15747. request to create catch binding
15748. nop
15749. start of valstack allocation
15750. next instruction to execute (points to 'func' bytecode, stable pointer), NULL for native calls
15751. * Struct defines
15752. Value stack: these are expressed as pointers for faster stack manipulation. * `[valstack, valstack_top[` is GC-reachable, `[valstack_top, valstack_end[` is * not GC-reachable but kept initialized as 'undefined'.
15753. * Heap thread object representation. ** `duk_hthread` is also the 'context' (`duk_context`) for exposed APIs * which mostly operate on the topmost frame of the value stack.
15754. activation is a direct eval call
15755. Shared object part
15756. Interrupt counter for triggering a slow path check for execution * timeout, debugger interaction such as breakpoints, etc. The value * is valid for the current running thread, and both the init and * counter values are copied whenever a thread switch occurs. It's * important for the counter to be conveniently accessible for the * bytecode executor inner loop for performance reasons.
15757. start value for current countdown
15758. type and control flags, label number
15759. borrowed: function being executed; for bound function calls, this is the final, real function, NULL for lightfuncs
15760. current variable environment (may be NULL if delayed)
15761. Catch stack. `[0,catchstack_top[` is GC reachable.
15762. * Stack constants
15763. **comment:** XXX: for a memory-code tradeoff, remove 'func' and make it's access either a function * or a macro. This would make the activation 32 bytes long on 32-bit platforms again.
label: code-design
15764. * Activation defines
15765. Previous value of 'func' caller, restored when unwound. Only in use * when 'func' is non-strict.
15766. thread resumed another thread (active but not running)
15767. roughly 64 bytes
15768. Bottom of valstack for this activation, used to reset * `valstack_bottom` on return; index is absolute. Note: * `idx_top` not needed because top is set to 'nregs' always * when returning to an EcmaScript activation.
15769. catch or with binding is currently active
15770. resume execution from `pc_base` or `pc_base+1` (points to 'func' bytecode, stable pointer)
15771. countdown state
15772. * Thread defines
15773. thread has terminated
15774. **comment:** Current 'this' binding is the value just below `idx_bottom`. * Previously, 'this' binding was handled with an index to the * (calling) valstack. This works for everything except tail * calls, which must not "cumulate" valstack temps.

label: code-design

15775. * duk_hthread allocation and freeing.

15776. valstack

15777. callstack

15778. * Allocate initial stacks for a thread. Note that 'thr' must be reachable * as a garbage collection may be triggered by the allocation attempts. * Returns zero (without leaking memory) if init fails.

15779. For indirect allocs.

15780. catchstack

15781. **comment:** XXX: relocate

label: code-design

15782. DUK_HOBJECT_FLAG_EXOTIC_STRINGOBJ varies

15783. currently, even for Array.prototype

15784. normal valued properties

15785. 'prototype' property for all built-in objects (which have it) has attributes: * [[Writable]] = false, * [[Enumerable]] = false, * [[Configurable]] = false

15786. Architecture, OS, and compiler strings

15787. DUK_USE_LIGHTFUNC_BUILTINS

15788. Setup builtins from ROM objects. All heaps/threads will share * the same readonly objects.

15789. **comment:** Currently all built-in native functions are strict. * This doesn't matter for many functions, but e.g. * String.prototype.charAt (and other string functions) * rely on being strict so that their 'this' binding is * not automatically coerced.

label: code-design

15790. enable exotic behaviors last

15791. Built-in 'name' is not writable by default. Function '.name' * is writable to allow user code to set a '.name' on a native * function.

15792. initjs data is NUL terminated

15793. **comment:** XXX: try to optimize to 8 (would now be possible, <200 used)

label: code-design

15794. * First create all built-in bare objects on the empty valstack. * * Built-ins in the index range [0,DUK_NUM_BUILTINS-1] have value * stack indices matching their eventual thr->builtins[] index. * * Built-ins in the index range [DUK_NUM_BUILTINS,DUK_NUM_ALL_BUILTINS] * will exist on the value stack during init but won't be placed * into thr->builtins[]. These are objects referenced in some way * from thr->builtins[] roots but which don't need to be indexed by * Duktape through thr->builtins[] (e.g. user custom objects).

15795. [(builtin objects) name func]

15796. chain

15797. By default the global object is read-only which is often much * more of an issue than having read-only built-in objects (like * RegExp, Date, etc). Use a RAM-based copy of the global object * and the global environment object for convenience.

15798. Cast converts magic to 16-bit signed value

15799. Packed or unpacked tval

15800. Currently all built-in native functions are strict. * duk_push_c_function() now sets strict flag, so * assert for it.

15801. **comment:** XXX: magic for getter/setter? use duk_def_prop()?

label: code-design

15802. Object property allocation layout

15803. avoid empty label at the end of a compound statement

15804. Create a fresh object environment for the global scope. This is * needed so that the global scope points to the newly created RAM-based * global object.

15805. Inherit from ROM-based global object: less RAM usage, less transparent.

15806. **comment:** XXX: keep property attributes or tweak them here? * Properties will now be non-configurable even when they're * normally configurable for the global object.

label: code-design

15807. always provideThis=true

15808. unsigned

15809. 'constructor' property for all built-in objects (which have it) has attributes: * [[Writable]] = true, * [[Enumerable]] = false, * [[Configurable]] = true

15810. **comment:** XXX: refactor into internal helper, duk_clone_hobject()

label: code-design

15811. Encoding endianness must match target memory layout, * build scripts and genbuiltins.py must ensure this.

15812. def_value

15813. DUK_USE_USER_INITJS

15814. * The default property attributes are correct for all * function valued properties of built-in objects now.

15815. genbuiltins.py ensures

15816. DUK_HOBJECT_FLAG_CONSTRUCTABLE varies

15817. Endianness indicator

15818. **comment:** XXX: any way to avoid decoding magic bit; there are quite * many function properties and relatively few with magic values.

label: code-design

15819. **comment:** XXX: this is a bit awkward because there is no exposed helper * in the API style, only this internal helper.

label: code-design

15820. **comment:** XXX: function properties

label: code-design

15821. * Since built-ins are not often extended, compact them.

15822. convenience

15823. * Pop built-ins from stack: they are now INCREF'd and * reachable from the builtins[] array or indirectly * through builtins[].

15824. **comment:** * Special post-tweaks, for cases not covered by the init data format. * * - Set Date.prototype.toGMTString to Date.prototype.toUTCString. * toGMTString is required to have the same Function object as * toUTCString in E5 Section B.2.6. Note that while Smjs respects * this, V8 does not (the Function objects are distinct). * * - Make DoubleError non-extensible. * * - Add info about most important effective compile options to Duktape. * * - Possibly remove some properties (values or methods) which are not * desirable with current feature options but are not currently * conditional in init data.

label: code-design

15825. **comment:** XXX: set magic directly here? (it could share the c_nargs arg)

label: code-design

15826. only Array.prototype matches

15827. [(builtin objects) name]

15828. DUK_USE_ROM_GLOBAL_CLONE || DUK_USE_ROM_GLOBAL_INHERIT

15829. **comment:** 0 = DUK_HOBJECT_CLASS_UNUSED

label: code-design

15830. * Initialize built-in objects. Current thread must have a valstack * and initialization errors may longjmp, so a setjmp() catch point * must exist.

15831. exhaustive

15832. -> [... new_global new_globalenv new_global new_global]

15833. Alignment guarantee

15834. These functions have trouble working as lightfuncs. * Some of them have specific asserts and some may have * additional properties (e.g. 'require.id' may be written).

15835. DUK_USE_ROM_OBJECTS

15836. some assertions

15837. **comment:** XXX: compression

label: code-design

15838. side effect free
15839. **comment:** integer mixed endian not really used now
 label: code-design
15840. * InitJS code - Ecmascript code evaluated from a built-in source * which provides e.g. backward compatibility. User can also provide * JS code to be evaluated at startup.
15841. always 1 arg
15842. * Create built-in objects by parsing an init bitstream generated * by genbuiltins.py.
15843. * Property attribute defaults are defined in E5 Section 15 (first * few pages); there is a default for all properties and a special * default for 'length' properties. Variation from the defaults is * signaled using a single flag bit in the bitstream.
15844. DUK_HOBJECT_FLAG_STRICT varies
15845. * Then decode the builtins init data (see genbuiltins.py) to * init objects
15846. DUK_HOBJECT_FLAG_EXOTIC_ARRAY varies
15847. must hold DUK_VARARGS
15848. Almost all global level Function objects are constructable * but not all: Function.prototype is a non-constructable, * callable Function.
15849. key, getter and setter, now reachable through object
15850. **comment:** XXX: compression (as an option)
 label: code-design
15851. **comment:** Clone the properties of the ROM-based global object to create a * fully RAM-based global object. Uses more memory than the inherit * model but more compliant.
 label: code-design
15852. No built-in functions are constructable except the top * level ones (Number, etc).
15853. Low memory options
15854. accessor flag not encoded explicitly
15855. probe
15856. no prototype or class yet
15857. DUK_USE_BUILTIN_INITJS
15858. **comment:** XXX: other properties of function instances; 'arguments', 'caller'.
 label: code-design
15859. always 0 args
15860. [(builtin objects)]
15861. must be signed
15862. DUK_HOBJECT_FLAG_NATIVEFUNCTION varies
15863. * Encoding constants, must match genbuiltins.py
15864. XXX: ARRAY_PART for Array prototype?
15865. * For top-level objects, 'length' property has the following * default attributes: non-writable, non-enumerable, non-configurable * (E5 Section 15). * * However, 'length' property for Array.prototype has attributes * expected of an Array instance which are different: writable, * non-enumerable, non-configurable (E5 Section 15.4.5.2). * * This is currently determined implicitly based on class; there are * no attribute flags in the init data.
15866. native function properties
15867. all native functions have NEWENV
15868. push operation normalizes NaNs
15869. **comment:** XXX: This won't be shown in practice now * because this code is not run when builtins * are in ROM.
 label: code-design
15870. no prototype
15871. Copy the property table verbatim; this handles attributes etc. * For ROM objects it's not necessary (or possible) to update * refcounts so leave them as is.
15872. no need to decref
15873. **comment:** XXX: Shrink the stacks to minimize memory usage? May not * be worth the effort because terminated threads are usually * garbage collected quite soon.
 label: code-design
15874. **comment:** XXX: store 'bcode' pointer to activation for faster lookup?
 label: code-design
15875. Write bytecode executor's curr_pc back to topmost activation (if any).
15876. Here we could remove references to built-ins, but it may not be * worth the effort because built-ins are quite likely to be shared * with another (unterminated) thread, and terminated threads are also * usually garbage collected quite quickly. Also, doing DECREFs * could trigger finalization, which would run on the current thread * and have access to only some of the built-ins. Garbage collection * deals with this correctly already.
15877. Order of unwinding is important
15878. ptr_curr_pc != NULL only when bytecode executor is active.
15879. unwinds valstack, updating refcounts
15880. DUK_USE_DEBUGGER_SUPPORT
15881. * Thread support.
15882. side effects, possibly errors
15883. Note: this may cause a corner case situation where a finalizer * may see a currently reachable activation whose 'func' is NULL.
15884. avoid warning (unsigned)
15885. unsigned
15886. * Restore 'caller' property for non-strict callee functions.
15887. **comment:** * Note: must use indirect variant of DUK_REALLOC() because underlying * pointer may be changed by mark-and-sweep.
 label: code-design
15888. true, despite side effect resizes
15889. note: any entries above the callstack top are garbage and not zeroed
15890. cannot grow
15891. With lightfuncs, act 'func' may be NULL
15892. The act->prev_caller should only be set if the entry for 'caller' * exists (as it is only set in that case, and the property is not * configurable), but handle all the cases anyway.
15893. func is NULL for lightfunc
15894. must be, since env was created when catcher was created
15895. note: any entries above the catchstack top are garbage and not zeroed
15896. * Update preventcount
15897. avoid side effect issues
15898. current lex_env of the activation (created for catcher)
15899. Note that multiple catchstack entries may refer to the same * callstack entry.
15900. **comment:** * Manipulation of thread stacks (valstack, callstack, catchstack). * * Ideally unwinding of stacks should have no side effects, which would * then favor separate unwinding and shrink check primitives for each * stack type. A shrink check may realloc and thus have side effects. * * However, currently callstack unwinding itself has side effects, as it * needs to DECREF multiple objects, close environment records, etc. * Stacks must thus be unwound in the correct order by the caller. * * (XXX: This should be probably reworked so that there is a shared * unwind primitive which handles all stacks as requested, and knows * the proper order for unwinding.) * * Valstack entries above 'top' are always kept initialized to * "undefined unused". Callstack and catchstack entries above 'top' * are not zeroed and are left as garbage. * * Value stack handling is mostly a part of the API implementation.
 label: code-design
15901. * Unwind debugger state. If we unwind while stepping * (either step over or step into), pause execution.
15902. Note: any entries above the callstack top are garbage and not zeroed. * Also topmost activation idx_retval is garbage (not zeroed), and must * be ignored.

15962. DUK_OP_RETURN flags in A
15963. DUK_OP_CALL flags in A
15964. DUK_OP_DECLVAR flags in A; bottom bits are reserved for propdesc flags (DUK_PROPDESC_FLAG_XXX)
15965. * Ecmascript bytecode
15966. Constants should be signed so that signed arithmetic involving them * won't cause values to be coerced accidentally to unsigned.
15967. DUK_OP_EXTRA, sub-operation in A
15968. function declaration
15969. DUK_JS_BYTECODE_H_INCLUDED
15970. dummy value used as marker
15971. see duk_js_executor.c
15972. **comment:** XXX: incref by count (2) directly
 label: code-design
15973. Use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to an existing one).
15974. No entry in the catchstack which would actually catch a * throw can refer to the callstack entry being reused. * There *can* be catchstack entries referring to the current * callstack entry as long as they don't catch (e.g. label sites).
15975. success/error path both do this
15976. num entries of new func at entry
15977. other call
15978. no need to unwind
15979. * Forward declarations.
15980. Strict functions don't get their 'caller' updated.
15981. **comment:** XXX: don't want to shrink allocation here
 label: code-design
15982. non-strict: non-deletable, writable
15983. **comment:** The variable environment for magic variable bindings needs to be * given by the caller and recorded in the arguments object. * * See E5 Section 10.6, the creation of setters/getters. * * The variable environment also provides access to the callee, so * an explicit (internal) callee property is not needed.
 label: code-design
15984. Get valstack index for the func argument or throw if insane stack.
15985. 'i' is the first entry we'll keep
15986. Callee/caller are throwers and are not deletable etc. They * could be implemented as virtual properties, but currently * there is no support for virtual properties which are accessors * (only plain virtual properties). This would not be difficult * to change in duk_hobject_props, but we can make the throwers * normal, concrete properties just as easily. * * Note that the specification requires that the *same* thrower * built-in object is used here! See E5 Section 10.6 main * algorithm, step 14, and Section 13.2.3 which describes the * thrower. See test case test-arguments-throwers.js.
15987. tv1 is -below- valstack_bottom
15988. [args(n) [crud] formals arguments map mappednames]
15989. nuke values at idx_rebase to get the first retval (initially * at idx_rcbase) to idx_rebase
15990. * Interrupt counter fixup (for development only).
15991. Setting to NULL causes 'caller' to be set to * 'null' as desired.
15992. 'caller' must only take on 'null' or function value
15993. Call handling and success path. Success path exit cleans * up almost all state.
15994. **comment:** XXX: avoid this check somehow
 label: code-design
15995. 'func' on stack (borrowed reference)
15996. at least effective 'this'
15997. [... this_new | arg1 ... argN]
15998. never executed
15999. **comment:** Longjmp state is cleaned up by error handling
 label: code-design
16000. * Thread state check and book-keeping.
16001. bottom of current func
16002. For native calls must be NULL so we don't sync back
16003. **comment:** XXX: assume these?
 label: code-design
16004. Then reuse the unwound activation; callstack was not shrunk so there is always space
16005. **comment:** * Helper for handling an Ecmascript-to-Ecmascript * function (initial) Duktape.Thread.resume(). * * Compared to normal calls handled by duk_handle_call(), there are a * bunch of differences: * * - the call is never protected * - there is no C recursion depth increase (hence an "ignore recursion * limit" flag is not applicable) * - instead of making the call, this helper just performs the thread * setup and returns; the bytecode executor then restarts execution * internally * - ecmascript functions are never 'vararg' functions (they access * varargs through the 'arguments' object) * * The callstack of the target contains an earlier Ecmascript call in case * of an Ecmascript-to-Ecmascript call (whose idx_retval is updated), or * is empty in case of an initial Duktape.Thread.resume(). * * The first thing to do here is to figure out whether an ecma-to-ecma * call is actually possible. It's not always the case if the target is * a bound function; the final function may be native. In that case, * return an error so caller can fall back to a normal call path.
 label: code-design
16006. use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to the existing one)
16007. step 11
16008. duk_tval ptr for 'func' on stack (borrowed reference) or tv_func_copy
16009. Note: either pointer may be NULL (at entry), so don't assert
16010. **comment:** XXX: unify handling with native call.
 label: code-design
16011. Lightfuncs are always native functions and have "newenv".
16012. **comment:** XXX: we should never shrink here; when we error out later, we'd * need to potentially grow the value stack in error unwind which could * cause another error.
 label: code-design
16013. XXX: byte offset?
16014. first call
16015. may need unwind
16016. * Helper for updating callee 'caller' property.
16017. side effects
16018. bound chain resolved
16019. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * Lightfunc detection happens here too. Note that lightweight functions * can be wrapped by (non-lightweight) bound functions so we must resolve * the bound function chain first.
16020. If the debugger is active we need to force an interrupt so that * debugger breakpoints are rechecked. This is important for function * calls caused by side effects (e.g. when doing a DUK_OP_GETPROP), see * GH-303. Only needed for success path, error path always causes a * breakpoint recheck in the executor. It would be enough to set this * only when returning to an Ecmascript activation, but setting the flag * on every return should have no ill effect.
16021. **comment:** XXX: very slow - better to bulk allocate a gap, and copy * from args_array directly (we know it has a compact array * part, etc).
 label: code-design

16022. formals for 'func' (may be NULL if func is a C function)
16023. Normal non-bound function.
16024. valstack index of start of args (arg1) (relative to entry valstack_bottom)
16025. should never happen for a strict callee
16026. Backup 'caller' property and update its value.
16027. * Valstack manipulation for results.
16028. no need to handle thread state book-keeping here
16029. out of spec, must be configurable
16030. This may only happen if built-ins are being "torn down". * This behavior is out of specification scope.
16031. same as during entry
16032. Note: careful with indices like '-x'; if 'x' is zero, it refers to bottom
16033. original input stack before nargs/nregs handling must be * intact for 'arguments' object
16034. third arg: absolute index (to entire valstack) of idx_bottom of new activation
16035. strict: non-deletable, non-writable
16036. -> [... func this arg1 ... argN _Args length]
16037. 1 = num actual 'return values'
16038. **comment:** XXX: Is this INCREF necessary? 'func' is always a borrowed * reference reachable through the value stack? If changed, stack * unwind code also needs to be fixed to match.
label: code-design
16039. current thread
16040. * Setup value stack: clamp to 'nargs', fill up to 'nregs' * * Value stack may either grow or shrink, depending on the * number of func registers and the number of actual arguments. * If nregs >= 0, func wants args clamped to 'nargs'; else it * wants all args (= 'num_stack_args').
16041. Currently the bytecode executor and executor interrupt * instruction counts are off because we don't execute the * interrupt handler when we're about to exit from the initial * user call into Duktape. * * If we were to execute the interrupt handler here, the counts * would match. You can enable this block manually to check * that this is the case.
16042. Duktape/C API guaranteed entries (on top of args)
16043. Success path.
16044. XXX: We can't resize the value stack to a size smaller than the * current top, so the order of the resize and adjusting the stack * top depends on the current vs. final size of the value stack. * The operations could be combined to avoid this, but the proper * fix is to only grow the value stack on a function call, and only * shrink it (without throwing if the shrink fails) on function * return.
16045. bound function chain has already been resolved
16046. base of known return values
16047. **comment:** * Call handling. * * Main functions are: * * - duk_handle_call_unprotected(): unprotected call to Ecmascript or * Duktape/C function * - duk_handle_call_protected(): protected call to Ecmascript or * Duktape/C function * - duk_handle_safe_call(): make a protected C call within current * activation * - duk_handle_ecma_call_setup(): Ecmascript-to-Ecmascript calls * (not always possible), including tail calls and coroutine resume * * See 'execution.rst'. * * Note: setjmp() and local variables have a nasty interaction, * see execution.rst; non-volatile locals modified after setjmp() * call are not guaranteed to keep their value.
label: code-design
16048. step 11.c is relevant only if non-strict (checked in 11.c.ii)
16049. * Manipulate valstack so that args are on the current bottom and the * previous caller's 'this' binding (which is the value preceding the * current bottom) is replaced with the new 'this' binding: * * [... this_old | (crud) func this_new arg1 ... argN] * --> [... this_new | arg1 ... argN] * * For tail calling to work properly, the valstack bottom must not grow * here; otherwise crud would accumulate on the valstack.
16050. Return value handling.
16051. **comment:** XXX: because we unwind stacks above, thr->heap->curr_thread is at * risk of pointing to an already freed thread. This was indeed the * case in test-bug-multithread-valgrind.c, until duk_handle_call() * was fixed to restore thr->heap->curr_thread before rethrowing an * uncaught error.
label: code-design
16052. **comment:** XXX: currently NULL allocations are not supported; remove if later allowed
label: code-design
16053. flags
16054. Unwind the topmost callstack entry before reusing it
16055. borrowed, no refcount
16056. same thread
16057. may be NULL
16058. pop 'name'
16059. Other longjmp types are handled by executor before propagating * the error here.
16060. * Bytecode executor call. * * Execute bytecode, handling any recursive function calls and * thread resumptions. Returns when execution would return from * the entry level activation. When the executor returns, a * single return value is left on the stack top. * * The only possible longjmp() is an error (DUK_LJ_TYPE_THROW), * other types are handled internally by the executor.
16061. **comment:** XXX: duk_get_length?
label: code-design
16062. side effects (currently none though)
16063. [... formals arguments map mappedNames]
16064. # argument registers target function wants (< 0 => "as is")
16065. For now all calls except Ecma-to-Ecma calls prevent a yield.
16066. duk_hthread_callstack_unwind() will decrease this on unwind
16067. [... formals]
16068. See: tests/ecmascript/test-spec-bound-constructor.js
16069. * duk_handle_call_protected() and duk_handle_call_unprotected(): * call into a Duktape/C or an Ecmascript function from any state. * * Input stack (thr): * * [func this arg1 ... argN] * * Output stack (thr): * * [retval] (DUK_EXEC_SUCCESS) * [errobj] (DUK_EXEC_ERROR (normal error), protected call) * * Even when executing a protected call an error may be thrown in rare cases * such as an insane num_stack_args argument. If there is no catchpoint for * such errors, the fatal error handler is called. * * The error handling path should be error free, even for out-of-memory * errors, to ensure safe sandboxing. (As of Duktape 1.4.0 this is not * yet the case, see XXX notes below.)
16070. * Init argument related properties
16071. Tolerate act_caller->func == NULL which happens in * some finalization cases; treat like unknown caller.
16072. **comment:** * duk_handle_safe_call(): make a "C protected call" within the * current activation. * * The allowed thread states for making a call are the same as for * duk_handle_call_xxx(). * * Error handling is similar to duk_handle_call_xxx(); errors may be thrown * (and result in a fatal error) for insane arguments.
label: code-design
16073. **comment:** XXX: remove DUK_CALL_FLAG_IGNORE_RECLIMIT flag: there's now the * reclimit bump?
label: code-design
16074. DUK_USE_ASSERTIONS
16075. **comment:** XXX: inefficient; block remove primitive
label: code-design
16076. Note: multiple threads may be simultaneously in the RUNNING * state, but not in the same "resume chain".
16077. Lightfuncs are always considered strict.
16078. Ensure space for final configuration (idx_retbases + num_stack_rets) * and intermediate configurations.
16079. **comment:** Note: nargs (and nregs) may be negative for a native, * function, which indicates that the function wants the * input stack "as is" (i.e. handles "vararg" arguments).
label: code-design

16080. 'act' already set above
16081. * Setup value stack: clamp to 'nargs', fill up to 'nregs'
16082. lightfuncs are treated like objects and not coerced
16083. [...] func this | arg1 ... argN] ('this' must precede new bottom)
16084. * Make the C call
16085. **comment:** XXX: rework
 label: code-design
16086. index
16087. act_caller->func may be NULL in some finalization cases, * just treat like we don't know the caller.
16088. XXX: Multiple tv_func lookups are now avoided by making a local * copy of tv_func. Another approach would be to compute an offset * for tv_func from valstack bottom and recomputing the tv_func * pointer quickly as valstack + offset instead of calling duk_get_tval().
16089. **comment:** XXX: this should be an assert
 label: test
16090. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur. If we end up not making the * call we must restore the value.
16091. We don't need to sync back thr->ptr_curr_pc here because * the bytecode executor always has a setjmp catchpoint which * does that before errors propagate to here.
16092. [...] | (crud) errobj]
16093. Start filling in the activation
16094. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
 label: code-design
16095. assumed to bottom relative
16096. Previous value doesn't need refcount changes because its ownership * is transferred to prev_caller.
16097. Preliminaries, required by setjmp() handler. Must be careful not * to throw an unintended error here.
16098. * Manipulate value stack so that exactly 'num_stack_rets' return * values are at 'idx_retbase' in every case, assuming there are * 'rc' return values on top of stack. * This is a bit tricky, because the called C function operates in * the same activation record and may have e.g. popped the stack * empty (below idx_retbase).
16099. Exotic behaviors are only enabled for arguments objects * which have a parameter map (see E5 Section 10.6 main * algorithm, step 12). * * In particular, a non-strict arguments object with no * mapped formals does *NOT* get exotic behavior, even * for e.g. "caller" property. This seems counterintuitive * but seems to be the case.
16100. [...] func this <some bound args> arg1 ... argN _Args]
16101. really 'not applicable' anymore, should not be referenced after this
16102. idx of first argument on stack
16103. thr->ptr_curr_pc is set by bytecode executor early on entry
16104. Chop extra retvals away / extend with undefined.
16105. setjmp catchpoint setup
16106. [...] errobj]
16107. Longjmp state is kept clean in success path
16108. **comment:** * Recursion limit check. * * Note: there is no need for an "ignore recursion limit" flag * for duk_handle_safe_call now.
 label: code-design
16109. final configuration
16110. Automatic error throwing, retval check.
16111. no need to decref previous value
16112. Explicit check for fastint downgrade.
16113. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur.
16114. refer to callstack entries below current
16115. Note: not a valid stack index if num_stack_args == 0
16116. tailcall cannot be flagged to resume calls, and a * previous frame must exist
16117. * Shared helper for non-bound func lookup. * * Returns duk_hobject * to the final non-bound function (NULL for lightfunc).
16118. * Preliminary activation record and valstack manipulation. * The concrete actions depend on whether we're dealing * with a tail call (reuse an existing activation), a resume, * or a normal call. * * The basic actions, in varying order, are: * * - Check stack size for call handling * - Grow call stack if necessary (non-tail-calls) * - Update current activation (idx_retval) if necessary * (non-tail, non-resume calls) * - Move start of args (idx_args) to valstack bottom * (tail calls) * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
16119. must be updated to work properly (e.g. creation of 'arguments')
16120. write back
16121. extend with undefined
16122. **comment:** * Helper for handling a "bound function" chain when a call is being made. * * Follows the bound function chain until a non-bound function is found. * Prepends the bound arguments to the value stack (at idx_func + 2), * updating 'num_stack_args' in the process. The 'this' binding is also * updated if necessary (at idx_func + 1). Note that for constructor calls * the 'this' binding is never updated by [[BoundThis]]. * * XXX: bound function chains could be collapsed at bound function creation * time so that each bound function would point directly to a non-bound * function. This would make call time handling much easier.
 label: code-design
16123. * Helper for setting up var_env and lex_env of an activation, * assuming it does NOT have the DUK_HOBJECT_FLAG_NEWENV flag.
16124. should actually never happen, but check anyway
16125. Use a new environment but there's no 'arguments' object; * delayed environment initialization. This is the most * common case.
16126. Use loop to minimize code size of relookup after bound function case
16127. **comment:** XXX: should this happen in the callee's activation or after unwinding?
 label: code-design
16128. don't want an intermediate exposed state with invalid pc
16129. catchstack limits
16130. # total registers target function wants on entry (< 0 => never for ecma calls)
16131. idx_argbase
16132. * Value stack resize and stack top adjustment helper. * * XXX: This should all be merged to duk_valstack_resize_raw().
16133. don't want an intermediate exposed state with func == NULL
16134. Error; error value is in heap->jl.value1.
16135. * Store entry state.
16136. **comment:** Note: cannot portably debug print a function pointer, hence 'func' not printed!
 label: code-design
16137. [...] retval]
16138. no incref
16139. may have side effects
16140. we're running inside the caller's activation, so no change in call/catch stack or valstack bottom
16141. if a tail call: * - an Ecmascript activation must be on top of the callstack * - there cannot be any active catchstack entries
16142. NULL for lightfunc
16143. Use a new environment and there's an 'arguments' object. * We need to initialize it right now.
16144. Lightweight function: never bound, so terminate.
16145. # argument registers target function wants (< 0 => never for ecma calls)
16146. compiler ensures this

16147. internal spare
16148. already NULLed (by unwind)
16149. * Misc shared helpers.
16150. out of spec, don't care
16151. [... func this arg1 ... argN envobj]
16152. + spare
16153. keep current valstack_top
16154. XXX: optimize value stack operation
16155. bottom of new func
16156. can't happen when keeping current stack size
16157. Ensure there is internal valstack spare before we exit; this may * throw an alloc error. The same guaranteed size must be available * as before the call. This is not optimal now: we store the valstack * allocated size during entry; this value may be higher than the * minimal guarantee for an application.
16158. steps 11.c.ii.1 - 11.c.ii.4, but our internal book-keeping * differs from the reference model
16159. XXX: byte offset
16160. may be changed by call
16161. DUK_USE_NONSTD_FUNC_CALLER_PROPERTY
16162. topmost activation idx_retval is considered garbage, no need to init
16163. [args [crud] arguments]
16164. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. ** If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. ** If the final target function cannot be handled by an ecma-to-ecma * call, return to the caller with a return value indicating this case. * The bound chain is resolved and the caller can resume with a plain * function call.
16165. **comment:** XXX: some overlapping code; cleanup
label: code-design
16166. must be signed for loop structure
16167. [... func this <bound args> arg1 ... argN]
16168. vararg
16169. no need to create environment record now; leave as NULL
16170. [... arg1 ... argN]
16171. caller must check
16172. set exotic behavior only after we're done
16173. [... name]
16174. [... | (crud)]
16175. [... arg1 ... argN envobj]
16176. nothing to incref
16177. Note: may be NULL if first call
16178. num args starting from idx_argbase
16179. check that the valstack has space for the final amount and any * intermediate space needed; this is unoptimal but should be safe
16180. If caller is global/eval code, 'caller' should be set to * 'null'. ** XXX: there is no exotic flag to infer this correctly now. * The NEWENV flag is used now which works as intended for * everything (global code, non-strict eval code, and functions) * except strict eval code. Bound functions are never an issue * because 'func' has been resolved to a non-bound function.
16181. valstack index of 'func' and retval (relative to entry valstack_bottom)
16182. **comment:** These base values are never used, but if the compiler doesn't know * that DUK_ERROR() won't return, these are needed to silence warnings. * On the other hand, scan-build will warn about the values not being * used, so add a DUK_UNREF.
label: code-design
16183. [... |] or [... | errobj (M * undefined)] where M = num_stack_rets - 1
16184. Restore entry thread executor curr_pc stack frame pointer.
16185. idx_args = idx_func + 2
16186. On entry, item at idx_func is a bound, non-lightweight function, * but we don't rely on that below.
16187. Success path handles
16188. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** This handling is now identical for C and Ecmascript functions. * C functions always have the 'NEWENV' flag set, so their * environment record initialization is delayed (which is good). ** Delayed creation (on demand) is handled in duk_js_var.c.
16189. XXX: bump preventcount by one for the duration of this call?
16190. stack discipline consistency check
16191. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** Delayed creation (on demand) is handled in duk_js_var.c.
16192. See: test-bug-tailcall-preventyield-assert.c.
16193. insert 'undefined' values at idx_rcbase to get the * return values to idx_retbase
16194. [... func this (crud) retval]
16195. XXX: assert? compiler is responsible for this never happening
16196. Helper for creating the arguments object and adding it to the env record * on top of the value stack. This helper has a very strict dependency on * the shape of the input stack.
16197. * Shift to new valstack_bottom.
16198. **comment:** XXX: unnecessary, handle in adjust
label: code-design
16199. already updated
16200. to avoid relookups
16201. **comment:** * Update idx_retval of current activation. ** Although it might seem this is not necessary (bytecode executor * does this for Ecmascript-to-Ecmascript calls; other calls are * handled here), this turns out to be necessary for handling yield * and resume. For them, an Ecmascript-to-native call happens, and * the Ecmascript call's idx_retval must be set for things to work.
label: code-design
16202. Note: either pointer may be NULL (at entry), so don't assert.
16203. * Create required objects: * - 'arguments' object: array-like, but not an array * - 'map' object: internal object, tied to 'arguments' * - 'mappedNames' object: temporary value used during construction
16204. [... func this arg1 ... argN]
16205. replace in stack
16206. **comment:** XXX: check .caller writability?
label: code-design
16207. Since stack indices are not reliable, we can't do anything useful * here. Invoke the existing setjmp catcher, or if it doesn't exist, * call the fatal error handler.
16208. duk_tval ptr for 'func' on stack (borrowed reference)
16209. tail call -> reuse current "frame"
16210. Unwind catchstack entries referring to the callstack entry we're reusing
16211. local copy to avoid relookups
16212. **comment:** * Error during call. The error value is at heap->jl.value1. ** The very first thing we do is restore the previous setjmp catcher. * This means that any error in error handling will propagate outwards * instead of causing a setjmp() re-entry above.
label: code-design

16213. * Determine call type, then finalize activation, shift to * new value stack bottom, and call the target.
16214. * Arguments object creation. ** Creating arguments objects involves many small details, see E5 Section * 10.6 for the specific requirements. Much of the arguments object exotic * behavior is implemented in duk_hobject_props.c, and is enabled by the * object flag DUK_HOBJECT_FLAG_EXOTIC_ARGUMENTS.
16215. 'arguments'
16216. **comment:** XXX: remove the preventcount and make yield walk the callstack? * Or perhaps just use a single flag, not a counter, faster to just * set and restore?
 label: code-design
16217. at least errobj must be on stack
16218. **comment:** XXX: inefficient; block insert primitive
 label: code-design
16219. **comment:** XXX: space in valstack? see discussion in duk_handle_call_xxx().
 label: code-design
16220. **comment:** * Tailcall handling ** Although the callstack entry is reused, we need to explicitly unwind * the current activation (or simulate an unwind). In particular, the * current activation must be closed, otherwise something like * test-bug-reduce-judofy.js results. Also catchstack needs be unwound * because there may be non-error-catching label entries in valid tail calls.
 label: code-design
16221. Unwind.
16222. Restore the previous setjmp catcher so that any error in * error handling will propagate outwards rather than re-enter * the same handler. However, the error handling path must be * designed to be error free so that sandboxing guarantees are * reliable, see e.g. <https://github.com/svaarala/duktape/issues/476>.
16223. * Setup a preliminary activation and figure out nargs/nregs. ** Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
16224. Error path.
16225. **comment:** XXX: could be eliminated with valstack adjust
 label: code-design
16226. * Ecmascript call
16227. clamp anything above nargs
16228. **comment:** XXX: this could be more compact by accessing the internal properties * directly as own properties (they cannot be inherited, and are not * externally visible).
 label: code-design
16229. Argument validation and func/args offset.
16230. # total registers target function wants on entry (< 0 => "as is")
16231. cannot be strict (never mapped variables)
16232. * C call recursion depth check, which provides a reasonable upper * bound on maximum C stack size (arbitrary C stack growth is only * possible by recursive handle_call / handle_safe_call calls).
16233. no compact
16234. XXX: callstack unwind may now throw an error when closing * scopes; this is a sandboxing issue, described in: * <https://github.com/svaarala/duktape/issues/476>
16235. * Valstack spare check
16236. * Determine the effective 'this' binding and coerce the current value * on the valstack to the effective one (in-place, at idx_this). ** The current this value in the valstack (at idx_this) represents either: * - the caller's requested 'this' binding; or * - a 'this' binding accumulated from the bound function chain * * The final 'this' binding for the target function may still be * different, and is determined as described in E5 Section 10.4.3. ** For global and eval code (E5 Sections 10.4.1 and 10.4.2), we assume * that the caller has provided the correct 'this' binding explicitly * when calling, i.e.: * * - global code: this=global object * - direct eval: this=copy from eval() caller's this binding * - other eval: this=global object * * Note: this function may cause a recursive function call with arbitrary * side effects, because ToObject() may be called.
16237. 'func' wants stack "as is"
16238. * Init arguments properties, map, etc.
16239. name
16240. [... arg1 ... argN envobj argobj]
16241. act->func
16242. These are just convenience "wiping" of state. Side effects should * not be an issue here: thr->heap and thr->heap->jl have a stable * pointer. Finalizer runs etc capture even out-of-memory errors so * nothing should throw here.
16243. idx_this = idx_func + 1
16244. or a non-catching entry
16245. **comment:** XXX: tv_func is not actually needed
 label: requirement
16246. XXX: this doesn't actually work properly for tail calls, so * tail calls are disabled when DUK_USE_NONSTD_FUNC_CALLER_PROPERTY * is in use.
16247. different thread
16248. **comment:** Function.prototype.bind() should never let this happen, * ugly error message is enough.
 label: code-design
16249. **comment:** XXX: this needs to be reworked so that we never shrink the value * stack on function entry so that we never need to grow it here. * Needing to grow here is a sandboxing issue because we need to * allocate which may cause an error in the error handling path * and thus propagate an error out of a protected call.
 label: code-design
16250. step 12
16251. [... name name]
16252. -> [... func this arg1 ... argN _Args]
16253. steps 13-14
16254. no prototype
16255. leave formals on stack for later use
16256. * Return to bytecode executor, which will resume execution from * the topmost activation.
16257. Note: 'func' is popped from valstack here, but it is * already reachable from the activation.
16258. [... func this arg1 ... argN] (not tail call) * [this | arg1 ... argN] (tail call) * * idx_args updated to match
16259. * Native call.
16260. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
 label: code-design
16261. duk_has_prop() popped the second 'name'
16262. A debugger forced interrupt check is not needed here, as * problematic safe calls are not caused by side effects.
16263. **comment:** XXX: block removal API primitive
 label: code-design
16264. The final object may be a normal function or a lightfunc. * We need to re-lookup tv_func because it may have changed * (also value stack may have been resized). Loop again to * do that; we're guaranteed not to come here again.
16265. **comment:** * Cleanups (all statement parsing flows through here). ** Pop label site and reset labels. Reset 'next temp' to value at * entry to reuse temps.
 label: code-design
16266. * Rewind lexer. ** duk_parse_stmts() expects curr Tok to be set; parse in "allow regexp * literal" mode with current strictness. ** curr_token line number info should be initialized for pass 2 before * generating prologue, to ensure prologue bytecode gets nice line numbers.
16267. only functions can have arguments
16268. **comment:** * Ecmascript compiler. ** Parses an input string and generates a function template result. * Compilation may happen in multiple contexts (global code, eval * code, function code). ** The parser uses a traditional top-down recursive parsing for the * statement level, and an operator precedence based top-down approach * for the expression level. The attempt is to minimize the C stack * depth. Bytecode is generated directly without an intermediate * representation (tree), at the cost of needing two passes over each * function. ** The top-down recursive parser functions are named "duk_parse_XXX". ** Recursion limits are in key

functions to prevent arbitrary C recursion: * function body parsing, statement parsing, and expression parsing. * * See doc/compiler.rst for discussion on the design. * * A few typing notes: * * - duk_regconst_t: unsigned, no marker value for "none" * - duk_reg_t: signed, < 0 = none * - PC values: duk_int_t, negative values used as markers

label: code-design

16269. DUK_TOK_SNEQ

16270. DUK_TOK_TRUE

16271. * Variant 2

16272. UnaryExpression

16273. Buffer length is bounded to 0xffff automatically, avoid compile warning.

16274. **comment:** * Shared handling for logical AND and logical OR. * * args = (truthval << 8) + rbp * * Truthval determines when to skip right-hand-side. * For logical AND truthval=1, for logical OR truthval=0. * * See doc/compiler.rst for discussion on compiling logical * AND and OR expressions. The approach here is very simplistic, * generating extra jumps and multiple evaluations of truth values, * but generates code on-the-fly with only local back-patching. * * Both logical AND and OR are syntactically left-associated. * However, logical ANDs are compiled as right associative * expressions, i.e. "A && B && C" as "A && (B && C)", to allow * skip jumps to skip over the entire tail. Similarly for logical OR.

label: code-design

16275. No need to assert, buffer size maximum is 0xffff.

16276. Tailcalls are handled by back-patching the TAILCALL flag to the * already emitted instruction later (in return statement parser). * Since A and C have a special meaning here, they cannot be "shuffled".

16277. inserted jump

16278. curr_token follows 'function'

16279. Output shuffle needed after main operation

16280. DUK_TOK_TRY

16281. evaluate to final form (e.g. coerce GETPROP to code), throw away temp

16282. push before advancing to keep reachable

16283. DUK_TOK_GET

16284. init function state: inits valstack allocations

16285. for storing/restoring the varmap binding for catch variable

16286. convenience helpers

16287. * Build function fixed size 'data' buffer, which contains bytecode, * constants, and inner function references. * * During the building phase 'data' is reachable but incomplete. * Only incref's occur during building (no refzero or GC happens), * so the building process is atomic.

16288. omitted

16289. Check statement type based on the first token type. * * Note: expression parsing helpers expect 'curr_tok' to * contain the first token of the expression upon entry.

16290. -> pushes fixed buffer

16291. Not safe to use 'reg_varbind' as assignment expression * value, so go through a temp.

16292. Ensure compact use of temps.

16293. default case exists: go there if no case matches

16294. **comment:** Temporary structure used to pass a stack allocated region through * duk_safe_call().

label: code-design

16295. eat 'do'

16296. * Do-while statement is mostly trivial, but there is special * handling for automatic semicolon handling (triggered by the * DUK_ALLOW_AUTO_SEMI_ALWAYS flag related to a bug filed at: * * https://bugs.ecmascript.org/show_bug.cgi?id=8 * * See doc/compiler.rst for details.

16297. -> [... name index]

16298. max # of values initialized in one MPUTARR set

16299. * Reset function state and perform register allocation, which creates * 'varmap' for second pass. Function prologue for variable declarations, * binding value initializations etc is emitted as a by-product. * * Strict mode restrictions for duplicate and invalid argument * names are checked here now that we know whether the function * is actually strict. See: test-dev-strict-mode-boundary.js. * * Inner functions are compiled during pass 1 and are not reset.

16300. Strict equality is NOT enough, because we cannot use the same * constant for e.g. +0 and -0.

16301. e.g. DUK_OP_PREINCV

16302. reuse 'res' as 'left'

16303. Initial bytecode size allocation.

16304. XXX: patch initial size afterwards?

16305. yes, must set array length explicitly

16306. in-place setup

16307. Does not assume that jump_pc contains a DUK_OP_JUMP previously; this is intentional * to allow e.g. an INVALID opcode be overwritten with a JUMP (label management uses this).

16308. **comment:** Delete semantics are a bit tricky. The description in E5 specification * is kind of confusing, because it distinguishes between resolvability of * a reference (which is only known at runtime) seemingly at compile time * (= SyntaxError throwing).

label: code-design

16309. binding power must be high enough to NOT allow comma expressions directly

16310. flags

16311. _Formals: omitted if function is guaranteed not to need a (non-strict) arguments object

16312. Note: these variables must reside in 'curr_func' instead of the global * context: when parsing function expressions, expression parsing is nested.

16313. implicit_return_value

16314. * Initialize function state for a zero-argument function

16315. * Special name handling

16316. eat 'for'

16317. right associative

16318. init 'curr_token'

16319. allow a constant to be returned

16320. DUK_TOK_LBRACKET

16321. XXX: check num_args

16322. DUK_TOK_LPAREN

16323. directly uses slow accesses

16324. **comment:** It'd be nice to do something like this - but it doesn't * work for closures created inside the catch clause.

label: code-design

16325. duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp literal" mode with current strictness

16326. **comment:** XXX: Shuffling support could be implemented here so that LDINT+LDINTX * would only shuffle once (instead of twice). The current code works * though, and has a smaller compiler footprint.

label: code-design

16327. * Post-increment/decrement will return the original value as its * result value. However, even that value will be coerced using * ToNumber() which is quite awkward. Specific bytecode opcodes * are used to handle these semantics. * * Note that post increment/decrement has a "no LineTerminator here" * restriction. This is handled by duk_expr_lbp(), which forcibly terminates * the previous expression if a LineTerminator occurs before '++'/'--'.

16328. DUK_TOK_LT

16329. If duk_ivalue_toplain_raw() allocates a temp, forget it and * restore next temp state.

16330. catch depth from current func

16331. eat colon

16332. **comment:** XXX: since the enumerator may be a memory expensive object, * perhaps clear it explicitly here? If so, break jump must * go through this clearing operation.

label: code-design

16333. [... pattern flags]

16334. * Init compiler and lexer contexts

16335. DUK_TOK_NEQ

16336. * Code emission helpers * * Some emission helpers understand the range of target and source reg/const * values and automatically emit shuffling code if necessary. This is the * case when the slot in question (A, B, C) is used in the standard way and * for opcodes the emission helpers explicitly understand (like DUK_OP_CALL). * * The standard way is that: * - slot A is a target register * - slot B is a source register/constant * - slot C is a source register/constant * * If a slot is used in a non-standard way the caller must indicate this * somehow. If a slot is used as a target instead of a source (or vice * versa), this can be indicated with a flag to trigger proper shuffling * (e.g. DUK_EMIT_FLAG_B_IS_TARGET). If the value in the slot is not * register/const related at all, the caller must ensure that the raw value * fits into the corresponding slot so as to not trigger shuffling. The * caller must set a "no shuffle" flag to ensure compilation fails if * shuffling were to be triggered because of an internal error. * * For slots B and C the raw slot size is 9 bits but one bit is reserved for * the reg/const indicator. To use the full 9-bit range for a raw value, * shuffling must be disabled with the DUK_EMIT_FLAG_NO_SHUFFLE_{B,C} flag. * Shuffling is only done for A, B, and C slots, not the larger BC or ABC slots. * * There is call handling specific understanding in the A-B-C emitter to * convert call setup and call instructions into indirect ones if necessary.

16337. if num_values > 0, MPUTARR emitted by outer loop after break

16338. **comment:** XXX: would be nice to omit this jump when the jump is not * reachable, at least in the obvious cases (such as the case * ending with a 'break'. * * Perhaps duk_parse_stmt() could provide some info on whether * the statement is a "dead end"? * * If implemented, just set pc_prevstmt to -1 when not needed.

label: code-design

16339. **comment:** Two temporaries are preallocated here for variants 3 and 4 which need * registers which are never clobbered by expressions in the loop * (concretely: for the enumerator object and the next enumerated value). * Variants 1 and 2 "release" these temps.

label: code-design

16340. DUK_USE_NONSTD_FUNC_STMT

16341. DUK_TOK_STRING

16342. stmt has non-empty value

16343. for array and object literals

16344. When LHS is not register bound, always go through a * temporary. No optimization for top level assignment.

16345. non-strict eval: env is caller's env or global env (direct vs. indirect call) * global code: env is global env

16346. **comment:** XXX: use the exactly same arithmetic function here as in executor

label: code-design

16347. -1 == not set, -2 == pending (next statement list)

16348. Explicit check for fastint downgrade.

16349. DUK_TOK_RETURN

16350. **comment:** _Varmap: omitted if function is guaranteed not to do slow path identifier * accesses or if it would turn out to be empty of actual register mappings * after a cleanup. When debugging is enabled, we always need the varmap to * be able to lookup variables at any point.

label: code-design

16351. We could rely on max temp/const checks: if they don't exceed BC * limit, nothing here can either (just asserts would be enough). * Currently we check for the limits, which provides additional * protection against creating invalid bytecode due to compiler * bugs.

16352. **comment:** XXX: Currently function source code is not stored, as it is not * required by the standard. Source code should not be stored by * default (user should enable it explicitly), and the source should * probably be compressed with a trivial text compressor; average * compression of 20-30% is quite easy to achieve even with a trivial * compressor (RLE + backwards lookup). * * Debugging needs source code to be useful: sometimes input code is * not found in files as it may be generated and then eval()'d, given * by dynamic C code, etc. * * Other issues: * * - Need tokenizer indices for start and end to substring * - Always normalize function declaration part? * - If we keep _Formals, only need to store body

label: code-design

16353. require initializer for var/const

16354. DUK_TOK_FALSE

16355. fall-through control flow patchup; note that pc_prevstmt may be * < 0 (i.e. no case clauses), in which case this is a no-op.

16356. * Detect iteration statements; if encountered, establish an * empty label.

16357. **comment:** Should never happen but avoid infinite loop just in case.

label: code-design

16358. indirect opcode follows direct

16359. statement parsing

16360. * Function declarations

16361. **comment:** * Parse variant 1 or 2. The first part expression (which differs * in the variants) has already been parsed and its code emitted. * * reg_temps + 0: unused * reg_temps + 1: unused

label: code-design

16362. Valstack should suffice here, required on function valstack init

16363. reg_res should be smallest possible

16364. pushes function template

16365. **comment:** Pick a destination register. If either base value or key * happens to be a temp value, reuse it as the destination. * * XXX: The temp must be a "mutable" one, i.e. such that no * other expression is using it anymore. Here this should be * the case because the value of a property access expression * is neither the base nor the key, but the lookup result.

label: code-design

16366. Note: although there is no 'undefined' literal, undefined * values can occur during compilation as a result of e.g. * the 'void' operator.

16367. Property access expressions are critical for correct LHS ordering, * see comments in duk_expr()!

16368. Identifier, i.e. don't allow reserved words

16369. **comment:** * Emit a final RETURN. * * It would be nice to avoid emitting an unnecessary "return" opcode * if the current PC is not reachable. However, this cannot be reliably * detected; even if the previous instruction is an unconditional jump, * there may be a previous jump which jumps to current PC (which is the * case for iteration and conditional statements, for instance).

label: code-design

16370. **comment:** Parse an inner function, adding the function template to the current function's * function table. Return a function number to be used by the outer function. * * Avoiding O(depth^2) inner function parsing is handled here. On the first pass, * compile and register the function normally into the 'funcs' array, also recording * a lexer point (offset/line) to the closing brace of the function. On the second * pass, skip the function and return the same 'fnum' as on the first pass by using * a running counter. * * An unfortunate side effect of this is that when parsing the inner function, almost * nothing is known of the outer function, i.e. the inner function's scope. We don't * need that information at the moment, but it would allow some optimizations if it * were used.

label: code-design

16371. Slot B

16372. **comment:** * Emit initializers in sets of maximum max_init_values. * Corner cases such as single value initializers do not have * special handling now. * * Elided elements must not be emitted as 'undefined' values, * because such values would be enumerable (which is incorrect). * Also note that trailing elisions must be reflected in the * length of the final array but cause no elements to be actually * inserted.

label: code-design

16373. XXX: to util

16374. Specific assumptions for opcode numbering.

16375. _Pc2line

16376. EQUALITY EXPRESSION

16377. register declarations in first pass

16378. DUK_TOK_EQUALSIGN

16379. **comment:** No catch variable, e.g. a try-finally; replace LDCONST with * NOP to avoid a bogus LDCONST.

label: code-design

16380. useful for patching jumps later
16381. name
16382. -> [... func name]
16383. inherit initial strictness from parent
16384. **comment:** XXX: this illustrates that a C catchpoint implemented using duk_safe_call() * is a bit heavy at the moment. The wrapper compiles to ~180 bytes on x64. * Alternatives would be nice.
label: code-design
16385. DUK_TOK_WHILE
16386. Use special opcodes to load short strings
16387. temp variable to pass constants and flags to shared code
16388. truncated in case pass 3 needed
16389. reg
16390. * Second (and possibly third) pass. * * Generate actual code. In most cases the need for shuffle * registers is detected during pass 1, but in some corner cases * we'll only detect it during pass 2 and a third pass is then * needed (see GH-115).
16391. DUK_TOK_MUL_EQ
16392. **comment:** maintain highest 'used' temporary, needed to figure out nregs of function
label: code-design
16393. Slot C is used in a non-standard fashion (range of regs), * emitter code has special handling for it (must not set the * "no shuffle" flag).
16394. Catch attempts to use out-of-range reg/const. Without this * check Duktape 0.12.0 could generate invalid code which caused * an assert failure on execution. This error is triggered e.g. * for functions with a lot of constants and a try-catch statement. * Shuffling or opcode semantics change is needed to fix the issue. * See: test-bug-trycatch-many-constants.js.
16395. DUK_TOK_DO
16396. **comment:** XXX: match flag is awkward, rework
label: code-design
16397. continue matched this label -- we can only continue if this is the empty * label, for which duplication is allowed, and thus there is hope of * finding a match deeper in the label stack.
16398. Evaluate RHS only when LHS is safe.
16399. DUK_TOK_VOID
16400. DUK_USE_PC2LINE
16401. Special handling for call setup instructions. The target * is expressed indirectly, but there is no output shuffling.
16402. no catchers
16403. A number can be loaded either through a constant, using * LDINT, or using LDINT+LDINTX. LDINT is always a size win, * LDINT+LDINTX is not if the constant is used multiple times. * Currently always prefer LDINT+LDINTX over a double constant.
16404. DUK_USE_NONSTD_FUNC_SOURCE_PROPERTY
16405. * Variant 4
16406. * Preliminaries
16407. is_extra
16408. Slot A
16409. terminates expression; e.g. post-increment/-decrement
16410. * Parser duk__advance() token eating functions
16411. inline string concatenation
16412. no operators
16413. * Variable declarations. * * Unlike function declarations, variable declaration values don't get * assigned on entry. If a binding of the same name already exists, just * ignore it silently.
16414. Strict: never allow function declarations outside top level.
16415. **comment:** * XXX: for now, indicate that an expensive catch binding * declarative environment is always needed. If we don't * need it, we don't need the const_varname either.
label: code-design
16416. **comment:** cleanup varmap from any null entries, compact it, etc; returns number * of final entries after cleanup.
label: code-design
16417. * comp_ctx->curr_func is now ready to be converted into an actual * function template.
16418. BITWISE EXPRESSIONS
16419. Used for minimal 'const': initializer required.
16420. DUK_TOK_CONST
16421. DUK_TOK_RPAREN
16422. Slot C
16423. true
16424. comma after a value, expected
16425. general temp register
16426. **comment:** XXX: common reg allocation need is to reuse a sub-expression's temp reg, * but only if it really is a temp. Nothing fancy here now.
label: code-design
16427. DUK_TOK_SUPER
16428. **comment:** * Declaration binding instantiation conceptually happens when calling a * function; for us it essentially means that function prologue. The * conceptual process is described in E5 Section 10.5. * * We need to keep track of all encountered identifiers to (1) create an * identifier-to-register map ("varmap"); and (2) detect duplicate * declarations. Identifiers which are not bound to registers still need * to be tracked for detecting duplicates. Currently such identifiers * are put into the varmap with a 'null' value, which is later cleaned up. * * To support functions with a large number of variable and function * declarations, registers are not allocated beyond a certain limit; * after that limit, variables and functions need slow path access. * Arguments are currently always register bound, which imposes a hard * (and relatively small) argument count limit. * * Some bindings in E5 are not configurable (= deletable) and almost all * are mutable (writable). Exceptions are: * * - The 'arguments' binding, established only if no shadowing argument * or function declaration exists. We handle 'arguments' creation * and binding through an explicit slow path environment record. * * - The "name" binding for a named function expression. This is also * handled through an explicit slow path environment record.
label: code-design
16429. DUK_TOK_NULL
16430. step 4.a
16431. Check whether statement list ends.
16432. Setup state. Initial ivalue is 'undefined'.
16433. DUK_TOK_STATIC
16434. first coerce to a plain value
16435. **comment:** XXX: coerce to regs? it might be better for enumeration use, where the * same PROP ivalue is used multiple times. Or perhaps coerce PROP further * there?
label: code-design
16436. res' contains expression value
16437. invalidates tv1, tv2
16438. rc_varname and reg_varbind are ignored here
16439. **comment:** Extraop arithmetic opcodes must have destination same as * first source. If second source matches destination we need * a temporary register to avoid clobbering the second source. * * XXX: change calling code to avoid this situation in most cases.
label: code-design
16440. reg(ignored)

16441. bytecode limits
16442. DUK_TOK_SEQ
16443. XXX: clarify on when and where DUK_CONST_MARKER is allowed
16444. bump up "allocated" reg count, just in case
16445. overwrite any previous binding of the same name; the effect is * that last argument of a certain name wins.
16446. A bunch of helpers (for size optimization) that combine duk_expr()/duk_exprtop() * and result conversions. ** Each helper needs at least 2-3 calls to make it worth while to wrap.
16447. DUK_TOK_INSTANCEOF
16448. * Parse an individual source element (top level statement) or a statement. ** Handles labeled statements automatically (peeling away labels before * parsing an expression that follows the label(s)). ** Upon entry, 'curr_tok' contains the first token of the statement (parsed * in "allow regexp literal" mode). Upon exit, 'curr_tok' contains the first * token following the statement (if the statement has a terminator, this is * the token after the terminator).
16449. DUK_TOK_IF
16450. evaluate to plain value, no forced register (temp/bound reg both ok)
16451. JUMP L1 omitted
16452. e.g. DUK_OP_POSTINCR
16453. **comment:** XXX: many operations actually want toforcedtemp() -- brand new temp?
 label: code-design
16454. * Debug dumping
16455. * The switch statement is pretty messy to compile. * See the helper for details.
16456. DUK_TOK_ENUM
16457. when key is NULL, value is garbage so no need to set
16458. knock back "next temp" to this whenever possible
16459. **comment:** * Parser control values for tokens. The token table is ordered by the * DUK_TOK_XXX defines. ** The binding powers are for lbp() use (i.e. for use in led() context). * Binding powers are positive for typing convenience, and bits at the * top should be reserved for flags. Binding power step must be higher * than 1 so that binding power "lbp - 1" can be used for right associative * operators. Currently a step of 2 is used (which frees one more bit for * flags).
 label: code-design
16460. * Detected a directive
16461. entry_top + 3
16462. ASSIGNMENT EXPRESSION
16463. if a site already exists, nop: max one label site per statement
16464. first value: comma must not precede the value
16465. DUK_TOK_CATCH
16466. rethrow
16467. key encountered as a plain property
16468. e.g. DUK_OP_POSTINCV
16469. * Label handling ** Labels are initially added with flags prohibiting both break and continue. * When the statement type is finally uncovered (after potentially multiple * labels), all the labels are updated to allow/prohibit break and continue.
16470. sufficient for keeping temp reg numbers in check
16471. Note: when parsing a formal list in non-strict context, e.g. * "implements" is parsed as an identifier. When the function is * later detected to be strict, the argument list must be rechecked * against a larger set of reserved words (that of strict mode). * This is handled by duk_parse_func_body(). Here we recognize * whatever tokens are considered reserved in current strictness * (which is not always enough).
16472. **comment:** Update function min/max line from current token. Needed to improve * function line range information for debugging, so that e.g. opening * curly brace is covered by line range even when no opcodes are emitted * for the line containing the brace.
 label: code-design
16473. pass2 allocation handles this
16474. DUK_TOK_IN
16475. jump to false
16476. require a short (8-bit) reg/const which fits into bytecode B/C slot
16477. **comment:** Sanity workaround for handling functions with a large number of * constants at least somewhat reasonably. Otherwise checking whether * we already have the constant would grow very slow (as it is O(N^2)).
 label: code-design
16478. **comment:** XXX: duk_set_length
 label: code-design
16479. DUK_TOK_BOR_EQ
16480. **comment:** XXX: Add a proper condition. If formals list is omitted, recheck * handling for 'length' in duk_js_push_closure(); it currently relies * on _Formals being set. Removal may need to be conditional to debugging * being enabled/disabled too.
 label: code-design
16481. NB: must accept reserved words as property name
16482. +1, right after inserted jump
16483. * After arguments, allocate special registers (like shuffling temps)
16484. * Variant 1
16485. DUK_TOK_DEFAULT
16486. temp reset is not necessary after duk_parse_stmt(), which already does it
16487. same handling for identifiers and strings
16488. e.g. DUK_OP_PREINCP
16489. decl name
16490. LOGICAL EXPRESSIONS
16491. jump is inserted here
16492. Resolve 'res' directly into the LHS binding, and use * that as the expression value if safe. If not safe, * resolve to a temp/const and copy to LHS.
16493. * Switch is pretty complicated because of several conflicting concerns: * * - Want to generate code without an intermediate representation, * i.e., in one go * * - Case selectors are expressions, not values, and may thus e.g. throw * exceptions (which causes evaluation order concerns) * * - Evaluation semantics of case selectors and default clause need to be * carefully implemented to provide correct behavior even with case value * side effects * * - Fall through case and default clauses; avoiding dead JUMPs if case * ends with an unconditional jump (a break or a continue) * * - The same case value may occur multiple times, but evaluation rules * only process the first match before switching to a "propagation" mode * where case values are no longer evaluated * * See E5 Section 12.11. Also see doc/compiler.rst for compilation * discussion.
16494. Ensure argument name is not a reserved word in current * (final) strictness. Formal argument parsing may not * catch reserved names if strictness changes during * parsing. ** We only need to do this in strict mode because non-strict * keyword are always detected in formal argument parsing.
16495. const flag for C
16496. unary minus
16497. number of pairs in current MPUTOBJ set
16498. Tear down state.
16499. **comment:** XXX: where to release temp regs in intermediate expressions? * e.g. 1+2+3 -> don't inflate temp register count when parsing this. * that particular expression temp regs can be forced here.
 label: code-design
16500. expression parsing helpers
16501. [... key_obj key]
16502. **comment:** XXX: actually single step levels would work just fine, clean up
 label: code-design

16503. one token
16504. DUK_TOK_BNOT
16505. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.
16506. * Peephole optimizer for finished bytecode. ** Does not remove opcodes; currently only straightens out unconditional * jump chains which are generated by several control structures.
16507. **comment:** alloc temp just in case, to update max temp
label: code-design
16508. is_setget
16509. DUK_TOK_COMMA
16510. Shuffle decision not changed.
16511. DUK__ALLOW_AUTO_SEMI_ALWAYS workaround
16512. temp_start+0 = key, temp_start+1 = closure
16513. forced_reg
16514. DUK_TOK_FINALLY
16515. 'data' (and everything in it) is reachable through h_res now
16516. arg2 would be clobbered so reassign it to a temp.
16517. DUK_TOK_REGEXP
16518. Note: this is correct even for default clause statements: * they participate in 'fall-through' behavior even if the * default clause is in the middle.
16519. DUK_TOK_PACKAGE
16520. * Peephole optimize JUMP chains.
16521. temps
16522. to body
16523. no match, next case
16524. * Directive prologue tracking.
16525. DUK_TOK_IDENTIFIER
16526. * Misc helpers
16527. finished jump
16528. POSTFIX EXPRESSION
16529. base
16530. DUK_TOK_PUBLIC
16531. variable name reg/const, if variable not register-bound
16532. Eating a left curly; regexp mode is allowed by left curly * based on duk_token_lbp[] automatically.
16533. Get the value represented by an duk_ispec to a register or constant. * The caller can control the result by indicating whether or not: ** (1) a constant is allowed (sometimes the caller needs the result to * be in a register) ** (2) a temporary register is required (usually when caller requires * the register to be safely mutable; normally either a bound * register or a temporary register are both OK) ** (3) a forced register target needs to be used ** Bytecode may be emitted to generate the necessary value. The return * value is either a register or a constant.
16534. NEXTENUM jump slot: executed when enum finished
16535. single string token
16536. Insert an empty jump in the middle of code emitted earlier. This is * currently needed for compiling for-in.
16537. DUK_TOK_ARSHIFT
16538. eat the right paren
16539. **comment:** XXX: possibly incorrect handling of empty expression
label: code-design
16540. **comment:** XXX: what about statements which leave a half-cooked value in 'res' * but have no stmt value? Any such statements?
label: code-design
16541. MAXVAL is inclusive
16542. "release" preallocated temps since we won't need them
16543. temp reg value for start of loop
16544. is_decl
16545. unsigned
16546. Expression, ']' terminates
16547. Don't allow a constant for the object (even for a number etc), as * it goes into the 'A' field of the opcode.
16548. no need to coerce
16549. func limits
16550. 'res' is used for "left", and 'tmp' for "right"
16551. [... template]
16552. end switch (tok)
16553. XXX: shared handling for 'duk_expr_lhs'?
16554. keep func->h_funcs; inner functions are not reparsed to avoid O(depth^2) parsing
16555. num_values and temp_start reset at top of outer loop
16556. Fix for https://github.com/svaarala/duktape/issues/155: * If 'default' is first clause (detected by pc_prevcase < 0) * we need to ensure we stay in the matching chain.
16557. allow_source_elem
16558. duplicate/invalid key checks; returns 1 if syntax error
16559. eat 'in'
16560. Name will be filled from function expression, not by caller. * This case is used by Function constructor and duk_compile() * API with the DUK_COMPILE_FUNCTION option.
16561. * Detected label
16562. * Case clause. ** Note: cannot use reg_case as a temp register (for SEQ target) * because it may be a constant.
16563. DUK_TOK_SUB
16564. Default implicit return value.
16565. elision - flush
16566. temp reg for key literal
16567. * Parse a case or default clause.
16568. **comment:** Try to optimize X <op>= Y for reg-bound * variables. Detect side-effect free RHS * narrowly by seeing whether it emits code. * If not, rewind the code emitter and overwrite * the unnecessary temp reg load.
label: code-design
16569. DUK_TOK_THIS
16570. Unlike break/continue, throw statement does not allow an empty value.
16571. DUK_TOK_VAR
16572. * Expression parsing: duk_expr_nud(), duk_expr_led(), duk_expr_lbp(), and helpers. ** - duk_expr_nud(): ("null denotation"): process prev_token as a "start" of an expression (e.g. literal) * - duk_expr_led(): ("left denotation"): process prev_token in the "middle" of an expression (e.g. operator) * - duk_expr_lbp(): ("left-binding power"): return left-binding power of curr_token
16573. max # of key-value pairs initialized in one MPUTOBJ set
16574. expect_token
16575. Note: if the reg_temp load generated shuffling * instructions, we may need to rewind more than * one instruction, so use explicit PC computation.

16576. **comment:** Function declaration for global/eval code is emitted even * for duplicates, because of E5 Section 10.5, step 5.e of * E5.1 (special behavior for variable bound to global object). * * DECLVAR will not re-declare a variable as such, but will * update the binding value.

label: code-design

16577. Stack top contains plain value

16578. make current token the previous; need to fiddle with valstack "backing store"

16579. comp_ctx->lex.input and comp_ctx->lex.input_length filled by caller

16580. >>

16581. trailing elisions?

16582. unary plus of a number is identity

16583. no comma

16584. advance to get one step of lookup

16585. right paren eaten

16586. MPUTARR emitted by outer loop

16587. DUK_TOK_NUMBER

16588. **comment:** * XXX: attempt to get the call result to "next temp" whenever * possible to avoid unnecessary register shuffles. * * XXX: CSPROP (and CSREG) can overwrite the call target register, and save one temp, * if the call target is a temporary register and at the top of the temp reg "stack".

label: code-design

16589. jump for 'finally' case or end (if no finally)

16590. * Identifier handling

16591. DUK_TOK_SET

16592. Code is not accepted before the first case/default clause

16593. Note: 'q_instr' is still used below

16594. start array index of current MPUTARR set

16595. expect_eof

16596. XXX: request a "last statement is terminal" from duk_parse_stmt() and duk_parse_stmts(); * we could avoid the last RETURN if we could ensure there is no way to get here * (directly or via a jump)

16597. [...] key_obj key key flags]

16598. rbp_flags

16599. use 'left' as a temp

16600. don't allow const

16601. Note: special DUK_EMIT_FLAG_B_IS_TARGETSOURCE * used to indicate that B is both a source and a * target register. When shuffled, it needs to be * both input and output shuffled.

16602. '++' or '--' in a post-increment/decrement position, * and a LineTerminator occurs between the operator and * the preceding expression. Force the previous expr * to terminate, in effect treating e.g. "a,b\n++" as * "a,b;++" (= SyntaxError).

16603. DUK_TOK_BAND

16604. if highest bit of a register number is set, it refers to a constant instead

16605. no need to reset temps, as we're finished emitting code

16606. -> [...]

16607. 'typeof' must handle unresolvable references without throwing * a ReferenceError (E5 Section 11.4.3). Register mapped values * will never be unresolvable so special handling is only required * when an identifier is a "slow path" one.

16608. Expression_opt

16609. * The catch variable must be updated to reflect the new allocated * register for the duration of the catch clause. We need to store * and restore the original value for the varmap entry (if any).

16610. * Parse variant 3 or 4. * * For variant 3 (e.g. "for (A in C) D;") the code for A (except the * final property/variable write) has already been emitted. The first * instruction of that code is at pc_v34_lhs; a JUMP needs to be inserted * there to satisfy control flow needs. * * For variant 4, if the variable declaration had an initializer * (e.g. "for (var A = B in C) D;") the code for the assignment * (B) has already been emitted. * * Variables set before entering here: * * pc_v34_lhs: insert a "JUMP L2" here (see doc/compiler.rst example). * reg_temps + 0: iteration target value (written to LHS) * reg_temps + 1: enumerator object

16611. still in prologue

16612. DUK_TOK_DIV_EQ

16613. * Arguments check

16614. **comment:** The line number tracking is a bit inconsistent right now, which * affects debugger accuracy. Mostly call sites emit opcodes when * they have parsed a token (say a terminating semicolon) and called * duk_advance(). In this case the line number of the previous * token is the most accurate one (except in prologue where * prev_token.start_line is 0). This is probably not 100% correct * right now.

label: code-design

16615. bc

16616. parse new token

16617. Parse enumeration target and initialize enumerator. For 'null' and 'undefined', * INITENUM will creates a 'null' enumerator which works like an empty enumerator * (E5 Section 12.6.4, step 3). Note that INITENUM requires the value to be in a * register (constant not allowed).

16618. numargs

16619. **comment:** XXX: some code might benefit from DUK_SETTEMP_IFTEMP(ctx,x)

label: code-design

16620. **comment:** Very minimal inlining to handle common idioms '!0' and '!1', * and also boolean arguments like '!false' and '!true'.

label: code-design

16621. DUK_TOK_ELSE

16622. nret

16623. See MPUTOBJ comments above.

16624. preincrement and predecrement

16625. Need a short reg/const, does not have to be a mutable temp.

16626. Expression, terminates at a ')

16627. convert duk_compiler_func into a function template, leaving the result * on top of stack.

16628. Special handling for CALL/NEW/MPUTOBJ/MPUTARR shuffling. * For each, slot B identifies the first register of a range * of registers, so normal shuffling won't work. Instead, * an indirect version of the opcode is used.

16629. DUK_TOK_FOR

16630. DUK_TOK_DEBUGGER

16631. patched later

16632. Duplicate (shadowing) labels are not allowed, except for the empty * labels (which are used as default labels for switch and iteration * statements). * * We could also allow shadowing of non-empty pending labels without any * other issues than breaking the required label shadowing requirements * of the E5 specification, see Section 12.12.

16633. **comment:** XXX: depend on available temps?

label: code-design

16634. 'reg_varbind' is the operation result and can also * become the expression value for top level assignments * such as: "var x; x += y;".

16635. last array index explicitly initialized, +1

16636. Output shuffling: only one output register is realistically possible. * * (Zero would normally be an OK marker value: if the target register * was zero, it would never be shuffled. But with DUK_USE_SHUFFLE_TORTURE * this is no longer true, so use -1 as a marker instead.)

16637. **comment:** * Three possible element formats: * 1)PropertyName : AssignmentExpression * 2) getPropertyName () { FunctionBody } * 3) setPropertyName (PropertySetParameterList) { FunctionBody } * * PropertyName can be IdentifierName (includes reserved words), a string * literal, or a number literal. Note that IdentifierName allows 'get' and * 'set' too, so we need to look ahead to the next token to distinguish: * * { get : 1 } * * and * * { get foo() { return 1 } } * { get get() { return 1 } } // 'get' as getter propertyname * * Finally, a trailing comma is allowed. * * Key name is coerced to string at compile time (and ends up as a * a string

constant) even for numeric keys (e.g. "{1:'foo'}"). * These could be emitted using e.g. LDINT, but that seems hardly * worth the effort and would increase code size.

label: code-design

16638. MPUTOBJ emitted by outer loop

16639. MultiplicativeExpression

16640. prev_token.start_offset points to the closing brace here; when skipping * we're going to reparse the closing brace to ensure semicolon insertion * etc work as expected.

16641. more initializers

16642. need side effects, not value

16643. FunctionDeclaration: not strictly a statement but handled as such. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnnum().

16644. * Check function name validity now that we know strictness. * This only applies to function declarations and expressions, * not setter/getter name. * * See: test-dev-strict-mode-boundary.js

16645. allow automatic semicolon even without lineterm (compatibility)

16646. binding power "levels" (see doc/compiler.rst)

16647. * Cleanup: restore original function, restore valstack state.

16648. DUK_TOK_QUESTION

16649. **comment:** XXX: macro smaller than call?

label: code-design

16650. Note: negative pc values are ignored when patching jumps, so no explicit checks needed

16651. result reg

16652. Coerce an duk_alue to a 'plain' value by generating the necessary * arithmetic operations, property access, or variable access bytecode. * The duk_alue argument ('x') is converted into a plain value as a * side effect.

16653. eat comma

16654. **comment:** XXX: source code property

label: code-design

16655. this is needed for regexp mode

16656. * Compile input string into an executable function template without * arguments. * * The string is parsed as the "Program" production of Ecmascript E5. * Compilation context can be either global code or eval code (see E5 * Sections 14 and 15.1.2.1). * * Input stack: [... filename] * Output stack: [... func_template]

16657. [... varname]

16658. preinitialize lexer state partially

16659. jump to next case clause

16660. const

16661. Code emission flags, passed in the 'opcode' field. Opcode + flags * fit into 16 bits for now, so use duk_small_uint.t.

16662. eat 'if'

16663. key encountered as a getter

16664. **comment:** XXX: macros

label: code-design

16665. must have catch and/or finally

16666. DUK_TOK_GT

16667. 'new' MemberExpression

16668. DUK_TOK_BXOR

16669. * Store lexer position for a later rewind

16670. regCatch+0 and regCatch+1 are reserved for TRYCATCH

16671. Function name is an Identifier (not IdentifierName), but we get * the raw name (not recognizing keywords) here and perform the name * checks only after pass 1.

16672. DUK_TOK_DECREMENT

16673. **comment:** The target for this LDCONST may need output shuffling, but we assume * that 'pc_ldconst' will be the LDCONST that we can patch later. This * should be the case because there's no input shuffling. (If there's * no catch clause, this LDCONST will be replaced with a NOP.)

label: code-design

16674. default case control flow patchup; note that if pc_prevcase < 0 * (i.e. no case clauses), control enters default case automatically.

16675. At this point 'res' holds the potential expression value. * It can be basically any ivalue here, including a reg-bound * identifier (if code above deems it safe) or a unary/binary * operation. Operations must be resolved to a side effect free * plain value, and the side effects must happen exactly once.

16676. require a (mutable) temporary as a result (or a const if allowed)

16677. DUK_OP_NONE marks a 'plain' assignment

16678. keep in on valstack, use borrowed ref below

16679. * Parse code after the clause. Possible terminators are * 'case', 'default', and ')'. * * Note that there may be no code at all, not even an empty statement, * between case clauses. This must be handled just like an empty statement * (omitting seemingly pointless JUMPs), to avoid situations like * test-bug-case-fallthrough.js.

16680. res.x1 -> res.x2

16681. number of values in current MPUTARR set

16682. <<

16683. to exit

16684. **comment:** has inner functions which may slow access (XXX: this can be optimized by looking at the inner functions)

label: code-design

16685. Parse formals.

16686. **comment:** XXX: need to determine LHS type, and check that it is LHS compatible

label: code-design

16687. getter/setter

16688. false

16689. register bound variables are non-configurable -> always false

16690. nargs

16691. XXX: init or assert catch depth etc -- all values

16692. **comment:** Flags for intermediate value coercions. A flag for using a forced reg * is not needed, the forced_reg argument suffices and generates better * code (it is checked as it is used).

label: code-design

16693. copy bytecode instructions one at a time

16694. LHS is already side effect free

16695. * Variant 3

16696. DUK_TOK_IMPORT

16697. XXX: range assert

16698. points to LABEL; pc+1 = jump site for break; pc+2 = jump site for continue

16699. * On second pass, skip the function.

16700. **comment:** XXX: hack, remove when const lookup is not O(n)

label: code-design

16701. token closes expression, e.g. ')', ']'

16702. fileName

16703. chain jumps for case * evaluation and checking

16704. needed for line number tracking (becomes prev_token.start_line)

16705. * 'key_obj' tracks keys encountered so far by associating an * integer with flags with already encountered keys. The checks * below implement E5 Section 11.1.5, step 4 for production: ** PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment

16706. function call

16707. DUK_TOK_MUL

16708. **comment:** XXX: opcode specific assertions on when consts are allowed
label: code-design

16709. const limits

16710. don't coerce yet to a plain value (variant 3 needs special handling)

16711. The issues below can be solved with better flags

16712. [...] res]

16713. If forced reg, use it as destination. Otherwise try to * use either coerced ispec if it is a temporary. ** When using extraops, avoid reusing arg2 as dest because that * would lead to an LDREG shuffle below. We still can't guarantee * dest != arg2 because we may have a forced_reg.

16714. continue jump

16715. LeftHandSideExpression does not allow empty expression

16716. Don't allow negative zero as it will cause trouble with * LDINT+LDINTX. But positive zero is OK.

16717. First evaluate LHS fully to ensure all side effects are out.

16718. Object literal set/get functions have a name (property * name) but must not have a lexical name binding, see * test-bug-getset-func-name.js.

16719. allow source elements

16720. **comment:** XXX: This now coerces an identifier into a GETVAR to a temp, which * causes an extra LDREG in call setup. It's sufficient to coerce to a * unary ivalue?
label: code-design

16721. eat 'function'

16722. one token before

16723. keep in valstack

16724. Limit checks for bytecode byte size and line number.

16725. Parse a single statement. ** Creates a label site (with an empty label) automatically for iteration * statements. Also "peels off" any label statements for explicit labels.

16726. normal key/value

16727. Note: escaped characters differentiate directives

16728. Setting "no shuffle A" is covered by the assert, but it's needed * with DUK_USE_SHUFFLE_TORTURE.

16729. reject 'in' token (used for for-in)

16730. Note: 0xff != DUK_BC_B_MAX

16731. Function expression. Note that any statement beginning with 'function' * is handled by the statement parser as a function declaration, or a * non-standard function expression/statement (or a SyntaxError). We only * handle actual function expressions (occurring inside an expression) here. ** O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().

16732. special RegExp literal handling after IdentifierName

16733. **comment:** XXX: awkward, especially the bunch of separate output values -> output struct?
label: code-design

16734. statement is guaranteed to be terminal (control doesn't flow to next statement)

16735. fall-through jump to next code of next case (backpatched)

16736. Use a fast break/continue when possible. A fast break/continue is * just a jump to the LABEL break/continue jump slot, which then jumps * to an appropriate place (for break, going through ENDLABEL correctly). * The peephole optimizer will optimize the jump to a direct one.

16737. * The length comparisons are present to handle * strings like "use strict\u0000foo" as required.

16738. directive prologue status at entry

16739. * Function declaration, function expression, or (non-standard) * function statement. ** The E5 specification only allows function declarations at * the top level (in "source elements"). An ExpressionStatement * is explicitly not allowed to begin with a "function" keyword * (E5 Section 12.4). Hence any non-error semantics for such * non-top-level statements are non-standard. Duktape semantics * for function statements are modelled after V8, see * test-dev-func-decl-outside-top.js.

16740. Parse argument list. Arguments are written to temps starting from * "next temp". Returns number of arguments parsed. Expects left paren * to be already eaten, and eats the right paren before returning.

16741. * Variant 1 or 3

16742. temp object for tracking / detecting duplicate keys

16743. Opcode slot C is used in a non-standard way, so shuffling * is not allowed.

16744. DUK_TOK_SEMICOLON

16745. DUK_TOK_RSHIFT

16746. * Size-optimized pc->line mapping.

16747. Note: if target_pc1 == i, we'll optimize a jump to itself. * This does not need to be checked for explicitly; the case * is rare and max iter breaks us out.

16748. DUK_TOK_EQ

16749. code emission

16750. strict mode restrictions (E5 Section 12.2.1)

16751. result in csreg

16752. **comment:** Technically return value is not needed because INVLHS will * unconditionally throw a ReferenceError. Coercion is necessary * for proper semantics (consider ToNumber() called for an object). * Use DUK_EXTRAOP_UNP with a dummy register to get ToNumber().
label: code-design

16753. decl type

16754. **comment:** The entries can either be register numbers or 'null' values. * Thus, no need to DECREF them and get side effects. DECREF'ing * the keys (strings) can cause memory to be freed but no side * effects as strings don't have finalizers. This is why we can * rely on the object properties not changing from underneath us.
label: code-design

16755. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

16756. PropertyName -> IdentifierName | StringLiteral | NumericLiteral

16757. top-down expression parser

16758. Look up an identifier name in the current varmap, indicating whether the * identifier is register-bound and if not, allocating a constant for the * identifier name. Returns 1 if register-bound, 0 otherwise. Caller can * also check (out_reg_varbind >= 0) to check whether or not identifier is * register bound. The caller must NOT use out_rc_varname at all unless * return code is 0 or out_reg_varbind is < 0; this is because out_rc_varname * is unsigned and doesn't have a "unused" / none value.

16759. E5 Section steps 7-8

16760. **comment:** XXX: unoptimal use of temps, resetting
label: code-design

16761. Any 'res' will do.

16762. is_decl

16763. varmap is already in comp_ctx->curr_func.varmap_idx

16764. DUK_TOK_SWITCH

16765. eat 'try'

16766. b

16767. Note: 0xff != DUK_BC_C_MAX

16768. _Source

16769. Lenient: allow function declarations outside top level in * non-strict mode but reject them in strict mode.
16770. start variant 3/4 left-hand-side code (L1 in doc/compiler.rst example)
16771. COMMA
16772. Parse statements until a closing token (EOF or '}') is found.
16773. XXX: if assertions enabled, walk through all valid PCs * and check line mapping.
16774. The token in the switch has already been eaten here
16775. DUK_TOK_LAND
16776. currently all labels accept a break, so no explicit check for it now
16777. DUK_TOK_NEW
16778. force_no_namebind
16779. DUK_TOK_RBRACKET
16780. * Setup call: target and 'this' binding. Three cases: * * 1. Identifier base (e.g. "foo()") * 2. Property base (e.g. "foo.bar()") * 3. Register base (e.g. "foo()0"); i.e. when a return value is a function
16781. statement does not terminate directive prologue
16782. SHIFT EXPRESSION
16783. **comment:** XXX: awkward and bloated asm -- use faster internal accesses
 label: code-design
16784. approximation, close enough
16785. DUK_TOK_LOR
16786. **comment:** XXX: specific getter
 label: code-design
16787. **comment:** Then evaluate RHS fully (its value becomes the expression value too). * Technically we'd need the side effect safety check here too, but because * we always throw using INVLHS the result doesn't matter.
 label: code-design
16788. First we need to insert a jump in the middle of previously * emitted code to get the control flow right. No jumps can * cross the position where the jump is inserted. See doc/compiler.rst * for discussion on the intricacies of control flow and side effects * for variants 3 and 4.
16789. * Emit initializers in sets of maximum max_init_pairs keys. * Setter/getter is handled separately and terminates the * current set of initializer values. Corner cases such as * single value initializers do not have special handling now.
16790. intentional overlap with earlier memzero
16791. Match labels starting from latest label because there can be duplicate empty * labels in the label set.
16792. Need to set curr_token.t because lexing regexp mode depends on current * token type. Zero value causes "allow regexp" mode.
16793. chain jumps for 'fall-through' * after a case matches.
16794. assumed to also be PC of "LABEL"
16795. * Main switch for statement / source element type.
16796. * Function name (if any) * * We don't check for prohibited names here, because we don't * yet know whether the function will be strict. Function body * parsing handles this retroactively. * * For function expressions and declarations function name must * be an Identifier (excludes reserved words). For setter/getter * it is a PropertyName which allows reserved words and also * strings and numbers (e.g. "{ get 1() { ... } }").
16797. DUK_TOK_DELETE
16798. DUK_TOK_BOR
16799. DUK_TOK_CASE
16800. DUK_TOK_ADD_EQ
16801. **comment:** XXX: optimize temp reg use
 label: code-design
16802. * Compact the function template.
16803. >>>
16804. PRIMARY EXPRESSIONS
16805. left.x1 -> res.x1
16806. num_args
16807. DUK_TOK_ADD
16808. always terminates led()
16809. allocate to reg only (not const)
16810. skip jump conditionally
16811. prevent duk__expr_led() by using a binding power less than anything valid
16812. default clause matches next statement list (if any)
16813. filter out flags from exptop rbp_flags here to save space
16814. eat 'return'
16815. * Parse a function body or a function-like expression, depending * on flags.
16816. **comment:** must be able to emit code, alloc consts, etc.
 label: code-design
16817. **comment:** We want the argument expression value to go to "next temp" * without additional moves. That should almost always be the * case, but we double check after expression parsing. * * This is not the cleanest possible approach.
 label: code-design
16818. slot B is both a target and a source (used by extraops like DUK_EXTRAOP_INSTOF)
16819. FUNCTION EXPRESSIONS
16820. MULTIPLICATIVE EXPRESSION
16821. may be a string constant
16822. Labels can be used for iteration statements but also for other statements, * in particular a label can be used for a block statement. All cases of a * named label accept a 'break' so that flag is set here. Iteration statements * also allow 'continue', so that flag is updated when we figure out the * statement type.
16823. entry_top + 1
16824. **comment:** NEXTENUM needs a jump slot right after the main instruction. * When the JUMP is taken, output spilling is not needed so this * workaround is possible. The jump slot PC is exceptionally * plumbed through comp_ctx to minimize call sites.
 label: code-design
16825. UNARY EXPRESSIONS
16826. Parse a function-like expression, assuming that 'comp_ctx->curr_func' is * correctly set up. Assumes that curr_token is just after 'function' (or * 'set'/'get' etc).
16827. function: always register bound
16828. a statement following a label cannot be a source element * (a function declaration).
16829. Note: Identifier rejects reserved words
16830. [... filename (temps) func]
16831. DUK_TOK_EOF
16832. temp reg
16833. * Second pass parsing.
16834. spare
16835. tmp -> res
16836. DUK_TOK_LCURLY
16837. DUK_TOK_BREAK
16838. **comment:** XXX: for simple cases like x[y] an unnecessary LDREG is * emitted for the base value; could avoid it if we knew that * the key expression is safe (e.g. just a single literal).
 label: code-design

16839. DUK_TOK_MOD_EQ
16840. E5 Section 11.13.1 (and others) step 4 never matches for prop writes -> no check
16841. DUK_TOK_WITH
16842. These limits are based on bytecode limits. Max temps is limited * by duk_hcompiledfunction nargs/nregs fields being 16 bits.
16843. Note that duk_exptop() here can clobber any reg above current temp_next, * so any loop variables (e.g. enumerator) must be "preallocated".
16844. **comment:** XXX: need a toplain_ignore() which will only coerce a value to a temp * register if it might have a side effect. Side-effect free values do not * need to be coerced.
label: code-design
16845. * Parse a statement list. ** Handles automatic semicolon insertion and implicit return value. ** Upon entry, 'curr_tok' should contain the first token of the first * statement (parsed in the "allow regexp literal" mode). Upon exit, * 'curr Tok' contains the token following the statement list terminator * (EOF or closing brace).
16846. * Init object properties ** Properties should be added in decreasing order of access frequency. * (Not very critical for function templates.)
16847. DUK_TOK_PERIOD
16848. * ctx->prev_token token to process with duk_expr_led() * ctx->curr_token updated by caller
16849. 'res' must be a plain ivalue, and not register-bound variable.
16850. not present
16851. ADDITIVE EXPRESSION
16852. function helpers
16853. default case does not exist, or no statements present * after default case: finish case evaluation
16854. expected ival
16855. use temp_next for tracking register allocations
16856. Note: must coerce to a (writable) temp register, so that e.g. "!x" where x * is a reg-mapped variable works correctly (does not mutate the variable register).
16857. **comment:** may indirectly slow access through a direct eval
label: code-design
16858. Update all labels with matching label_id.
16859. <
16860. The directive prologue flag is cleared by default so that it is * unset for any recursive statement parsing. It is only "revived" * if a directive is detected. (We could also make directives only * allowed if 'allow_source_elem' was true.)
16861. variable binding register if register-bound (otherwise < 0)
16862. key
16863. **comment:** XXX: to be implemented?
label: code-design
16864. Property access expressions are critical for correct LHS ordering, * see comments in duk_expr()! ** A conservative approach would be to use duk_ivalue_totempconst() * for 'left'. However, allowing a reg-bound variable seems safe here * and is nice because "foo.bar" is a common expression. If the ivalue * is used in an expression a GETPROP will occur before any changes to * the base value can occur. If the ivalue is used as an assignment * LHS, the assignment code will ensure the base value is safe from * RHS mutation.
16865. * Shared handling of binary operations ** args = (is_extrap << 16) + (opcode << 8) + rbp
16866. just in case
16867. entry_top + 4
16868. key encountered as a setter
16869. continue jump not patched, an INVALID opcode remains there
16870. **comment:** XXX: increase ctx->expr_tokens here for every consumed token * (this would be a nice statistic)?
label: code-design
16871. **comment:** unused
label: code-design
16872. current (next) array index
16873. * Push result object and init its flags
16874. step 4.d
16875. duk_hobject_set_length_zero(thr, func->h_funcs);
16876. register binding lookup is based on varmap (even in first pass)
16877. empty statement with an explicit semicolon
16878. one operator (= assign)
16879. * Expression parsing. ** Upon entry to 'expr' and its variants, 'curr Tok' is assumed to be the * first token of the expression. Upon exit, 'curr Tok' will be the first * token not part of the expression (e.g. semicolon terminating an expression * statement).
16880. see above
16881. If an implicit return value is needed by caller, it must be * initialized to 'undefined' because we don't know whether any * non-empty (where "empty" is a continuation type, and different * from an empty statement) statements will be executed. ** However, since 1st pass is a throwaway one, no need to emit * it here.
16882. [... name reg/null] -> [...]
16883. no need to init reg, it will be undefined on entry
16884. must restore reliably before returning
16885. DUK_TOK_THROW
16886. Just flip the single bit.
16887. **comment:** XXX: add support for variables to not be register bound always, to * handle cases with a very large number of variables?
label: code-design
16888. entry_top + 2
16889. break jump
16890. Property access expressions ('a[b]') are critical to correct * LHS evaluation ordering, see test-dev-assign-eval-order*.js. * We must make sure that the LHS target slot (base object and * key) don't change during RHS evaluation. The only concrete * problem is a register reference to a variable-bound register * (i.e., non-temp). Require temp regs for both key and base. ** Don't allow a constant for the object (even for a number * etc), as it goes into the 'A' field of the opcode.
16891. [... key_obj key key]
16892. allow empty expression
16893. expect_eof
16894. * For/for-in main variants are: ** 1. for (ExpressionNoIn_opt; Expression_opt; Expression_opt) Statement * 2. for (var VariableDeclarationNoIn; Expression_opt; Expression_opt) Statement * 3. for (LeftHandSideExpression in Expression) Statement * 4. for (var VariableDeclarationNoIn in Expression) Statement ** Parsing these without arbitrary lookahead or backtracking is relatively * tricky but we manage to do so for now. ** See doc/compiler.rst for a detailed discussion of control flow * issues, evaluation order issues, etc.
16895. label handling
16896. DUK_TOK_SUB_EQ
16897. Expression
16898. allow negative PCs, behave as a no-op
16899. Important main primitive.
16900. ivalue/ispec helpers
16901. DUK_TOK_BXOR_EQ
16902. code_idx = entry_top + 0
16903. this optimization is important to handle negative literals * (which are not directly provided by the lexical grammar)
16904. name
16905. c
16906. E5 Section 11.13.1 (and others for other assignments), step 4.
16907. DUK_TOK_CONTINUE

16908. eat 'throw'
16909. DUK_TOK_CLASS
16910. * First pass. ** Gather variable/function declarations needed for second pass. * Code generated is dummy and discarded.
16911. Potential direct eval call detected, flag the CALL * so that a run-time "direct eval" check is made and * special behavior may be triggered. Note that this * does not prevent 'eval' from being register bound.
16912. **comment:** XXX: spare handling, slow now
 label: code-design
16913. comma check
16914. pop varname
16915. * Parse function body
16916. * Use current token to decide whether a RegExp can follow. ** We can use either 't' or 't_nores'; the latter would not * recognize keywords. Some keywords can be followed by a * RegExp (e.g. "return"), so using 't' is better. This is * not trivial, see doc/compiler.rst.
16917. the outer loop will recheck and exit
16918. allow caller to give a const number with the DUK_CONST_MARKER
16919. varname
16920. jump to end
16921. * ctx->prev_token token to process with duk_expr_nud() * ctx->curr_token updated by caller ** Note: the token in the switch below has already been eaten.
16922. basereg
16923. a
16924. * Else, must be one of: * - ExpressionStatement, possibly a directive (String) * - LabelledStatement (Identifier followed by ':') ** Expressions beginning with 'function' keyword are covered by a case * above (such expressions are not allowed in standard E5 anyway). * Also expressions starting with '{' are interpreted as block * statements. See E5 Section 12.4. ** Directive detection is tricky; see E5 Section 14.1 on directive * prologue. A directive is an expression statement with a single * string literal and an explicit or automatic semicolon. Escape * characters are significant and no parens etc are allowed: ** 'use strict'; // valid 'use strict' directive * 'use\u0020strict'; // valid directive, not a 'use strict' directive * ('use strict'); // not a valid directive ** The expression is determined to consist of a single string literal * based on duk_expr_nud() and duk_expr_led() call counts. The string literal * of a 'use strict' directive is determined to lack any escapes based * num_escapes count from the lexer. Note that other directives may be * allowed to contain escapes, so a directive with escapes does not * terminate a directive prologue. ** We rely on the fact that the expression parser will not emit any * code for a single token expression. However, it will generate an * intermediate value which we will then successfully ignore. ** A similar approach is used for labels.
16925. * Default clause.
16926. Note: 'fast' breaks will jump to pc_label_site + 1, which will * then jump here. The double jump will be eliminated by a * peephole pass, resulting in an optimal jump here. The label * site jumps will remain in bytecode and will waste code size.
16927. Used by duk_emit() calls so that we don't shuffle the loadints that * are needed to handle indirect opcodes.
16928. DUK_USE_REGEX_SUPPORT
16929. CONDITIONAL EXPRESSION
16930. break/continue without label
16931. the DUK_TOK_RCURLY is eaten by duk_parse_stmts()
16932. jump for 'catch' case
16933. **comment:** XXX: append primitive
 label: code-design
16934. [... name reg_bind]
16935. e.g. DUK_OP_PREINCR
16936. eat 'while'
16937. try part
16938. XXX: here we need to know if 'left' is left-hand-side compatible. * That information is no longer available from current expr parsing * state; it would need to be carried into the 'left' ivalue or by * some other means.
16939. * Parsing an expression starting with 'new' is tricky because * there are multiple possible productions deriving from * LeftHandSideExpression which begin with 'new'. ** We currently resort to one-token lookahead to distinguish the * cases. Hopefully this is correct. The binding power must be * such that parsing ends at an LPAREN (CallExpression) but not at * a PERIOD or LBRACKET (MemberExpression). ** See doc/compiler.rst for discussion on the parsing approach, * and testcases/test-dev-new.js for a bunch of documented tests.
16940. exptop is the top level variant which resets nud/led counts
16941. next loop requires a comma
16942. If there's no catch block, rc_varname will be 0 and duk_patch_trycatch() * will replace the LDCONST with a NOP. For any actual constant (including * constant 0) the DUK_CONST_MARKER flag will be set in rc_varname.
16943. * Init remaining result fields * * 'nreg' controls how large a register frame is allocated. ** 'nargs' controls how many formal arguments are written to registers: * r0, ... r(nargs-1). The remaining registers are initialized to * undefined.
16944. get const for value at valstack top
16945. eat 'break' or 'continue'
16946. DUK_TOK_TYPEOF
16947. * Parse inner function
16948. **comment:** unused
 label: code-design
16949. No BC shuffling now.
16950. number
16951. syntactically left-associative but parsed as right-associative
16952. reserve a jumpslot after instr before target spilling, used for NEXTENUM
16953. * Wrap up
16954. at least one opcode emitted
16955. Encode into a single opcode (18 bits can encode 1-2 bytes + length indicator)
16956. relookup after possible realloc
16957. The 'left' value must not be a register bound variable * because it may be mutated during the rest of the expression * and E5.1 Section 11.2.1 specifies the order of evaluation * so that the base value is evaluated first. * See: test-bug-nested-prop-mutate.js.
16958. regexp literal must not follow this token
16959. -> [... func]
16960. automatic semi will be inserted
16961. Special shuffling for INITGET/INITSET, where slot C * identifies a register pair and cannot be shuffled * normally. Use an indirect variant instead.
16962. a label site has been emitted by duk_parse_stmt() automatically * (it will also emit the ENDLABEL).
16963. slot B is a target (default: source)
16964. without finally, the second jump slot is used to jump to end of stmt
16965. forget temp
16966. [... filename &comp_stk]
16967. break/continue with label (label cannot be a reserved word, production is 'Identifier'
16968. Have a catch variable.
16969. * Intermediate value helpers
16970. * Shared assignment expression handling ** args = (opcode << 8) + rbp * * If 'opcode' is DUK_OP_NONE, plain assignment without arithmetic. * Syntactically valid left-hand-side forms which are not accepted as * left-hand-side values (e.g. as in "f0 = 1") must NOT cause a * SyntaxError, but rather a run-time ReferenceError. ** When evaluating X <op>= Y, the LHS (X) is conceptually evaluated * to a temporary first. The RHS is then evaluated. Finally, the * <op> is applied to the initial value of RHS (not the value after * RHS evaluation), and written to X. Doing so concretely generates * inefficient code so we'd like to avoid the temporary when possible. * See: https://github.com/svaarala/duktape/pull/992. ** The expression value (final LHS value, written to RHS) is * conceptually

copied into a fresh temporary so that it won't * change even if the LHS/RHS values change in outer expressions. * For example, it'd be generally incorrect for the expression value * to be the RHS register binding, unless there's a guarantee that it * won't change during further expression evaluation. Using the * temporary concretely produces inefficient bytecode, so we try to * avoid the extra temporary for some known-to-be-safe cases. * Currently the only safe case we detect is a "top level assignment", * for example "x = y + z;", where the assignment expression value is * ignored. * See: test-dev-assign Expr.js and test-bug-assign-mutate-gh381.js.

16971. TRYCATCH, cannot emit now (not enough info)

16972. reset function state (prepare for pass 2)

16973. **comment:** Lookup active label information. Break/continue distinction is necessary to handle switch * statement related labels correctly: a switch will only catch a 'break', not a 'continue'. * * An explicit label cannot appear multiple times in the active set, but empty labels (unlabelled * iteration and switch statements) can. A break will match the closest unlabelled or labelled * statement. A continue will match the closest unlabelled or labelled iteration statement. It is * a syntax error if a continue matches a labelled switch statement; because an explicit label cannot * be duplicated, the continue cannot match any valid label outside the switch. * * A side effect of these rules is that a LABEL statement related to a switch should never actually * catch a continue abrupt completion at run-time. Hence an INVALID opcode can be placed in the * continue slot of the switch's LABEL statement.

label: code-design

16974. Value stack slot limits: these are quite approximate right now, and * because they overlap in control flow, some could be eliminated.

16975. **comment:** NOTE: "get" and "set" are not officially ReservedWords and the lexer * currently treats them always like ordinary identifiers (DUK_TOK_GET * and DUK_TOK_SET are unused). They need to be detected based on the * identifier string content.

label: code-design

16976. **comment:** * Note: currently register bindings must be fixed for the entire * function. So, even though the catch variable is in a register * we know, we must use an explicit environment record and slow path * accesses to read/write the catch binding to make closures created * within the catch clause work correctly. This restriction should * be fixable (at least in common cases) later. * * See: test-bug-catch-binding-2.js. * * XXX: improve to get fast path access to most catch clauses.

label: code-design

16977. DUK_TOK_EXPORT

16978. entry_top + 0

16979. * Variant 2 or 4

16980. [... escaped_source bytecode]

16981. return 'res' (of right part) as our result

16982. jump from true part to end

16983. **comment:** XXX: spilling

label: code-design

16984. eat identifier

16985. x1 must be a string

16986. break matches always

16987. dangerous: must only lower (temp_max not updated)

16988. * Inside one or more 'with' statements fall back to slow path always. * (See e.g. test-stmt-with.js.)

16989. format is bit packed

16990. DUK_TOK_LBRACKET already eaten, current token is right after that

16991. parse args starting from "next temp", reg_target + 1

16992. is_setget

16993. enable manually for dumping

16994. shadowed, ignore

16995. DUK_TOK_COLON

16996. DUK_TOK_IMPLEMENT

16997. jump to end or else part

16998. for side effects, result ignored

16999. for duk_error_augment.c

17000. AssignmentExpression

17001. DUK_TOK_MOD

17002. **comment:** XXX: this is pointless here because pass 1 is throw-away

label: code-design

17003. * On first pass, perform actual parsing. Remember valstack top on entry * to restore it later, and switch to using a new function in comp_ctx.

17004. non-Reference deletion is always 'true', even in strict mode

17005. no statement value (unlike function expression)

17006. jump to next loop, using reg_v34_iter as iterated value

17007. DUK_USE_TAILCALL

17008. For X <op>= Y we need to evaluate the pre-op * value of X before evaluating the RHS: the RHS * can change X, but when we do <op> we must use * the pre-op value.

17009. * Convert duk_compiler_func to a function template

17010. final result is already in 'res'

17011. -> loop body

17012. catch jump

17013. num_pairs and temp_start reset at top of outer loop

17014. **comment:** XXX: return indication of "terminalness" (e.g. a 'throw' is terminal)

label: code-design

17015. * Wrapping duk_safe_call() will mangle the stack, just return stack top

17016. pc_label

17017. slot A is a source (default: target)

17018. assume 'var' has been eaten

17019. old value is number: no refcount

17020. empty expressions can be detected conveniently with nud/led counts

17021. DUK_TOK_ALSHIFT_EQ

17022. label limits

17023. A 'return' statement is only allowed inside an actual function body, * not as part of eval or global code.

17024. DUK_TOK_GE

17025. DUK_TOK_LET

17026. Only a reg fits into 'A' so coerce 'res' into a register * for PUTVAR. * * XXX: here the current A/B/C split is suboptimal: we could * just use 9 bits for reg_res (and support constants) and 17 * instead of 18 bits for the varname const index.

17027. else an array initializer element

17028. tracks maximum initialized index + 1

17029. * Convert duk_compiler_func to a function template and add it * to the parent function table.

17030. **comment:** XXX: very similar to DUK_IVAL_ARITH - merge

label: code-design

17031. **comment:** * Formal argument list * * We don't check for prohibited names or for duplicate argument * names here, because we don't yet know whether the function will * be strict. Function body parsing handles this retroactively.

label: code-design

17032. object literal key tracking flags

17033. reset 'allow_in' for parenthesized expression

17034. Coerce an duk_alue to a register or constant; result register may * be a temp or a bound register. * * The duk_alue argument ('x') is converted into a regconst as a * side effect.
17035. MEMBER/NEW/CALL EXPRESSIONS
17036. **comment:** XXX: add flag to indicate whether caller cares about return value; this * affects e.g. handling of assignment expressions. This change needs API * changes elsewhere too.
label: code-design
17037. Input shuffling happens before the actual operation, while output * shuffling happens afterwards. Output shuffling decisions are still * made at the same time to reduce branch clutter; output shuffle decisions * are recorded into X_out variables.
17038. parse args starting from "next temp"
17039. **comment:** XXX: support unary arithmetic ivalues (useful?)
label: code-design
17040. arguments object would be accessible; note that shadowing * bindings are arguments or function declarations, neither * of which are deletable, so this is safe.
17041. lexing
17042. Main operation
17043. duk_regconst_t is unsigned, so use 0 as dummy value (ignored by caller)
17044. * For global or eval code this is straightforward. For functions * created with the Function constructor we only get the source for * the body and must manufacture the "function ..." part. * * For instance, for constructed functions (v8): * * > a = new Function("foo", "bar", "print(foo)"); * [Function] * > a.toString() * 'function anonymous(foo,bar) {\nprint(foo)\n}' * * Similarly for e.g. getters (v8): * * > x = { get a(foo,bar) { print(foo); } } * { a: [Getter] } * > Object.getOwnPropertyDescriptor(x, 'a').get.toString() * 'function a(foo,bar) { print(foo); }'
17045. initial index
17046. 'new' MemberExpression Arguments
17047. stmt has explicit/implicit semicolon terminator
17048. op_flags
17049. **comment:** * See the following documentation for discussion: * * doc/execution.rst: control flow details * * Try, catch, and finally "parts" are Blocks, not Statements, so * they must always be delimited by curly braces. This is unlike e.g. * the if statement, which accepts any Statement. This eliminates any * questions of matching parts of nested try statements. The Block * parsing is implemented inline here (instead of calling out). * * Finally part has a 'let scoped' variable, which requires a few kinks * here.
label: code-design
17050. XXX: integrate support for this into led() instead? * Similar issue as post-increment/post-decrement.
17051. DUK_TOK_FUNCTION
17052. array writes autoincrement length
17053. * Prototypes
17054. main expression parser function
17055. * 'arguments' binding is special; if a shadowing argument or * function declaration exists, an arguments object will * definitely not be needed, regardless of whether the identifier * 'arguments' is referenced inside the function body.
17056. statements go here (if any) on next loop
17057. **comment:** XXX: make this lenience dependent on flags or strictness?
label: code-design
17058. silence scan-build warning
17059. varname is popped by above code
17060. always allow 'in', coerce to 'tr' just in case
17061. A top-level assignment is e.g. "x = y". For these it's safe * to use the RHS as-is as the expression value, even if the RHS * is a reg-bound identifier. The RHS ('res') is right associative * so it has consumed all other assignment level operations; the * only relevant lower binding power construct is comma operator * which will ignore the expression value provided here. Usually * the top level assignment expression value is ignored, but it * is relevant for e.g. eval code.
17062. Tail call check: if last opcode emitted was CALL(I), and * the context allows it, change the CALL(I) to a tail call. * This doesn't guarantee that a tail call will be allowed at * runtime, so the RETURN must still be emitted. (Duktape * 0.10.0 avoided this and simulated a RETURN if a tail call * couldn't be used at runtime; but this didn't work * correctly with a thread yield/resume, see * test-bug-tailcall-thread-yield-resume.js for discussion.) * * In addition to the last opcode being CALL, we also need to * be sure that 'rc_val' is the result register of the CALL(I). * For instance, for the expression 'return 0, (function () * { return 1; })', 2' the last opcode emitted is CALL (no * bytecode is emitted for '2') but 'rc_val' indicates * constant '2'. Similarly if '2' is replaced by a register * bound variable, no opcodes are emitted but tail call would * be incorrect. * * This is tricky and easy to get wrong. It would be best to * track enough expression metadata to check that 'rc_val' came * from that last CALL instruction. We don't have that metadata * now, so we check that 'rc_val' is a temporary register result * (not a constant or a register bound variable). There should * be no way currently for 'rc_val' to be a temporary for an * expression following the CALL instruction without emitting * some opcodes following the CALL. This proxy check is used * below. * * See: test-bug-comma-expr-gh131.js. * * The non-standard 'caller' property disables tail calls * because they pose some special cases which haven't been * fixed yet.
17063. DUK_TOK_YIELD
17064. DUK_TOK_LNOT
17065. slot C is a target (default: source)
17066. Match labels starting from latest; once label_id no longer matches, we can * safely exit without checking the rest of the labels (only the topmost labels * are ever updated).
17067. reset bytecode buffer but keep current size; pass 2 will * require same amount or more.
17068. The 'else' ambiguity is resolved by 'else' binding to the innermost * construct, so greedy matching is correct here.
17069. **comment:** not allowed in strict mode, regardless of whether resolves; * in non-strict mode DELVAR handles both non-resolving and * resolving cases (the specification description is a bit confusing).
label: code-design
17070. end switch
17071. DUK_TOK_LE
17072. Use 'res' as the expression value (it's side effect * free and may be a plain value, a register, or a * constant) and write it to the LHS binding too.
17073. **comment:** 'res' setup can be moved to function body level; in fact, two 'res' * intermediate values suffice for parsing of each function. Nesting is needed * for nested functions (which may occur inside expressions).
label: code-design
17074. Parse a single variable declaration (e.g. "i" or "i=10"). A leading 'var' * has already been eaten. These is no return value in 'res', it is used only * as a temporary. * * When called from 'for-in' statement parser, the initializer expression must * not allow the 'in' token. The caller supply additional expression parsing * flags (like DUK_EXPR_FLAG_REJECT_IN) in 'expr_flags'. * * Finally, out_rc_varname and out_reg_varbind are updated to reflect where * the identifier is bound: * * If register bound: out_reg_varbind >= 0, out_rc_varname == 0 (ignore) * If not register bound: out_reg_varbind < 0, out_rc_varname >= 0 * * These allow the caller to use the variable for further assignment, e.g. * as is done in 'for-in' parsing.
17075. XXX: similar coercion issue as in DUK_TOK_PERIOD
17076. borrowed reference
17077. * Statement terminator check, including automatic semicolon * handling. After this step, 'curr_tok' should be the first * token after a possible statement terminator.
17078. bp to use when parsing a top level Expression
17079. Matches both +0 and -0 on purpose.
17080. bp is assumed to be even
17081. The code for writing reg_temps + 0 to the left hand side has already * been emitted.
17082. * Statement value handling. * * Global code and eval code has an implicit return value * which comes from the last statement with a value * (technically a non- "empty" continuation, which is * different from an empty statement). * * Since we don't know whether a later statement will * override the value of the current statement, we need * to coerce the statement value to a register allocated * for implicit return values. In other cases we need * to coerce the statement value to a plain value to get * any side effects out (consider e.g. "foo.bar;").
17083. transfer flags
17084. XXX: valstack handling is awkward. Add a valstack helper which * avoids dup():ing; valstack_copy(src, dst)?

17085. eat 'var'
17086. DUK_TOK_INTERFACE
17087. DUK_TOK_DIV
17088. * Any catch bindings ("catch (e)") also affect identifier binding. * * Currently, the varmap is modified for the duration of the catch * clause to ensure any identifier accesses with the catch variable * name will use slow path.
17089. iteration statements allow continue
17090. reg/const for switch value
17091. unary plus
17092. may be undefined
17093. reg/const for case value
17094. * Function formal arguments, always bound to registers * (there's no support for shuffling them now).
17095. **comment:** NEXTENUM needs a jump slot right after the main opcode. * We need the code emitter to reserve the slot: if there's * target shuffling, the target shuffle opcodes must happen * after the jump slot (for NEXTENUM the shuffle opcodes are * not needed if the enum is finished).
label: code-design
17096. DUK_TOK_PRIVATE
17097. Encode into a double constant (53 bits can encode $6 \times 8 = 48$ bits + 3-bit length)
17098. then to a register
17099. expect EOF instead of }
17100. DUK_TOK_ARSHIFT_EQ
17101. **comment:** * For/for-in statement is complicated to parse because * determining the statement type (three-part for vs. a * for-in) requires potential backtracking. * * See the helper for the messy stuff.
label: code-design
17102. no code needs to be emitted, the regs already have values
17103. DUK_TOK_RCURLY
17104. fills window
17105. DUK_TOK_BAND_EQ
17106. jump is inserted here (variant 3)
17107. * Program code (global and eval code) has an implicit return value * from the last statement value (e.g. eval("1; 2+3;") returns 3). * This is not the case with functions. If implicit statement return * value is requested, all statements are coerced to a register * allocated here, and used in the implicit return statement below.
17108. **comment:** * Assignments are right associative, allows e.g. * a = 5; * a += b = 9; // same as a += (b = 9) * -> expression value 14, a = 14, b = 9 * * Right associativity is reflected in the BP for recursion, * "-1" ensures assignment operations are allowed. * * XXX: just use DUK_BP_COMMA (i.e. no need for 2-step bp levels)?
label: code-design
17109. don't allow continue
17110. valstack will be unbalanced, which is OK
17111. == DUK_MAX_TEMPS is OK
17112. DUK_TOK_EXTENDS
17113. [...] varmap]
17114. inline arithmetic check for constant values
17115. identifier handling
17116. * Helpers for duk_compiler_func.
17117. DUK_TOK_RSHIFT_EQ
17118. step 4.b
17119. explicit semi follows
17120. shadowed; update value
17121. No support for lvalues returned from new or function call expressions. * However, these must NOT cause compile-time SyntaxErrors, but run-time * ReferenceErrors. Both left and right sides of the assignment must be * evaluated before throwing a ReferenceError. For instance: * * f() = g(); * * must result in f() being evaluated, then g() being evaluated, and * finally, a ReferenceError being thrown. See E5 Section 11.13.1.
17122. advance, whatever the current token is; parse next token in regexp context
17123. DUK_TOK_PROTECTED
17124. curr_token = get/set name
17125. target
17126. XXX: default priority for infix operators is duk__expr_lbp(tok) -> get it here?
17127. * Parse a function-like expression: * * - function expression * - function declaration * - function statement (non-standard) * - setter/getter * * Adds the function to comp_ctx->curr_func function table and returns the * function number. * * On entry, curr_token points to: * * - the token after 'function' for function expression/declaration/statement * - the token after 'set' or 'get' for setter/getter
17128. expr_flags
17129. Note: expect that caller has already eaten the left paren
17130. advance, expecting current token to be a specific token; parse next token in regexp context
17131. already in correct reg
17132. RELATIONAL EXPRESSION
17133. * Source filename (or equivalent), for identifying thrown errors.
17134. e.g. DUK_OP_POSTINCP
17135. preallocated temporaries (2) for variants 3 and 4
17136. emit instruction; could return PC but that's not needed in the majority * of cases.
17137. DUK_TOK_ALSHIFT
17138. **comment:** * Parse a function-body-like expression (FunctionBody or Program * in E5 grammar) using a two-pass parse. The productions appear * in the following contexts: * * - function expression * - function statement * - function declaration * - getter in object literal * - setter in object literal * - global code * - eval code * - Function constructor body * * This function only parses the statement list of the body; the argument * list and possible function name must be initialized by the caller. * For instance, for Function constructor, the argument names are originally * on the value stack. The parsing of statements ends either at an EOF or * a closing brace; this is controlled by an input flag. * * Note that there are many differences affecting parsing and even code * generation: * * - Global and eval code have an implicit return value generated * by the last statement; function code does not * * - Global code, eval code, and Function constructor body end in * an EOF, other bodies in a closing brace ('}') * * Upon entry, 'curr_tok' is ignored and the function will pull in the * first token on its own. Upon exit, 'curr_tok' is the terminating * token (EOF or closing brace).
label: code-design
17139. trailing comma followed by rcurly
17140. eat 'with'
17141. comp_ctx->lex has been pre-initialized by caller: it has been * zeroed and input/input_length has been set.
17142. When torture not enabled, can just use the same helper because * 'reg' won't get spilled.
17143. step 4.c
17144. [...] key_obj key val]
17145. combine left->x1 and res->x1 (right->x1, really) -> (left->x1 OP res->x1)
17146. DUK_TOK_INCREMENT
17147. const flag for B
17148. keep func->h_argnames; it is fixed for all passes
17149. misc
17150. source is a function expression (used for Function constructor)
17151. **comment:** XXX: typing (duk_hcompiledfunction has duk_uint32_t)

label: code-design

17152. parsing in "directive prologue", recognize directives

17153. value stack indices for tracking objects

17154. DUK_ISPEC_XXX

17155. **comment:** code emission temporary**label:** code-design

17156. maximum bytecode length in instructions

17157. bufwriter for code

17158. C array of duk_labelinfo

17159. Compiling state of one function, eventually converted to duk_hcompiledfunction

17160. with stack depth (affects identifier lookups)

17161. variable map for pass 2 (identifier -> register number or null (unmapped))

17162. curr_token slot1 (matches 'lex' slot1_idx)

17163. * Compiler intermediate values * * Intermediate values describe either plain values (e.g. strings or * numbers) or binary operations which have not yet been coerced into * either a left-hand-side or right-hand-side role (e.g. object property).

17164. type to represent a reg/const reference during compilation

17165. function is strict

17166. * Ecmascript compiler.

17167. array of declarations: [name1, val1, name2, val2, ...] * valN = (typeN) | (fnum << 8), where fnum is inner func number (0 for vars) * record function and variable declarations in pass 1

17168. bytecode opcode (or extraop) for binary ops

17169. current and previous token for parsing

17170. **comment:** function name (borrowed reference), ends up in _name**label:** code-design

17171. lexing (tokenization) state (contains two valstack slot indices)

17172. property access

17173. array of active label names

17174. is eval code

17175. 1e8: protects against deeply nested inner functions

17176. source is eval code (not global)

17177. **comment:** function must not be tail called**label:** code-design17178. **comment:** code_idx: not needed**label:** requirement

17179. no value

17180. inner function numbering

17181. always set; points to a reserved valstack slot

17182. expression is left-hand-side compatible

17183. These pointers are at the start of the struct so that they pack * nicely. Mixing pointers and integer values is bad on some * platforms (e.g. if int is 32 bits and pointers are 64 bits).

17184. highest value of temp_reg (temp_max - 1 is highest used reg)

17185. **comment:** XXX: can be optimized for smaller footprint esp. on 32-bit environments**label:** code-design

17186. * Compiler state

17187. argument/function declaration shadows 'arguments'

17188. is a function declaration (as opposed to function expression)

17189. variable access

17190. function needs shuffle registers

17191. borrowed label name

17192. is global code

17193. strict outer context

17194. filename being compiled (ends up in functions' '_filename' property)

17195. * PLAIN: x1 * ARITH: x1 <op> x2 * PROP: x1.x2 * VAR: x1 (name)

17196. function may call direct eval

17197. catch depth at point of definition

17198. binary arithmetic using extraops; DUK_EXTRAOP_INSTOF etc

17199. **comment:** XXX: optimize to 16 bytes**label:** code-design

17200. binary arithmetic; DUK_OP_ADD, DUK_OP_EQ, other binary ops

17201. function refers to 'arguments' identifier

17202. **comment:** first register that is a temporary (below: variables)**label:** code-design

17203. catch stack depth

17204. maximum loopcount for peephole optimization

17205. value resides in 'valstack_idx'

17206. value resides in a register or constant

17207. Fast jumps (which avoid longjmp) jump directly to the jump sites * which are always known even while the iteration/switch statement * is still being parsed. A final peephole pass "straightens out" * the jumps.

17208. label id allocation (running counter)

17209. 1 GB

17210. current function being compiled (embedded instead of pointer for more compact access)

17211. stats for current expression being parsed

17212. * Prototypes

17213. DUK_IVAL_XXX

17214. **comment:** function makes one or more slow path accesses**label:** code-design

17215. is an actual function (not global/eval code)

17216. array of function templates: [func1, offset1, line1, func2, offset2, line2] * offset/line points to closing brace to allow skipping on pass 2

17217. is a setter/getter

17218. ecmascript compiler limits

17219. curr_token slot2 (matches 'lex' slot2_idx)

17220. borrowed reference

17221. type to represent a straight register reference, with <0 indicating none

17222. **comment:** bit mask which indicates that a regconst is a constant instead of a register**label:** code-design

17223. pc of label statement: * pc+1: break jump site * pc+2: continue jump site

17224. next temporary register to allocate

17225. parsing in "scanning" phase (first pass)

17226. shuffle registers if large number of regs/consts
17227. array
17228. array of formal argument names (-> _Formals)
17229. * Bytecode instruction representation during compilation ** Contains the actual instruction and (optionally) debug info.
17230. prev_token slot2
17231. DUK_JS_COMPILER_H_INCLUDED
17232. status booleans
17233. register for writing value of 'non-empty' statements (global or eval code), -1 is marker
17234. current paren level allows 'in' token
17235. numeric label_id (-1 reserved as marker)
17236. prev_token slot1
17237. reject RegExp literal on next advance() call; needed for handling IdentifierName productions
17238. statement id allocation (running counter)
17239. recursion limit
17240. parenthesis count, 0 = top level
17241. number of formal arguments
17242. register, constant, or value
17243. temp reg handling
17244. h_code: held in bw_code
17245. * Fast paths
17246. **comment:** 'd3' is never NaN, so no need to normalize
label: code-design
17247. * Executor interrupt handling ** The handler is called whenever the interrupt countdown reaches zero * (or below). The handler must perform whatever checks are activated, * e.g. check for cumulative step count to impose an execution step * limit or check for breakpoints or other debugger interaction. ** When the actions are done, the handler must reinits the interrupt * init and counter values. The 'init' value must indicate how many * bytecode instructions are executed before the next interrupt. The * counter must interface with the bytecode executor loop. Concretely, * the new init value is normally one higher than the new counter value. * For instance, to execute exactly one bytecode instruction the init * value is set to 1 and the counter to 0. If an error is thrown by the * interrupt handler, the counters are set to the same value (e.g. both * to 0 to cause an interrupt when the next bytecode instruction is about * to be executed after error handling). ** Maintaining the init/counter value properly is important for accurate * behavior. For instance, executor step limit needs a cumulative step * count which is simply computed as a sum of 'init' values. This must * work accurately even when single stepping.
17248. return value from Duktape.Thread.resume()
17249. Set catcher regs: idx_base+0 = value, idx_base+1 = lj_type.
17250. Result is always fastint compatible.
17251. Handle a BREAK/CONTINUE opcode. Avoid using longjmp() for BREAK/CONTINUE * handling because it has a measurable performance impact in ordinary * environments and an extreme impact in Emscripten (GH-342).
17252. -> [... val this]
17253. Preinc/predec for object properties.
17254. Assume that thr->valstack_bottom has been set-up before getting here.
17255. Sync and NULL early.
17256. * E5 Section 11.4.9
17257. **comment:** Hot variables for interpretation. Critical for performance, * but must add sparingly to minimize register shuffling.
label: code-design
17258. an Ecmascript function
17259. **comment:** XXX: improve check; check against nregs, not against top
label: code-design
17260. -> [... regexp_instance]
17261. Keep throwing an error whenever we get here. The unusual values * are set this way because no instruction is ever executed, we just * throw an error until all try/catch/finally and other catchpoints * have been exhausted. Duktape/C code gets control at each protected * call but whenever it enters back into Duktape the RangeError gets * raised. User exec timeout check must consistently indicate a timeout * until we've fully bubbled out of Duktape.
17262. x >= y --> not (x < y)
17263. A -> flags * B -> base register for call (base -> func, base+1 -> this, base+2 -> arg1 ... base+2+N-1 -> argN) * (for DUK_OP_CALLI, 'b' is indirect) * C -> nargs
17264. Unary plus is used to force a fastint check, so must include * downgrade check.
17265. Duktape.Thread.yield() should prevent
17266. **comment:** should never happen, but be robust
label: code-design
17267. [... enum] -> [...]
17268. A -> register of target object * B -> first register of key/value pair list * C -> number of key/value pairs
17269. **comment:** Note: 'act' is dangerous here because it may get invalidated at many * points, so we re-lookup it multiple times.
label: code-design
17270. XXX: E5.1 Section 11.1.4 coerces the final length through * ToUint32() which is odd but happens now as a side effect of * 'arr_idx' type.
17271. side effects -> don't use tv_x, tv_y after
17272. flags
17273. Relookup and initialize dispatch loop variables. Debugger check.
17274. never here
17275. add_auto_proto
17276. don't re-enter e.g. during Eval
17277. -> [... obj key value]
17278. duk_new() will call the constructor using duk_handle_call(). * A constructor call prevents a yield from inside the constructor, * even if the constructor is an Ecmascript function.
17279. may be < thr->catchstack initially
17280. Longjmp handling has restored jmpbuf_ptr.
17281. valstack top, should match before interpreting each op (no leftovers)
17282. Entry level info.
17283. Input values are signed 48-bit so we can detect overflow * reliably from high bits or just a comparison.
17284. XXX: assert 'c' is an enumerator
17285. -> [val]
17286. **comment:** XXX: could unwind catchstack here, so that call handling * didn't need to do that?
label: code-design
17287. **comment:** XXX: refactor out?
label: code-design
17288. [... env]
17289. B -> register for writing enumerator object * C -> value to be enumerated (register)
17290. cleared before entering catch part
17291. -> [... obj]
17292. no need to unwind catchstack
17293. **comment:** * Arithmetic operations other than '+' have number-only semantics * and are implemented here. The separate switch-case here means a * "double dispatch" of the arithmetic opcode, but saves code space. * * E5 Sections 11.5, 11.5.1, 11.5.2, 11.5.3, 11.6, 11.6.1, 11.6.2, 11.6.3.
label: code-design

17294. When debugger is enabled, we need to recheck the activation * status after returning. This is now handled by call handling * and heap->dbg_force_restart.
17295. thr->heap->lj.value1 is already the value to throw
17296. if boolean matches A, skip next inst
17297. Preinc/preddec for var-by-name, slow path.
17298. A -> target register * B -> bytecode (also contains flags) * C -> escaped source
17299. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
17300. bytecode has a stable pointer
17301. cleared before entering finally
17302. **comment:** Executor interrupt counter check, used to implement breakpoints, * debugging interface, execution timeouts, etc. The counter is heap * specific but is maintained in the current thread to make the check * as fast as possible. The counter is copied back to the heap struct * whenever a thread switch occurs by the DUK_HEAP_SWITCH_THREAD() macro.
label: code-design
17303. Signed shift, named "arithmetic" (asl) because the result * is signed, e.g. 4294967295 << 1 -> -2. Note that result * must be masked.
17304. Overflow of the execution counter is fine and doesn't break * anything here.
17305. Restart bytecode execution, possibly with a changed thread.
17306. Don't allow a zero divisor. Fast path check by * "verifying" with multiplication. Also avoid zero * dividend to avoid negative zero issues (0 / -1 = -0 * for instance).
17307. no need for refcount update
17308. reg count
17309. **comment:** ToBoolean() does not require any operations with side effects so * we can do it efficiently. For footprint it would be better to use * duk_js_toboolen() and then push+replace to the result slot.
label: code-design
17310. * Slow path: potentially requires function calls for coercion
17311. -> [Object defineProperty undefined obj key desc]
17312. DUK_USE_FASTINT
17313. ToNumber() for a fastint is a no-op.
17314. B -> object register * C -> C+0 contains key, C+1 closure (value)
17315. x < y
17316. call_flags
17317. **comment:** XXX: unnecessary copying of values? Just set 'top' to * b + c, and let the return handling fix up the stack frame?
label: code-design
17318. Force interrupt right away if we're paused or in "checked mode". * Step out is handled by callstack unwind.
17319. * Local declarations.
17320. * INITSET/INITGET are only used to initialize object literal keys. * The compiler ensures that there cannot be a previous data property * of the same name. It also ensures that setter and getter can only * be initialized once (or not at all).
17321. If we processed any debug messages breakpoints may have * changed; restart execution to re-check active breakpoints.
17322. Handle a RETURN opcode. Avoid using longjmp() for return handling because * it has a measurable performance impact in ordinary environments and an extreme * impact in Emscripten (GH-342). Return value is on value stack top.
17323. Ecma-to-ecma call possible, may or may not be a tail call. * Avoid C recursion by being clever.
17324. Note: MUST NOT wipe_and_return here, as heap->lj must remain intact
17325. x > y --> y < x
17326. **comment:** XXX: this is now a very unoptimal implementation -- this can be * made very simple by direct manipulation of the object internals, * given the guarantees above.
label: code-design
17327. curr_pc synced by duk_handle_ecma_call_setup()
17328. resumee: [... initial_func] (currently actually: [initial_func])
17329. **comment:** XXX: use duk_is_valid_index() instead?
label: code-design
17330. -> [... obj value]
17331. always normalized
17332. DUK_USE_INTERRUPT_COUNTER
17333. **comment:** writable, not configurable
label: code-design
17334. written by a previous RESUME handling
17335. **comment:** LDINTX is not necessarily in FASTINT range, so * no fast path for now. * * XXX: perhaps restrict LDINTX to fastint range, wider * range very rarely needed.
label: code-design
17336. duk_js_call.c is required to restore the stack reserve * so we only need to reset the top.
17337. **comment:** this is not strictly necessary, but helps debugging
label: code-design
17338. this should never be possible, because the switch-case is * comprehensive
17339. Pre/post inc/dec for register variables, important for loops.
17340. * Avoid nested calls. Concretely this happens during debugging, e.g. * when we eval() an expression. * * Also don't interrupt if we're currently doing debug processing * (which can be initiated outside the bytecode executor) as this * may cause the debugger to be called recursively. Check required * for correct operation of throw intercept and other "exotic" halting * scenarios.
17341. w/o refcounts
17342. fast path
17343. **comment:** XXX: this could be a DUK__CONSTP instead
label: code-design
17344. stable; precalculated for faster lookups
17345. **comment:** 'val' is never NaN, so no need to normalize
label: code-design
17346. may be reg or const
17347. +2 = catcher value, catcher lj_type
17348. **comment:** XXX: because we're dealing with 'own' properties of a fresh array, * the array initializer should just ensure that the array has a large * enough array part and write the values directly into array part, * and finally set 'length' manually in the end (as already happens now).
label: code-design
17349. * Longjmp and other control flow transfer for the bytecode executor. * * The longjmp handler can handle all longjmp types: error, yield, and * resume (pseudotypes are never actually thrown). * * Error policy for longjmp: should not ordinarily throw errors; if errors * occur (e.g. due to out-of-memory) they bubble outwards rather than being * handled recursively.
17350. **comment:** Reg/const access macros: these are very footprint and performance sensitive * so modify with care.
label: code-design
17351. [... obj] -> [...]
17352. If out of catchstack, cat = thr->catchstack - 1; * new_cat_top will be 0 in that case.
17353. duk_xdef_prop() will define an own property without any array * special behaviors. We'll need to set the array length explicitly * in the end. For arrays with elisions, the compiler will emit an * explicit SETALEN which will update the length.
17354. **comment:** Switch to caller's setjmp() catcher so that if an error occurs * during error handling, it is always propagated outwards instead * of causing an infinite loop in our own handler.

label: code-design

17355. **comment:** XXX: remove heap->dbg_exec_counter, use heap->inst_count_interrupt instead?

label: code-design

17356. functions always have a NEWENV flag, i.e. they get a * new variable declaration environment, so only lex_env * matters here.

17357. should not happen

17358. A -> flags * BC -> regCatch; base register for two registers used both during * trycatch setup and when catch is triggered * * If DUK_BC_TRYCATCH_FLAG_CATCH_BINDING set: * regCatch + 0: catch binding variable name (string). * Automatic declarative environment is established for * the duration of the 'catch' clause. ** If DUK_BC_TRYCATCH_FLAG_WITH_BINDING set: * regCatch + 0: with 'target value', which is coerced to * an object and then used as a bindind object for an * environment record. The binding is initialized here, for * the 'try' clause. ** Note that a TRYCATCH generated for a 'with' statement has no * catch or finally parts.

17359. [... constructor arg1 ... argN] -> [retval]

17360. Check for breakpoints only on line transition. * Breakpoint is triggered when we enter the target * line from a different line, and the previous line * was within the same function. * * This condition is tricky: the condition used to be * that transition to -or across- the breakpoint line * triggered the breakpoint. This seems intuitively * better because it handles breakpoints on lines with * no emitted opcodes; but this leads to the issue * described in: <https://github.com/svaarala/duktape/issues/263>.

17361. B -> target register * C -> constant index of identifier name

17362. * Execution timeout check

17363. side effects

17364. E5 Section 11.7.2, steps 7 and 8

17365. updates thread state, minimizes its allocations

17366. **comment:** It might seem that replacing 'thr->heap' with just 'heap' below * might be a good idea, but it increases code size slightly * (probably due to unnecessary spilling) at least on x64.

label: code-design

17367. use_prev_pc

17368. Must restart in all cases because we NULled thr->ptr_curr_pc.

17369. checked by Duktape.Thread.resume()

17370. unsigned

17371. duk.bi_duk_object_yield() and duk.bi_duk_object_resume() ensure all of these are met

17372. E5 Section 11.2.3, step 6.a.i

17373. A -> target register (A, A+1) for call setup * (for DUK_OP_CSREGI, 'a' is indirect) * B -> register containing target function (not type checked here)

17374. Must ensure result is 64-bit (no overflow); a * simple and sufficient fast path is to allow only * 32-bit inputs. Avoid zero inputs to avoid * negative zero issues (-1 * 0 = -0, for instance).

17375. lj.value1 is already set

17376. Dispatch loop.

17377. fall through if overflow etc

17378. this may have side effects, so re-lookup act

17379. Assume interrupt init/counter are properly initialized here.

17380. Continue checked execution if there are breakpoints or we're stepping. * Also use checked execution if paused flag is active - it shouldn't be * because the debug message loop shouldn't terminate if it was. Step out * is handled by callstack unwind and doesn't need checked execution. * Note that debugger may have detached due to error or explicit request * above, so we must recheck attach status.

17381. Set a high interrupt counter; the original executor * interrupt invocation will rewrite before exiting.

17382. The compiler should never emit DUK_OP_REGEXP if there is no * regexp support.

17383. **comment:** XXX: TRYCATCH handling should be reworked to avoid creating * an explicit scope unless it is actually needed (e.g. function * instances or eval is executed inside the catch block). This * rework is not trivial because the compiler doesn't have an * intermediate representation. When the rework is done, the * opcode format can also be made more straightforward.

label: code-design

17384. As a first approximation, buffer values are coerced to strings * for addition. This means that adding two buffers currently * results in a string.

17385. must be ecmascript

17386. **comment:** XXX: does not work if thr->catchstack is NULL

label: code-design

17387. The smallest fastint is no longer 48-bit when * negated. Positive zero becomes negative zero * (cannot be represented) when negated.

17388. allocate catcher and populate it (should be atomic)

17389. E5 Section 11.7.1, steps 7 and 8

17390. No need to check for size of bp_active list, * it's always larger than maximum number of * breakpoints.

17391. YIELDED: Ecmascript activation + Duktape.Thread.yield() activation

17392. * Handling DUK_OP_ADD this way is more compact (experimentally) * than a separate case with separate argument decoding.

17393. INACTIVE: no activation, single function value on valstack

17394. **comment:** * Restart execution by reloading thread state. ** Note that 'thr' and any thread configuration may have changed, * so all local variables are suspect and we need to reinitialize. ** The number of local variables should be kept to a minimum: if * the variables are spilled, they will need to be loaded from * memory anyway. ** Any 'goto restart_execution;' code path in opcode dispatch must * ensure 'curr_pc' is synced back to act->curr_pc before the goto * takes place. ** The interpreter must be very careful with memory pointers, as * many pointers are not guaranteed to be 'stable' and may be * reallocated and relocated on-the-fly quite easily (e.g. by a * memory allocation or a property access). ** The following are assumed to have stable pointers: * - the current thread * - the current function * - the bytecode, constant table, inner function table of the * current function (as they are a part of the function allocation) ** The following are assumed to have semi-stable pointers: * - the current activation entry: stable as long as callstack * is not changed (reallocated by growing or shrinking), or * by any garbage collection invocation (through finalizers) * - Note in particular that ANY DECREF can invalidate the * activation pointer, so for the most part a fresh lookup * is required * * The following are not assumed to have stable pointers at all: * - the value stack (registers) of the current thread * - the catch stack of the current thread * * See execution.rst for discussion.

label: code-design

17395. * Ecmascript bytecode executor. ** Resume execution for the current thread from its current activation. * Returns when execution would return from the entry level activation, * leaving a single return value on top of the stack. Function calls * and thread resumptions are handled internally. If an error occurs, * a longjmp() with type DUK_LJ_TYPE_THROW is called on the entry level * setjmp() jmpbuf. ** Ecmascript function calls and coroutine resumptions are handled * internally (by the outer executor function) without recursive C calls. * Other function calls are handled using duk_handle_call(), increasing * C recursion depth. * * Abrupt completions (= long control transfers) are handled either * directly by reconfiguring relevant stacks and restarting execution, * or via a longjmp. Longjmp-free handling is preferable for performance * (especially Emscripten performance), and is used for: break, continue, * and return. ** For more detailed notes, see doc/execution.rst. ** Also see doc/code-issues.rst for discussion of setjmp(), longjmp(), * and volatile.

17396. Compiler is responsible for selecting property flags (configurability, * writability, etc).

17397. enum_flags

17398. E5 Sections 11.9.1, 11.9.3

17399. * Note: we currently assume that the setjmp() catchpoint is * not re-entrant (longjmp() cannot be called more than once * for a single setjmp()). ** See doc/code-issues.rst for notes on variable assignment * before and after setjmp().

17400. **comment:** * (Re)try handling the longjmp. ** A longjmp handler may convert the longjmp to a different type and * "virtually" rethrow by goto'ing to 'check_longjmp'. Before the goto, * the following must be updated: * - the heap 'lj' state * - 'thr' must reflect the "throwing" thread

label: code-design

17401. Force pause if we were doing "step into" in another activation.

17402. Note: target registers a and a+1 may overlap with DUK_REGP(b) * and DUK_REGCONSTP(c). Careful here.

17403. already set

17404. [... val] -> [... enum]

17405. these are not necessarily 0 or 1 (may be other non-zero), that's ok

17406. +1 = finally
17407. Negative value checked so that a "time jump" works * reasonably. * * Same interval is now used for status sending and * peeking.
17408. Note: 'this' may be bound to any value, not just an object
17409. * Currently only allowed only if yielding thread has only * Ecmascript activations (except for the Duktape.Thread.yield() * call at the callstack top) and none of them constructor * calls. * * This excludes the 'entry' thread which will always have * a preventcount > 0.
17410. + spare
17411. * Process messages and send status if necessary. * * If we're paused, we'll block for new messages. If we're not * paused, we'll process anything we can peek but won't block * for more. Detach (and re-attach) handling is all localized * to duk_debug_process_messages() too. * * Debugger writes outside the message loop may cause debugger * detach1 phase to run, after which dbg_read_cb == NULL and * dbg_detaching != 0. The message loop will finish the detach * by running detach2 phase, so enter the message loop also when * detaching.
17412. must "goto restart_execution", e.g. breakpoints changed
17413. Share yield longjmp handler.
17414. **comment:** * Three possible outcomes: * * A try or finally catcher is found => resume there. * (or) * * The error propagates to the bytecode executor entry * level (and we're in the entry thread) => rethrow * with a new longjmp(), after restoring the previous * catchpoint. * * The error is not caught in the current thread, so * the thread finishes with an error. This works like * a yielded error, except that the thread is finished * and can no longer be resumed. (There is always a * resumer in this case.) * * Note: until we hit the entry level, there can only be * Ecmascript activations.
label: code-design
17415. no need to unwind callstack
17416. unwind to 'yield' caller
17417. 'this binding' is just under bottom
17418. stable
17419. Enter message processing loop for sending Status notifs and * to finish a pending detach.
17420. Duktape.Thread.resume()
17421. +0 = break, +1 = continue
17422. Outer executor with setjmp/longjmp handling.
17423. if debugging disabled
17424. E5 Section 10.4.3
17425. does not modify tv_x
17426. [... func this arg1 ... argN]
17427. Cause an interrupt after executing one instruction.
17428. side effects, perform in-place
17429. 'act' is no longer accessed, scanbuild fix
17430. Ecmascript activation + Duktape.Thread.yield() activation
17431. **comment:** Presence of 'fun' is config based, there's a marginal performance * difference and the best option is architecture dependent.
label: code-design
17432. reg
17433. no compact
17434. restricted to regs
17435. exit bytecode executor by rethrowing an error to caller
17436. **comment:** XXX: side effect handling is quite awkward here
label: code-design
17437. Note: thr->catchstack_top may be 0, so that cat < thr->catchstack * initially. This is OK and intended.
17438. Call setup checks callability.
17439. exit bytecode execution with return value
17440. A -> target reg * B -> object reg/const (may be const e.g. in "'foo'[1]") * C -> key reg/const
17441. There is no eval() special handling here: eval() is never * automatically converted to a lightfunc.
17442. Write curr_pc back for the debugger.
17443. Called for handling both a longjmp() with type DUK_LJ_TYPE_YIELD and * when a RETURN opcode terminates a thread and yields to the resumer.
17444. * Other cases, use C recursion. * * If a tail call was requested we ignore it and execute a normal call. * Since Duktape 0.11.0 the compiler emits a RETURN opcode even after * a tail call to avoid test-bug-tailcall-thread-yield-resume.js. * * Direct eval call: (1) call target (before following bound function * chain) is the built-in eval() function, and (2) call was made with * the identifier 'eval'.
17445. **comment:** Current PC, accessed by other functions through thr->ptr_to_curr_pc. * Critical for performance. It would be safest to make this volatile, * but that eliminates performance benefits; aliasing guarantees * should be enough though.
label: code-design
17446. no refcount changes
17447. 'null' enumerator case -> behave as with an empty enumerator
17448. **comment:** Compared to duk_handle_call(): * - protected call: never * - ignore recursion limit: never
label: code-design
17449. A -> unused (reserved for flags, for consistency with DUK_OP_CALL) * B -> target register and start reg: constructor, arg1, ..., argN * (for DUK_OP_NEWI, 'b' is indirect) * C -> num args (N)
17450. **comment:** XXX: putvar takes a duk_tval pointer, which is awkward and * should be reworked.
label: code-design
17451. bottom of current func
17452. duk__handle_return() is guaranteed never to throw, except * for potential out-of-memory situations which will then * propagate out of the executor longjmp handler.
17453. **comment:** XXX: could use at least one fewer loop counters
label: code-design
17454. A -> flags * B -> return value reg/const * C -> currently unused
17455. Note: left shift, should mask
17456. Ecmascript function
17457. curr_pc synced back above
17458. relookup (side effects)
17459. GH-303
17460. return value from Duktape.Thread.yield()
17461. * Ecmascript modulus (%) does not match IEEE 754 "remainder" * operation (implemented by remainder() in C99) but does seem * to match ANSI C fmod(). * * Compare E5 Section 11.5.3 and "man fmod".
17462. The counter value is one less than the init value: init value should * indicate how many instructions are executed before interrupt. To * execute 1 instruction (after interrupt handler return), counter must * be 0.
17463. unsigned shift
17464. Note: errors are augmented when they are created, not * when they are thrown. So, don't augment here, it would * break re-throwing for instance.
17465. We request a tail call, but in some corner cases * call handling can decide that a tail call is * actually not possible. * See: test-bug-tailcall-preventyield-assert.c.
17466. **comment:** Not necessary to unwind catchstack: return handling will * do it. The finally flag of 'cat' is no longer set. The * catch flag may be set, but it's not checked by return handling.
label: code-design
17467. **comment:** Inner executor, performance critical.
label: code-design
17468. **comment:** This could also be thrown internally (set the error, goto check_longjmp), * but it's better for internal errors to bubble outwards so that we won't * infinite loop in this catchpoint.

label: code-design

17469. * Assuming a register binds to a variable declared within this * function (a declarative binding), the 'this' for the call * setup is always 'undefined'. E5 Section 10.2.1.1.6.

17470. CATCH flag may be enabled or disabled here; it may be enabled if * the statement has a catch block but the try block does not throw * an error.

17471. * If entering a 'catch' block which requires an automatic * catch variable binding, create the lexical environment. * * The binding is mutable (= writable) but not deletable. * Step 4 for the catch production in E5 Section 12.14; * no value is given for CreateMutableBinding 'D' argument, * which implies the binding is not deletable.

17472. XXX: Set 32-bit result (but must then handle signed and * unsigned results separately).

17473. x

17474. lj.value1 already set

17475. A -> object reg * B -> key reg/const * C -> value reg/const * * Note: intentional difference to register arrangement * of e.g. GETPROP; 'A' must contain a register-only value.

17476. at least one activation, ours

17477. XXX: assert idx_base

17478. not caught by anything before entry level; rethrow and let the * final catcher unwind everything

17479. Quite heavy assert: check valstack policy. Improper * shuffle instructions can write beyond valstack_top/end * so this check catches them in the act.

17480. Note: currently the catch binding is handled without a register * binding because we don't support dynamic register bindings (they * must be fixed for an entire function). So, there is no need to * record regbases etc.

17481. * Breakpoint and step state checks

17482. [... env target target]

17483. Don't allow a zero divisor. Restrict both v1 and * v2 to positive values to avoid compiler specific * behavior.

17484. overflow, fall through

17485. restored if ecma-to-ecma setup fails

17486. type

17487. reset longjmp

17488. **comment:** pseudotypes, not used in actual longjumps

label: code-design

17489. borrowed reference; although 'tv1' comes from a register, * its value was loaded using LDCONST so the constant will * also exist and be reachable.

17490. A -> target reg * BC -> inner function index

17491. resumee

17492. **comment:** This macro works when a regconst field is 9 bits, [0,0x1ff]. Adding * DUK_LIKELY/DUK_UNLIKELY increases code footprint and doesn't seem to * improve performance on x64 (and actually harms performance in some tests).

label: code-design

17493. Fast path for the case where the register * is a number (e.g. loop counter).

17494. * Ecmascript bytecode executor.

17495. **comment:** * Four possible outcomes: * * 1. A 'finally' in the same function catches the 'return'. * It may continue to propagate when 'finally' is finished, * or it may be neutralized by 'finally' (both handled by * ENDFIN). * * 2. The return happens at the entry level of the bytecode * executor, so return from the executor (in C stack). * * 3. There is a calling (Ecmascript) activation in the call * stack => return to it, in the same executor instance. * * 4. There is no calling activation, and the thread is * terminated. There is always a resumer in this case, * which gets the return value similarly to a 'yield' * (except that the current thread can no longer be * resumed).

label: code-design

17496. **comment:** XXX: signalling the need to shrink check (only if unwound)

label: code-design

17497. **comment:** 'this' binding is not needed here

label: requirement

17498. DUK_USE_EXEC_TIMEOUT_CHECK

17499. Two lowest bits of opcode are used to distinguish * variants. Bit 0 = inc(0)/dec(1), bit 1 = pre(0)/post(1).

17500. entry level reached

17501. get_value

17502. Note: target registers a and a+1 may overlap with DUK__REGP(b). * Careful here.

17503. **comment:** * Enumeration semantics come from for-in statement, E5 Section 12.6.4. * If called with 'null' or 'undefined', this opcode returns 'null' as * the enumerator, which is special cased in NEXTENUM. This simplifies * the compiler part

label: code-design

17504. signed shift

17505. already declared, must update binding value

17506. **comment:** * To determine whether to use an optimized Ecmascript-to-Ecmascript * call, we need to know whether the final, non-bound function is an * Ecmascript function. * * This is now implemented so that we start to do an ecma-to-ecma call * setup which will resolve the bound chain as the first thing. If the * final function is not eligible, the return value indicates that the * ecma-to-ecma call is not possible. The setup will overwrite the call * target at DUK__REGP(idx) with the final, non-bound function (which * may be a lightfunc), and fudge arguments if necessary. * * XXX: If an ecma-to-ecma call is not possible, this initial call * setup will do bound function chain resolution but won't do the * "effective this binding" resolution which is quite confusing. * Perhaps add a helper for doing bound function and effective this * binding resolution - and call that explicitly? Ecma-to-ecma call * setup and normal function handling can then assume this prestep has * been done by the caller.

label: code-design

17507. ensures callstack_top - 1 >= 0

17508. Don't need to sync curr_pc here; duk_new() will do that * when it augments the created error.

17509. No need to reinit setjmp() catchpoint, as call handling * will store and restore our state.

17510. variable value (function, we hope, not checked here)

17511. x <= y --> not (x > y) --> not (y < x)

17512. mandatory if du.d is a NaN

17513. Typing: use duk_small_(u)int_fast_t when decoding small * opcode fields (op, A, B, C) and duk_(u)int_fast_t when * decoding larger fields (e.g. BC which is 18 bits). Use * unsigned variant by default, signed when the value is used * in signed arithmetic. Using variable names such as 'a', 'b', * 'c', 'bc', etc makes it easier to spot typing mismatches.

17514. **comment:** not needed

label: requirement

17515. **comment:** 'fun' is quite rarely used, so no local for it

label: code-design

17516. Return to the bytecode executor caller which will unwind stacks. * Return value is already on the stack top: [... retval].

17517. idx_retval unsigned

17518. DUK_USE_DEBUGGER_SUPPORT

17519. 'thr' is the current thread, as no-one resumes except us and we * switch 'thr' in that case.

17520. There is a caller; it MUST be an Ecmascript caller (otherwise it would * match entry level check)

17521. 'with' target must be created first, in case we run out of memory

17522. Avoid recursive re-entry; enter when we're attached or * detaching (to finish off the pending detach).

17523. 'this' value: * E5 Section 6.b.i * * The only (standard) case where the 'this' binding is non-null is when * (1) the variable is found in an object environment record, and * (2) that object environment record is a 'with' block. *

17524. not needed, as we exit right away

17525. **comment:** * Binary bitwise operations use different coercions (ToInt32, Uint32) * depending on the operation. We coerce the arguments first using * ToInt32(), and then cast to an 32-bit value if necessary. Note that * such casts must be correct even if there is no native 32-bit type * (e.g., duk_int32_t and duk_uint32_t are

64-bit). ** E5 Sections 11.10, 11.7.1, 11.7.2, 11.7.3

label: code-design

17526. Reconfigure value stack for return to an Ecmascript function at 'act_idx'.

17527. relookup, may have changed

17528. **comment:** unused for label

label: code-design

17529. **comment:** * Arithmetic, binary, and logical helpers. ** Note: there is no opcode for logical AND or logical OR; this is on * purpose, because the evalution order semantics for them make such * opcodes pretty pointless: short circuiting means they are most * comfortably implemented as jumps. However, a logical NOT opcode * is useful. ** Note: careful with duk_tval pointers here: they are potentially * invalidated by any DECREF and almost any API call. It's still * preferable to work without making a copy but that's not always * possible.

label: code-design

17530. **comment:** 'funcs' is quite rarely used, so no local for it

label: code-design

17531. relookup if changed

17532. Shouldn't happen but check anyway.

17533. **comment:** XXX: the best typing needs to be validated by perf measurement: * e.g. using a small type which is the cast to a larger duk_idx_t * may be slower than declaring the variable as a duk_idx_t in the * first place.

label: code-design

17534. -> [... escaped_source bytecode]

17535. result

17536. not protected, respect reclimit, not constructor

17537. [... env target]

17538. * Debugger handling for executor restart ** Check for breakpoints, stepping, etc, and figure out if we should execute * in checked or normal mode. Note that we can't do this when an activation * is created, because breakpoint status (and stepping status) may change * later, so we must recheck every time we're executing an activation. * This primitive should be side effect free to avoid changes during check.

17539. ignore

17540. special result value handling

17541. skip jump slots

17542. Opcode only emitted by compiler when debugger * support is enabled. Ignore it silently without * debugger support, in case it has been loaded * from precompiled bytecode.

17543. We can directly access value stack here.

17544. Set up curr_pc for opcode dispatch.

17545. val

17546. **comment:** XXX: use macros for the repetitive tval/refcount handling.

label: code-design

17547. **comment:** unused

label: code-design

17548. Note: target registers a and a+1 may overlap with DUK__REGCONSTP(b) * and DUK__REGCONSTP(c). Careful here.

17549. Ecmascript activation + Duktape.Thread.resume() activation

17550. always in executor

17551. reachable through activation

17552. thr->heap->lj.value2 is 'thread', will be wiped out at the end

17553. **comment:** XXX: faster initialization (direct access or better primitives)

label: code-design

17554. **comment:** XXX: shared decoding of 'b' and 'c'?

label: code-design

17555. important to use normalized NaN with 8-byte tagged types

17556. +0 = catch

17557. get before side effects

17558. **comment:** Note: combining comparison ops must be done carefully because * of uncomparable values (NaN): it's not necessarily true that * (x >= y) === !(x < y). Also, evaluation order matters, and * although it would only seem to affect the compiler this is * actually not the case, because there are also run-time coercions * of the arguments (with potential side effects). ** XXX: can be combined; check code size.

label: code-design

17559. * E5 Section 11.4.8

17560. always provideThis=true

17561. duk_small_uint_fast_t c = DUK_DEC_C(ins);

17562. **comment:** XXX: fastint fast path would be very useful here

label: code-design

17563. no specific action, resume normal execution

17564. Execute bytecode until returned or longjmp().

17565. Relookup if relocated

17566. Not necessary to unwind catchstack: break/continue * handling will do it. The finally flag of 'cat' is * no longer set. The catch flag may be set, but it's * not checked by break/continue handling.

17567. * Rate limit check for sending status update or peeking into * the debug transport. Both can be expensive operations that * we don't want to do on every opcode. * Making sure the interval remains reasonable on a wide variety * of targets and bytecode is difficult without a timestamp, so * we use a Date-provided timestamp for the rate limit check. * But since it's also expensive to get a timestamp, a bytecode * counter is used to rate limit getting timestamps.

17568. num_stack_args

17569. * Find a matching label catcher or 'finally' catcher in * the same function. ** A label catcher must always exist and will match unless * a 'finally' captures the break/continue first. It is the * compiler's responsibility to ensure that labels are used * correctly.

17570. is resume, not a tail call

17571. avoid referencing, invalidated

17572. +1 = one retval

17573. B -> target register for next key * C -> enum register

17574. Force restart caused by a function return; must recheck * debugger breakpoints before checking line transitions, * see GH-303. Restart and then handle interrupt_counter * zero again.

17575. **comment:** Registers 'bc' and 'bc + 1' are written in longjmp handling * and if their previous values (which are temporaries) become * unreachable -and- have a finalizer, there'll be a function * call during error handling which is not supported now (GH-287). * Ensure that both 'bc' and 'bc + 1' have primitive values to * guarantee no finalizer calls in error handling. Scrubbing also * ensures finalizers for the previous values run here rather than * later. Error handling related values are also written to 'bc' * and 'bc + 1' but those values never become unreachable during * error handling, so there's no side effect problem even if the * error value has a finalizer.

label: code-design

17576. no_block

17577. cat_idx catcher is kept, even for finally

17578. XXX: switch cast?

17579. keep label catcher

17580. state updated, restart bytecode execution

17581. Must be restored here to handle e.g. yields properly.

17582. **comment:** * Addition operator is different from other arithmetic * operations in that it also provides string concatenation. * Hence it is implemented separately. * * There is a fast path for number addition. Other cases go * through potentially multiple coercions as described in the * E5 specification. It may be possible to reduce the number * of coercions, but this must be done carefully to preserve * the exact semantics. * * E5 Section 11.6.1. * * Custom types also have special behavior implemented here.
label: code-design

17583. Longjmp callers are required to sync-and-null thr->ptr_curr_pc * before longjmp.

17584. ToNumber() for a double is a no-op.

17585. [... enum] -> [... next_key]

17586. leave 'cat' as top catcher (also works if catchstack exhausted)

17587. XXX: direct manipulation, or duk_replace_tval()

17588. We'll need to interrupt early so recompute the init * counter to reflect the number of bytecode instructions * executed so that step counts for e.g. debugger rate * limiting are accurate.

17589. A -> register of target object * B -> first register of value data (start_index, value1, value2, ..., valueN) * C -> number of key/value pairs (N)

17590. terminate

17591. unchanged by Duktape.Thread.resume()

17592. unchanged from Duktape.Thread.yield()

17593. A -> result reg * B -> object reg * C -> key reg/const

17594. **comment:** XXX: allow object to be a const, e.g. in 'foo'.toString()? * On the other hand, DUK_REGCONSTP() is slower and generates * more code.
label: code-design

17595. invalidated

17596. [... val obj]

17597. env created above to stack top

17598. -> [Object res]

17599. Trigger at zero or below

17600. 'this' binding

17601. end switch

17602. Successful return: restore jmpbuf and return to caller.

17603. resume caller must be an ecmascript func

17604. y

17605. 'with' binding has no catch clause, so can't be here unless a normal try-catch

17606. caller must change active thread, and set thr->resumer to NULL

17607. unresolvable, no stack changes

17608. [... s1 s2] -> [... s1+s2]

17609. * Throw the error in the resumed thread's context; the * error value is pushed onto the resumee valstack. * * Note: the callstack of the target may empty in this case * too (i.e. the target thread has never been resumed). The * value stack will contain the initial function in that case, * which we simply ignore.

17610. Clamp so that values at 'clamp_top' and above are wiped and won't * retain reachable garbage. Then extend to 'nregs' because we're * returning to an Ecmascript function.

17611. **comment:** not caught by current thread, thread terminates (yield error to resumer); * note that this may cause a cascade if the resumer terminates with an uncaught * exception etc (this is OK, but needs careful testing)
label: code-design

17612. Recompute argument count: bound function handling may have shifted.

17613. **comment:** XXX: declvar takes an duk_tval pointer, which is awkward and * should be reworked.
label: code-design

17614. no prototype, updated below

17615. unwind to 'resume' caller

17616. guarantees entry_callstack_top - 1 >= 0

17617. may be changed

17618. Rethrow error to calling state.

17619. * Update the interrupt counter

17620. Stepped? Step out is handled by callstack unwind.

17621. **comment:** XXX: does not work if thr->catchstack is allocated but lowest pointer
label: code-design

17622. relookup after side effects (no side effects currently however)

17623. resume: [... initial_func undefined(= this) resume_value]

17624. must relookup act in case of side effects

17625. fastint downgrade check for return values

17626. XXX: optimize reconfig valstack operations so that resize, clamp, and setting * top are combined into one pass.

17627. **comment:** Figure out all active breakpoints. A breakpoint is * considered active if the current function's fileName * matches the breakpoint's fileName, AND there is no * inner function that has matching line numbers * (otherwise breakpoint would be triggered both * inside and outside of the inner function which would * be confusing). Example: * * function foo() { print('foo'); * function bar() { <. breakpoints in these * print('bar'); | lines should not affect * } <- foo() execution * bar(); * } * * We need a few things that are only available when * debugger support is enabled: (1) a line range for * each function, and (2) access to the function * template to access the inner functions (and their * line ranges). * * It's important to have a narrow match for active * breakpoints so that we don't enter checked execution * when that's not necessary. For instance, if we're * running inside a certain function and there's * breakpoint outside in (after the call site), we * don't want to slow down execution of the function.
label: code-design

17628. pre-incremented, points to first jump slot

17629. nop

17630. * NEXTENUM checks whether the enumerator still has unenumerated * keys. If so, the next key is loaded to the target register * and the next instruction is skipped. Otherwise the next instruction * will be executed, jumping out of the enumeration loop.

17631. * Note: lj.value1 is 'value', lj.value2 is 'resume'. * This differs from YIELD.

17632. In-place unary operation.

17633. **comment:** Lookup current thread; use the stable 'entry_thread' for this to * avoid clobber warnings. Any valid, reachable 'thr' value would be * fine for this, so using 'entry_thread' is just to silence warnings.
label: code-design

17634. Thread may have changed, e.g. YIELD converted to THROW.

17635. throw

17636. unreferenced without assertions

17637. For cross-checking during development: ensure dispatch count * matches cumulative interrupt counter init value sums.

17638. valstack should not need changes

17639. [-0x80000000,0x7fffffff]

17640. Catches both doubles and cases where only one argument is a fastint

17641. x == -Infinity

17642. * Note: prototype chain is followed BEFORE first comparison. This * means that the instanceof lval is never itself compared to the * rval.prototype property. This is apparently intentional, see E5 * Section 15.3.5.3, step 4.a. * * Also note: * * js> (function() {}) instanceof Function * true * js> Function instanceof Function * true * * For the latter, h_proto will be Function.prototype, which is the * built-in Function prototype. Because Function.[[Prototype]] is * also the built-in Function prototype, the result is true.

17643. heap pointer comparison suffices

17644. E5 Section 9.3.1

17645. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack (e.g. if * lval is already a string).
17646. truncate towards zero
17647. func support for [[HasInstance]] checked in the beginning of the loop
17648. prefix matches, lengths matter now
17649. Note: undefined from Section 11.8.5 always results in false * return (see e.g. Section 11.8.3) - hence special treatment here.
17650. normal comparison; known: * - both x and y are not NaNs (but one of them can be) * - both x and y are not zero (but one of them can be) * - x and y may be denormal or infinite
17651. mimic semantics for strings
17652. **comment:** At least 'magic' has a significant impact on function * identity.
 label: code-design
17653. implementation specific
17654. x == +Infinity
17655. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack. * * The right hand side could be a light function (as they generally * behave like objects). Light functions never have a 'prototype' * property so E5.1 Section 15.3.5.3 step 3 always throws a TypeError. * * Using duk_require_hobject() is thus correct (except for error msg).
17656. overflow, more than 32 bits -> not an array index
17657. -> x in J-2**32, 2**32[
17658. String-number-buffer/object -> coerce object to primitive (apparently without hint), then try again.
17659. exposed, used by e.g. duk.bi_date.c
17660. Buffer/string -> compare contents.
17661. * ToNumber() (E5 Section 9.3) * * Value to convert must be on stack top, and is popped before exit. * * See: <http://www.cs.indiana.edu/~burger/FP-Printing-PLDI96.pdf> * <http://www.cs.indiana.edu/~burger/fp/index.html> * * Notes on the conversion: * * - There are specific requirements on the accuracy of the conversion * through a "Mathematical Value" (MV), so this conversion is not * trivial. * * - Quick rejects (e.g. based on first char) are difficult because * the grammar allows leading and trailing white space. * * - Quick reject based on string length is difficult even after * accounting for white space; there may be arbitrarily many * decimal digits. * * - Standard grammar allows decimal values ("123"), hex values * ("0x123") and infinities * * - Unlike source code literals, ToNumber() coerces empty strings * and strings with only whitespace to zero (not NaN).
17662. **comment:** Straightforward algorithm, makes fewer compiler assumptions.
 label: code-design
17663. Handle lightfuncs through object coercion for now.
17664. **comment:** unused
 label: code-design
17665. x = sign(x) * floor(abs(x)), i.e. truncate towards zero, keep sign
17666. Fast path for numbers (one of which may be a fastint)
17667. TypeError if rval is not an object (or lightfunc which should behave * like a Function instance).
17668. number
17669. radix
17670. Number/string-or-buffer -> coerce string to number (e.g. "'1.5' == 1.5" -> true).
17671. Undefined/null are considered equal (e.g. "null == undefined" -> true).
17672. **comment:** * ToString() (E5 Section 9.8) * * ==> implemented in the API.
 label: requirement
17673. * 'func' is now a non-bound object which supports [[HasInstance]] * (which here just means DUK_HOBJECT_FLAG_CALLABLE). Move on * to execute E5 Section 15.3.5.3.
17674. **comment:** XXX: FP_ZERO check can be removed, the else clause handles it * correctly (preserving sign).
 label: code-design
17675. **comment:** XXX: direct implementation
 label: requirement
17676. SameValue(NaN, NaN) = true, regardless of NaN sign or extra bits
17677. For all combinations: +0 < +0, +0 < -0, -0 < +0, -0 < -0, * steps e, f, and g.
17678. **comment:** XXX: make fast paths optional for size minimization?
 label: code-design
17679. **comment:** XXX: expensive, but numconv now expects to see a string
 label: code-design
17680. * ToInteger() (E5 Section 9.4)
17681. invalidates tv
17682. * E5 Section 11.8.6 describes the main algorithm, which uses * [[HasInstance]]. [[HasInstance]] is defined for only * function objects: * * - Normal functions: * E5 Section 15.3.5.3 * - Functions established with Function.prototype.bind(): * E5 Section 15.3.4.5.3 * * For other objects, a TypeError is thrown. * * Limited Proxy support: don't support 'getPrototypeOf' trap but * continue lookup in Proxy target if the value is a Proxy.
17683. never here
17684. Note that this is the same operation for strict and loose equality: * - E5 Section 11.9.3, step 1.c (loose) * - E5 Section 11.9.6, step 4 (strict)
17685. DUK_MEMCMP() is guaranteed to return zero (equal) for zero length * inputs so no zero length check is needed.
17686. y == +Infinity
17687. * Same type? * * Note: since number values have no explicit tag in the 8-byte * representation, need the awkward if + switch.
17688. **comment:** Note: reuse variables
 label: code-design
17689. * CheckObjectCoercible() (E5 Section 9.10) * * Note: no API equivalent now.
17690. **comment:** Coerce like a string. This makes sense because addition also treats * buffers like strings.
 label: code-design
17691. NOTE: fmod(x) result sign is same as sign of x, which * differs from what Javascript wants (see Section 9.6).
17692. equals and strict equals
17693. non-strict equality for buffers compares contents
17694. +(function(){}) -> NaN
17695. * Loose equality, strict equality, and SameValue (E5 Sections 11.9.1, 11.9.4, * 9.12). These have much in common so they can share some helpers. * * Future work notes: * * - Current implementation (and spec definition) has recursion; this should * be fixed if possible. * * - String-to-number coercion should be possible without going through the * value stack (and be more compact) if a shared helper is invoked.
17696. Called by duk_hstring.h macros
17697. non-strict equality from here on
17698. +0.0
17699. Better equivalent algorithm. If the compiler is compliant, C and * EcmaScript semantics are identical for this particular comparison. * In particular, NaNs must never compare equal and zeroes must compare * equal regardless of sign. Could also use a macro, but this inlines * already nicely (no difference on gcc, for instance).
17700. * E5 Sections 11.8.7, 8.12.6. * * Basically just a property existence check using [[HasProperty]].
17701. Boolean/any -> coerce boolean to number and try again. If boolean is * compared to a pointer, the final comparison after coercion now always * yields false (as pointer vs. number compares to false), but this is * not special cased.
17702. -> [... lval new_rval]
17703. Using classification has smaller footprint than direct comparison.
17704. coerce lval with ToString()
17705. **comment:** * typeof * * E5 Section 11.4.3. * * Very straightforward. The only question is what to return for our * non-standard tag / object types. * * There is an unfortunate string constant define naming problem with * typeof return values for e.g. "Object" and "object"; careful with * the built-in string defines. The

LC_XXX defines are used for the * lowercase variants now.

label: code-design

17706. IEEE requires that zeros compare the same regardless * of their signed, so if both x and y are zeroes, they * are caught above.

17707. * ToObject() (E5 Section 9.9) * * ==> implemented in the API.

17708. * ToBoolean() (E5 Section 9.2)

17709. follow prototype chain

17710. Difference to non-strict/strict comparison is that NaNs compare * equal and signed zero signs matter.

17711. Ordering should not matter (E5 Section 11.8.5, step 3.a) but * preserve it just in case.

17712. combined algorithm matching E5 Sections 9.5 and 9.6

17713. **comment:** XXX: this is possible without resorting to the value stack

label: code-design

17714. Similar to String comparison.

17715. Slow path

17716. **comment:** XXX: this is a very narrow check, and doesn't cover * zeroes, subnormals, infinities, which compare normally.

label: code-design

17717. **comment:** * Ecmascript specification algorithm and conversion helpers. * * These helpers encapsulate the primitive Ecmascript operation * semantics, and are used by the bytecode executor and the API * (among other places). Note that some primitives are only * implemented as part of the API and have no "internal" helper. * (This is the case when an internal helper would not really be * useful; e.g. the operation is rare, uses value stack heavily, * etc.) * * The operation arguments depend on what is required to implement * the operation: * * - If an operation is simple and stateless, and has no side * effects, it won't take an duk_hthread argument and its * arguments may be duk_tval pointers (which are safe as long * as no side effects take place). * * - If complex coercions are required (e.g. a "ToNumber" coercion) * or errors may be thrown, the operation takes an duk_hthread * argument. This also implies that the operation may have * arbitrary side effects, invalidating any duk_tval pointers. * * - For operations with potential side effects, arguments can be * taken in several ways: * * a) as duk_tval pointers, which makes sense if the "common case" * can be resolved without side effects (e.g. coercion); the * arguments are pushed to the valstack for coercion if * necessary * * b) as duk_tval values * * c) implicitly on value stack top * * d) as indices to the value stack * * Future work: * * - Argument styles may not be the most sensible in every case now. * * - In-place coercions might be useful for several operations, if * in-place coercion is OK for the bytecode executor and the API.

label: code-design

17718. recursive call for a primitive value (guaranteed not to cause second * recursion).

17719. **comment:** 'tv' becomes invalid

label: code-design

17720. y == -Infinity

17721. buffer values are coerced first to string here

17722. DUK_USE_FASTINT

17723. -> x in [-2**31,2**31[

17724. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)

17725. * [[DefaultValue]] (E5 Section 8.12.8) * * ==> implemented in the API.

17726. -> [... lval rval new_rval]

17727. * instanceof

17728. **comment:** * ToPrimitive() (E5 Section 9.1) * * ==> implemented in the API.

label: requirement

17729. Quite lenient, e.g. allow empty as zero, but don't allow trailing * garbage.

17730. IEEE requires that NaNs compare false

17731. * Comparisons (x >= y, x > y, x <= y, x < y) * * E5 Section 11.8.5: implement 'x < y' and then use negate and eval_left_first * flags to get the rest.

17732. Note: ToPrimitive(object,hint) == [[DefaultValue]][(object,hint)], * so use [[DefaultValue]] directly.

17733. XXX: this bound function resolution also happens elsewhere, * move into a shared helper.

17734. **comment:** * Note: of native Ecmascript objects, only Function instances * have a [[HasInstance]] internal property. Custom objects might * also have it, but not in current implementation. * * XXX: add a separate flag, DUK_HOBJECT_FLAG_ALLOW_INSTANCEOF?

label: code-design

17735. [... lval rval]

17736. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

17737. ToNumber(bool) is +1.0 or 0.0. Tagged boolean value is always 0 or 1.

17738. DUK_USE_PARANOIA_MATH

17739. * For bound objects, [[HasInstance]] just calls the target function * [[HasInstance]]. If that is again a bound object, repeat until * we find a non-bound Function object.

17740. -> x in [0, 2**32[

17741. **comment:** * IsCallable() (E5 Section 9.11) * * XXX: API equivalent is a separate implementation now, and this has * currently no callers.

label: code-design

17742. [0x00000000, 0xffffffff]

17743. If flags != 0 (strict or SameValue), thr can be NULL. For loose * equals comparison it must be != NULL.

17744. **comment:** Note: not a typo, "object" is returned for a null value

label: documentation

17745. **comment:** XXX: fastint

label: code-design

17746. should be a safe way to compute this

17747. x in [2**31, 2**32[

17748. Note: this must match ToObject() behavior

17749. Coerce like boolean

17750. e.g. "x" < "xx"

17751. **comment:** XXX: The ES5/5.1/6 specifications require that the key in 'key in obj' * must be string coerced before the internal HasProperty() algorithm is * invoked. A fast path skipping coercion could be safely implemented for * numbers (as number-to-string coercion has no side effects). For ES6 * proxy behavior, the trap 'key' argument must be in a string coerced * form (which is a shame).

label: code-design

17752. It is true'

17753. *ToInt32(), ToUint32(), ToUint16() (E5 Sections 9.5, 9.6, 9.7)

17754. SameValue

17755. Nothing worked -> not equal.

17756. -> [... lval rval]

17757. Accept 32-bit decimal integers, no leading zeroes, signs, etc. * Leading zeroes are not accepted (zero index "0" is an exception * handled above).

17758. the next 'if' is guaranteed to match after swap

17759. flags:nonstrict

17760. [... lval rval(func)]

17761. **comment:** * Array index and length * * Array index: E5 Section 15.4 * Array length: E5 Section 15.4.5.1 steps 3.c - 3.d (array length write) * * The DUK_HSTRING_GET_ARRIDX_SLOW() and DUK_HSTRING_GET_ARRIDX_FAST() macros * call duk_js_to_arrayindex_string_helper().

label: code-design

17762. Fast path for fastints

17763. **comment:** return a specific NaN (although not strictly necessary)

label: code-design

17764. whole, won't clip

17765. * String comparison (E5 Section 11.8.5, step 4), which * needs to compare codepoint by codepoint. ** However, UTF-8 allows us to use strcmp directly: the shared * prefix will be encoded identically (UTF-8 has unique encoding) * and the first differing character can be compared with a simple * unsigned byte comparison (which strcmp does). ** This will not work properly for non-xutf-8 strings, but this * is not an issue for compliance.

17766. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: ** signbit(x) == signbit(y)

17767. check func supports [[HasInstance]] (this is checked for every function * in the bound chain, including the final one)

17768. * Types are different; various cases for non-strict comparison ** Since comparison is symmetric, we use a "swap trick" to reduce * code size.

17769. **comment:** XXX: this should probably just operate on the stack top, because it * needs to push stuff on the stack anyway...

label: code-design

17770. * in

17771. -> [... lval rval rval.prototype]

17772. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

17773. apparently no hint?

17774. -> [... closure template newobj closure]

17775. * Make a value copy of the input val. This ensures that * side effects cannot invalidate the pointer.

17776. func is NULL for lightfuncs

17777. **comment:** XXX: any way to detect faster whether something needs to be closed? * We now look up _Callee and then skip the rest.

label: requirement

17778. formal arg limits

17779. Note: getprop may invoke any getter and invalidate any * duk_tval pointers, so this must be done first.

17780. * GETVAR ** See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is GetValue'd] * 8.7.1 GetValue (V) * 8.12.1 [[GetOwnProperty]] (P) * 8.12.2 [[GetProperty]] (P) * 8.12.3 [[Get]] (P) ** If 'throw' is true, always leaves two values on top of stack: [val this]. ** If 'throw' is false, returns 0 if identifier cannot be resolved, and the * stack will be unaffected in this case. If identifier is resolved, returns * 1 and leaves [val this] on top of stack. ** Note: the 'strict' flag of a reference returned by GetIdentifierReference * is ignored by GetValue. Hence we don't take a 'strict' parameter. ** The 'throw' flag is needed for implementing 'typeof' for an unreferenced * identifier. An unreferenced identifier in other contexts generates a * ReferenceError.

17781. * In strict mode E5 protects 'eval' and 'arguments' from being * assigned to (or even declared anywhere). Attempt to do so * should result in a compile time SyntaxError. See the internal * design documentation for details. ** Thus, we should never come here, run-time, for strict code, * and name 'eval' or 'arguments'.

17782. **comment:** * Prototype walking starting from 'env'. ** ('act' is not needed anywhere here.)

label: code-design

17783. [... closure template]

17784. [... closure template env]

17785. **comment:** * "prototype" is, by default, a fresh object with the "constructor" * property. ** Note that this creates a circular reference for every function * instance (closure) which prevents refcount-based collection of * function instances. ** XXX: Try to avoid creating the default prototype object, because * many functions are not used as constructors and the default * prototype is unnecessary. Perhaps it could be created on-demand * when it is first accessed?

label: code-design

17786. [holder name val] -> [holder]

17787. NB: 'val' may be invalidated here because put_value may realloc valstack, * caller beware.

17788. * Finish

17789. **comment:** * "arguments" and "caller" must be mapped to throwers for strict * mode and bound functions (E5 Section 15.3.5). ** XXX: This is expensive to have for every strict function instance. * Try to implement as virtual properties or on-demand created properties.

label: code-design

17790. lookup name from an open declarative record's registers

17791. env and act may be NULL

17792. [... env callee]

17793. registers are mutable, non-deletable

17794. side effects

17795. -> [... closure template newobj]

17796. * Local result type for duk__get_identifier_reference() lookup.

17797. Identifier found in registers (always non-deletable) * or declarative environment record and non-configurable.

17798. [... env callee varmap key val]

17799. -> [... closure template]

17800. * PUTVAR ** See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is PutValue'd] * 8.7.2 PutValue (V,W) [see especially step 3.b, undefined -> automatic global in non-strict mode] * 8.12.4 [[CanPut]] (P) * 8.12.5 [[Put]] (P) ** Note: may invalidate any valstack (or object) duk_tval pointers because * putting a value may reallocate any object or any valstack. Caller beware.

17801. must be found: was found earlier, and cannot be inherited

17802. * Not found: write to global object (non-strict) or ReferenceError * (strict); see E5 Section 8.7.2, step 3.

17803. [... closure]

17804. [... env]

17805. [this value] -> [value this]

17806. * Object environment record. ** Binding (target) object is an external, uncontrolled object. * Identifier may be bound in an ancestor property, and may be * an accessor. Target can also be a Proxy which we must support * here.

17807. non-standard

17808. Note: in strict mode the compiler should reject explicit * declaration of 'eval' or 'arguments'. However, internal * bytecode may declare 'arguments' in the function prologue. * We don't bother checking (or asserting) for these now.

17809. always for register bindings

17810. If function creation fails due to out-of-memory, the data buffer * pointer may be NULL in some cases. That's actually possible for * GC code, but shouldn't be possible here because the incomplete * function will be unwound from the value stack and never instantiated.

17811. can't get value, may be accessor

17812. unsigned

17813. writable but not deletable

17814. may be NULL

17815. assertions: env must be closed in the same thread as where it runs

17816. * "length" maps to number of formals (E5 Section 13.2) for function * declarations/expressions (non-bound functions). Note that 'nargs' * is NOT necessarily equal to the number of arguments.

17817. 1 -> needs a PUTVAR

17818. just check 'env'

17819. It's important that duk_xdef_prop() is a 'raw define' so that any * properties in an ancestor are never an issue (they should never be * e.g. non-writable, but just in case).

17820. * Named function expression, name needs to be bound * in an intermediate environment record. The "outer" * lexical/variable environment will thus be: * * a) { funcname: <func>, __prototype: outer_lex_env } * b) { funcname: <func>, __prototype: <globalenv> } (if outer_lex_env missing)

17821. global object doesn't have array part

17822. * Define new property * * Note: this may fail if the holder is not extensible.

17823. * HASVAR: check identifier binding from a given environment record * without traversing its parents. ** This primitive is not exposed to user code as such, but is used * internally for e.g. declaration binding instantiation. ** See E5 Sections: * 10.2.1.1 HasBinding(N) * 10.2.1.2 HasBinding(N) ** Note: strictness has no bearing on this check. Hence we don't take * a 'strict' parameter.

17824. lookup name from current activation record's functions' registers

17825. already closed

17826. -> [... closure template env]
17827. follow parent chain
17828. **comment:** XXX: this is awkward as we use an internal method which doesn't handle * extensibility etc correctly. Basically we'd want to do a [[DefineOwnProperty]] * or Object.defineProperty() here.
label: code-design
17829. **comment:** * Compact the closure, in most cases no properties will be added later. * Also, without this the closures end up having unused property slots * (e.g. in Duktape 0.9.0, 8 slots would be allocated and only 7 used). * A better future solution would be to allocate the closure directly * to correct size (and setup the properties directly without going * through the API).
label: code-design
17830. required if NAMEBINDING set
17831. [... closure template formals]
17832. env[funcname] = closure
17833. Note: we rely on the _Varmap having a bunch of nice properties, like: * - being compacted and unmodified during this process * - not containing an array part * - having correct value types
17834. * GETIDREF: a GetIdentifierReference-like helper. * * Provides a parent traversing lookup and a single level lookup * (for HasBinding). * * Instead of returning the value, returns a bunch of values allowing * the caller to read, write, or delete the binding. Value pointers * are duk_tval pointers which can be mutated directly as long as * refcounts are properly updated. Note that any operation which may * reallocate valstacks or compact objects may invalidate the returned * duk_tval (but not object) pointers, so caller must be very careful. * * If starting environment record 'env' is given, 'act' is ignored. * However, if 'env' is NULL, the caller may identify, in 'act', an * activation which hasn't had its declarative environment initialized * yet. The activation registers are then looked up, and its parent * traversed normally. * * The 'out' structure values are only valid if the function returns * success (non-zero).
17835. Update duk_tval in-place if pointer provided and the * property is writable. If the property is not writable * (immutable binding), use duk_hobject_putprop() which * will respect mutability.
17836. regnum is same
17837. DUK_HOBJECT_FLAG_ARRAY_PART: don't care
17838. XXX: additional conditions when to close variables? we don't want to do it * unless the environment may have "escaped" (referenced in a function closure). * With delayed environments, the existence is probably good enough of a check.
17839. Note: all references inside 'data' need to get their refcounts * upped too. This is the case because refcounts are decreased * through every function referencing 'data' independently.
17840. * Lookup variable and update in-place if found.
17841. -> [... closure template env funcname]
17842. [... closure template val]
17843. **comment:** order: most frequent to least frequent
label: code-design
17844. * Not found in registers, proceed to the parent record. * Here we need to determine what the parent would be, * if 'env' was not NULL (i.e. same logic as when initializing * the record). * * Note that environment initialization is only deferred when * DUK_HOBJECT_HAS_NEWENV is set, and this only happens for: * - Function code * - Strict eval code * * We only need to check _Lexenv here; _Varenv exists only if it * differs from _Lexenv (and thus _Lexenv will also be present).
17845. * Not found (even in global object). * * In non-strict mode this is a silent SUCCESS (!), see E5 Section 11.4.1, * step 3.b. In strict mode this case is a compile time SyntaxError so * we should not come here.
17846. XXX: incref by count (here 2 times)
17847. * Not found (in registers or record objects). Declare * to current variable environment.
17848. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
17849. implicit this value always undefined for * declarative environment records.
17850. * Conceptually, we look for the identifier binding by starting from * 'env' and following to chain of environment records (represented * by the prototype chain). * * If 'env' is NULL, the current activation does not yet have an * allocated declarative environment record; this should be treated * exactly as if the environment record existed but had no bindings * other than register bindings. * * Note: we assume that with the DUK_HOBJECT_FLAG_NEWENV cleared * the environment will always be initialized immediately; hence * a NULL 'env' should only happen with the flag set. This is the * case for: (1) function calls, and (2) strict, direct eval calls.
17851. * Not found (even in global object)
17852. * Create a new function object based on a "template function" which contains * compiled bytecode, constants, etc, but lacks a lexical environment. * * Ecmascript requires that each created closure is a separate object, with * its own set of editable properties. However, structured property values * (such as the formal arguments list and the variable map) are shared. * Also the bytecode, constants, and inner functions are shared. * * See E5 Section 13.2 for detailed requirements on the function objects; * there are no similar requirements for function "templates" which are an * implementation dependent internal feature. Also see function-objects.rst * for a discussion on the function instance properties provided by this * implementation. * * Notes: * * * Order of internal properties should match frequency of use, since the * properties will be linearly scanned on lookup (functions usually don't * have enough properties to warrant a hash part). * * * The created closure is independent of its template; they do share the * same 'data' buffer object, but the template object itself can be freed * even if the closure object remains reachable.
17853. **comment:** * Identifier access and function closure handling. * * Provides the primitives for slow path identifier accesses: GETVAR, * PUTVAR, DELVAR, etc. The fast path, direct register accesses, should * be used for most identifier accesses. Consequently, these slow path * primitives should be optimized for maximum compactness. * * Ecmascript environment records (declarative and object) are represented * as internal objects with control keys. Environment records have a * parent record ("outer environment reference") which is represented by * the implicit prototype for technical reasons (in other words, it is a * convenient field). The prototype chain is not followed in the ordinary * sense for variable lookups. * * See identifier-handling.rst for more details on the identifier algorithms * and the internal representation. See function-objects.rst for details on * what function templates and instances are expected to look like. * * Care must be taken to avoid duk_tval pointer invalidation caused by * e.g. value stack or object resizing. * * TODO: properties for function instances could be initialized much more * efficiently by creating a property allocation for a certain size and * filling in keys and values directly (and INCREFing both with "bulk incref" * primitives. * * XXX: duk_hobject_getprop() and duk_hobject_putprop() calls are a bit * awkward (especially because they follow the prototype chain); rework * if "raw" own property helpers are added.
label: code-design
17854. **comment:** * Variable already declared, ignore re-declaration. * The only exception is the updated behavior of E5.1 for * global function declarations, E5.1 Section 10.5, step 5.e. * This behavior does not apply to global variable declarations.
label: code-design
17855. assume plain values
17856. SCANBUILD: scan-build produces a NULL pointer dereference warning * below; it never actually triggers because holder is actually never * NULL.
17857. * Function gets no new environment when called. This is the * case for global code, indirect eval code, and non-strict * direct eval code. There is no direct correspondence to the * E5 specification, as global/eval code is not exposed as a * function.
17858. * Some assertions (E5 Section 13.2).
17859. SCANBUILD: NULL pointer dereference, doesn't actually trigger, * asserted above.
17860. ref.value and ref.this_binding invalidated here
17861. * DELVAR * * See E5 Sections: * 11.4.1 The delete operator * 10.2.1.1.5 DeleteBinding (N) [declarative environment record] * 10.2.1.2.5 DeleteBinding (N) [object environment record] * * Variable bindings established inside eval() are deletable (configurable), * other bindings are not, including variables declared in global level. * Registers are always non-deletable, and the deletion of other bindings * is controlled by the configurable flag. * * For strict mode code, the 'delete' operator should fail with a compile * time SyntaxError if applied to identifiers. Hence, no strict mode * run-time deletion of identifiers should ever happen. This function * should never be called from strict mode code!
17862. * Other cases (function declaration, anonymous function expression, * strict direct eval code). The "outer" environment will be whatever * the caller gave us.
17863. **comment:** * Special behavior in E5.1. * * Note that even though parents == 0, the conflicting property * may be an inherited property (currently our global object's * prototype is Object.prototype). Step 5.e first operates on * the existing property (which is potentially in an ancestor) * and then defines a new property in the global object (and * never modifies the ancestor). * * Also note that this logic would become even more complicated * if the conflicting property might be a virtual one. Object * prototype has no virtual properties, though. * * XXX: this is now very awkward, rework.

label: code-design

17864. * Setup environment record properties based on the template and * its flags. * * If DUK_HOBJECT_HAS_NEWENV(fun_temp) is true, the environment * records represent identifiers "outside" the function; the * "inner" environment records are created on demand. Otherwise, * the environment records are those that will be directly used * (e.g. for declarations). * * _Lexenv is always set; _Varenv defaults to _Lexenv if missing, * so _Varenv is only set if _Lexenv != _Varenv. * * This is relatively complex, see doc/identifier-handling.rst.

17865. **comment:** XXX: we could save space by using _Target OR _This. If _Target, assume * this binding is undefined. If _This, assumes this binding is _This, and * target is also _This. One property would then be enough.

label: code-design

17866. Push a new closure on the stack. * * Note: if fun_temp has NEWENV, i.e. a new lexical and variable declaration * is created when the function is called, only outer_lex_env matters * (outer_var_env is ignored and may or may not be same as outer_lex_env).

17867. * Declarative environment record. * * Identifiers can never be stored in ancestors and are * always plain values, so we can use an internal helper * and access the value directly with an duk_tval ptr. * * A closed environment is only indicated by it missing * the "book-keeping" properties required for accessing * register-bound variables.

17868. **comment:** * "name" is a non-standard property found in at least V8, Rhino, smjs. * For Rhino and smjs it is non-writable, non-enumerable, and non-configurable; * for V8 it is writable, non-enumerable, non-configurable. It is also defined * for an anonymous function expression in which case the value is an empty string. * We could also leave name 'undefined' for anonymous functions but that would * differ from behavior of other engines, so use an empty string. * * XXX: make optional? costs something per function.

label: code-design

17869. **comment:** XXX: use helper for size optimization

label: code-design

17870. * Delayed activation environment record initialization (for functions * with NEWENV). * * The non-delayed initialization is handled by duk_handle_call().

17871. [... closure template undefined]

17872. **comment:** XXX: could also copy from template, but there's no way to have any * other value here now (used code has no access to the template).

label: code-design

17873. Note: env_thr != thr is quite possible and normal, so careful * with what thread is used for valstack lookup.

17874. ref.value, ref.this.binding invalidated here by getprop call

17875. lookup results is ignored

17876. **comment:** XXX: duk_hobject_hasprop() would be correct for * non-Proxy objects too, but it is about ~20-25% * slower at present so separate code paths for * Proxy and non-Proxy now.

label: code-design

17877. [... env callee varmap]

17878. env is closed, should be missing _Callee, _Thread, _Regbase

17879. **comment:** * Copy some internal properties directly * * The properties will be writable and configurable, but not enumerable.

label: code-design

17880. **comment:** XXX: use a helper for prototype traversal; no loop check here

label: code-design

17881. ref.holder is global object, holder is the object with the * conflicting property.

17882. * Get holder object

17883. for object-bound identifiers

17884. DUK_HOBJECT_FLAG_NEWENV: handled below

17885. Target may be a Proxy or property may be an accessor, so we must * use an actual, Proxy-aware hasprop check here. * * out->holder is NOT set to the actual duk_hobject where the * property is found, but rather the object binding target object.

17886. **comment:** unused

label: code-design

17887. holder will be set to the target object, not the actual object * where the property was found (see duk_get_identifier_reference()).

17888. * Delayed env creation check

17889. * Delayed initialization only occurs for 'NEWENV' functions.

17890. no prototype, updated below

17891. [... closure template len_value]

17892. [this value]

17893. Here val would be potentially invalid if we didn't make * a value copy at the caller.

17894. **comment:** XXX: slightly awkward

label: code-design

17895. implicit this value always undefined for * declarative environment records.

17896. property attributes for identifier (relevant if value != NULL)

17897. 0 = no throw

17898. * Closing environment records. * * The environment record MUST be closed with the thread where its activation * is. In other words (if 'env' is open): * * - 'thr' must match _env.thread * - 'func' must match _env.callee * - 'regbase' must match _env.regbase * * These are not looked up from the env to minimize code size. * * XXX: should access the own properties directly instead of using the API

17899. * Init/assert flags, copying them where appropriate. Some flags * (like NEWENV) are processed separately below.

17900. compact the prototype

17901. assume keys are compacted

17902. * Try registers

17903. * DECLVAR * * See E5 Sections: * 10.4.3 Entering Function Code * 10.5 Declaration Binding Instantiation * 12.2 Variable Statement * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * * Variable declaration behavior is mainly discussed in Section 10.5, * and is not discussed in the execution semantics (Sections 11-13). * * Conceptually declarations happen when code (global, eval, function) * is entered, before any user code is executed. In practice, register- * bound identifiers are 'declared' automatically (by virtue of being * allocated to registers with the initial value 'undefined'). Other * identifiers are declared in the function prologue with this primitive. * * Since non-register bindings eventually back to an internal object's * properties, the 'prop_flags' argument is used to specify binding * type: * * - Immutable binding: set DUK_PROPDESC_FLAG_WRITABLE to false * - Non-deletable binding: set DUK_PROPDESC_FLAG_CONFIGURABLE to false * - The flag DUK_PROPDESC_FLAG_ENUMERABLE should be set, although it * doesn't really matter for internal objects * * All bindings are non-deletable mutable bindings except: * * - Declarations in eval code (mutable, deletable) * - 'arguments' binding in strict function code (immutable) * - Function name binding of a function expression (immutable) * * Declarations may go to declarative environment records (always * so for functions), but may also go to object environment records * (e.g. global code). The global object environment has special * behavior when re-declaring a function (but not a variable); see * E5.1 specification, Section 10.5, step 5.e. * * Declarations always go to the 'top-most' environment record, i.e. * we never check the record chain. It's not an error even if a * property (even an immutable or non-deletable one) of the same name * already exists. * * If a declared variable already exists, its value needs to be updated * (if possible). Returns 1 if a PUTVAR needs to be done by the caller; * otherwise returns 0.

17904. for register-bound and declarative env identifiers

17905. [... closure template name]

17906. * Check whether already declared. * * We need to check whether the binding exists in the environment * without walking its parents. However, we still need to check * register-bound identifiers and the prototype chain of an object * environment target object.

17907. bound functions are never in act 'func'

17908. **comment:** XXX: copy flags using a mask

label: code-design

17909. Although NAMEBINDING is not directly needed for using * function instances, it's needed by bytecode dump/load * so copy it too.

17910. **comment:** XXX: more accurate?

label: code-design

17911. irrelevant when out->value == NULL

17912. **comment:** XXX: if duk_hobject_define_property_internal() was updated * to handle a pre-existing accessor property, this would be * a simple call (like for the ancestor case).
label: code-design

17913. Note: not set in template (has no "prototype")
17914. -> [... closure template env funcname closure]
17915. assume value is a number
17916. **comment:** XXX: what to do if _Formals is not empty but compiler has * optimized it away -- read length from an explicit property * then?
label: code-design

17917. if property already exists, overwrites silently
17918. Note: val is a stable duk_tval pointer. The caller makes * a value copy into its stack frame, so 'tv_val' is not subject * to side effects here.
17919. parent env is the prototype
17920. open scope information, for compiled functions only
17921. shared helper
17922. explicit PropertyList
17923. How large a loop detection stack to use
17924. gap (if empty string, NULL)
17925. * Defines for JSON, especially duk.bi_json.c.
17926. How much stack to require on entry to object/array decode
17927. indexed by recursion_depth
17928. type bit mask: types which certainly produce 'undefined'
17929. escape any non-ASCII characters
17930. Encoding/decoding flags
17931. extended types: compatible encoding
17932. extended types: custom encoding
17933. Encoding state. Heap object references are all borrowed.
17934. valstack index of loop detection object
17935. avoid key quotes when key is an ASCII Identifier
17936. DUK_JSON_H_INCLUDED
17937. replacer function
17938. How much stack to require on entry to object/array encode
17939. output bufwriter
17940. '='
17941. Store lexer position, restoring if quantifier is invalid.
17942. * Set lexer input position and reinitialize lookup window.
17943. ')'
17944. '-' verbatim
17945. One digit octal escape, digit validated.
17946. ('
17947. DUK_USE_REGEXP_SUPPORT
17948. escaped NonEscapeCharacter
17949. The E5.1 specification does not seem to allow IdentifierPart characters * to be used as identity escapes. Unfortunately this includes '\$', which * cannot be escaped as '\$'; it needs to be escaped e.g. as '\u0024'. * Many other implementations (including V8 and Rhino, for instance) do * accept '\$' as a valid identity escape, which is quite pragmatic. * See: test-regexp-identity-escape-dollar.js.
17950. mark range 'direct', bypass canonicalization (see Wiki)
17951. **comment:** * Lexer for source files, ToNumber() string conversions, RegExp expressions, * and JSON. ** Provides a stream of Ecmascript tokens from an UTF-8/CESU-8 buffer. The * caller can also rewind the token stream into a certain position which is * needed by the compiler part for multi-pass scanning. Tokens are * represented as duk_token structures, and contain line number information. * Token types are identified with DUK_TOK_* defines. ** Characters are decoded into a fixed size lookup window consisting of * decoded Unicode code points, with window positions past the end of the * input filled with an invalid codepoint (-1). The tokenizer can thus * perform multiple character lookups efficiently and with few sanity * checks (such as access outside the end of the input), which keeps the * tokenization code small at the cost of performance. ** Character data in tokens, such as identifier names and string literals, * is encoded into CESU-8 format on-the-fly while parsing the token in * question. The string data is made reachable to garbage collection by * placing the token-related values in value stack entries allocated for * this purpose by the caller. The characters exist in Unicode code point * form only in the fixed size lookup window, which keeps character data * expansion (of especially ASCII data) low. * * Token parsing supports the full range of Unicode characters as described * in the E5 specification. Parsing has been optimized for ASCII characters * because ordinary Ecmascript code consists almost entirely of ASCII * characters. Matching of complex Unicode codepoint sets (such as in the * IdentifierStart and IdentifierPart productions) is optimized for size, * and is done using a linear scan of a bit-packed list of ranges. This is * very slow, but should never be entered unless the source code actually * contains Unicode characters. ** Ecmascript tokenization is partially context sensitive. First, * additional future reserved words are recognized in strict mode (see E5 * Section 7.6.1.2). Second, a forward slash character ('/) can be * recognized either as starting a RegExp literal or as a division operator, * depending on context. The caller must provide necessary context flags * when requesting a new token. ** Future work: * * * Make line number tracking optional, as it consumes space. * * * Add a feature flag for disabling UTF-8 decoding of input, as most * source code is ASCII. Because of Unicode escapes written in ASCII, * this does not allow Unicode support to be removed from e.g. * duk_unicode_is_identifier_start() nor does it allow removal of CESU-8 * encoding of e.g. string literals. * * * Add a feature flag for disabling Unicode compliance of e.g. identifier * names. This allows for a build more than a kilobyte smaller, because * Unicode ranges needed by duk_unicode_is_identifier_start() and * duk_unicode_is_identifier_part() can be dropped. String literals * should still be allowed to contain escaped Unicode, so this still does * not allow removal of CESU-8 encoding of e.g. string literals. * * * Character lookup tables for codepoints above BMP could be stripped. * * * Strictly speaking, E5 specification requires that source code consists * of 16-bit code units, and if not, must be conceptually converted to * that format first. The current lexer processes Unicode code points * and allows characters outside the BMP. These should be converted to * surrogate pairs while reading the source characters into the window, * not after tokens have been formed (as is done now). However, the fix * is not trivial because two characters are decoded from one codepoint. * * * Optimize for speed as well as size. Large if-else ladders are (at * least potentially) slow.
label: code-design

17952. * Parse a RegExp token. The grammar is described in E5 Section 15.10. * Terminal constructions (such as quantifiers) are parsed directly here. ** 0xffffffffU is used as a marker for "infinity" in quantifiers. Further, * DUK_MAX_RE_QUANT_DIGITS limits the maximum number of digits that * will be accepted for a quantifier.
17953. "/=" and not in regexp mode
17954. Note: if DecimalLiteral starts with a '0', it can only be * followed by a period or an exponent indicator which starts * with 'e' or 'E'. Hence the if-check above ensures that * OctalIntegerLiteral is the only valid NumericLiteral * alternative at this point (even if y is, say, '9').
17955. '('
17956. * Intern the temporary byte buffer into a valstack slot * (in practice, slot1 or slot2).
17957. Fast path.
17958. '/'
17959. LF line terminator; CR LF and Unicode lineterms are handled in slow path
17960. line tracking
17961. * Interned identifier is compared against reserved words, which are * currently interned into the heap context. See genbuiltins.py. ** Note that an escape in the identifier disables recognition of * keywords; e.g. "\u0069f = 1;" is a valid statement (assigns to * identifier named "if"). This is not necessarily compliant, * see test-dec-escaped-char-in-keyword.js. * * Note: "get" and "set" are awkward. They are not officially * ReservedWords (and indeed e.g. "var set = 1;" is valid), and * must come out as DUK_TOK_IDENTIFIER. The compiler needs to * work around this a bit.
17962. val1 = count
17963. '{'
17964. 0xxx xxxx -> fast path

17965. radix
17966. having this as a separate function provided a size benefit
17967. * E5 Section 7.4. If the multi-line comment contains a newline, * it is treated like a single line terminator for automatic * semicolon insertion.
17968. second, parse flags
17969. * Since character data is only generated by decoding the source or by * the compiler itself, we rely on the input codepoints being correct * and avoid a check here.
* * Character data can also come here through decoding of Unicode * escapes ("udead\ubef") so all 16-bit unsigned values can be * present, even when the source file itself is strict UTF-8.
17970. **comment:** * (Re)initialize the temporary byte buffer. May be called extra times * with little impact.
label: code-design
17971. **comment:** XXX: because of the final check below (that the literal is not * followed by a digit), this could maybe be simplified, if we bail * out early from a leading zero (and if there are no periods etc). * Maybe too complex.
label: code-design
17972. dash is already 0
17973. * Various defines and file specific helper macros
17974. multi-character sets not allowed as part of ranges, see * E5 Section 15.10.2.15, abstract operation CharacterRange.
17975. '?'
17976. re-lookup first char on first loop
17977. '\"'
17978. state == 3
17979. * Matching order: * * Punctuator first chars, also covers comments, regexps * LineTerminator * Identifier or reserved word, also covers null/true/false literals * NumericLiteral * StringLiteral * EOF * * The order does not matter as long as the longest match is * always correctly identified. There are order dependencies * in the clauses, so it's not trivial to convert to a switch.
17980. ''
17981. Automatic semicolon insertion is allowed if a token is preceded * by line terminator(s), or terminates a statement list (right curly * or EOF).
17982. no point in supporting encodings of 5 or more bytes
17983. line continuation
17984. Use temporaries and update lex_ctx only when finished.
17985. '|'
17986. check that byte has the form 10xx xxxx
17987. lookup for 0x00a above assumes shortest encoding now
17988. unsigned
17989. form: { DecimalDigits , }, val1 = min count
17990. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.
17991. note: long live range
17992. 1110 xxxx 10xx xxxx 10xx xxxx
17993. 110x xxxx 10xx xxxx
17994. 0=before period/exp, * 1=after period, before exp * 2=after exp, allow '+' or '-' * 3=after exp and exp sign
17995. re-lookup curr char on first round
17996. NB: duk_lexer_getpoint() is a macro only
17997. lexer character window helpers
17998. eat closing quote
17999. avoid multiply
18000. Not enough data to provide a full window, so "scroll" window to * start of buffer and fill up the rest.
18001. * Parse Ecmascript source InputElementDiv or InputElementRegExp * (E5 Section 7), skipping whitespace, comments, and line terminators. * * Possible results are: * (1) a token * (2) a line terminator (skipped) * (3) a comment (skipped) * (4) EOF * * White space is automatically skipped from the current position (but * not after the input element). If input has already ended, returns * DUK_TOK_EOF indefinitely. If a parse error occurs, uses an DUK_ERROR() * macro call (and hence a longjmp through current heap longjmp context). * Comments and line terminator tokens are automatically skipped. * * The input element being matched is determined by regexp_mode; if set, * parses a InputElementRegExp, otherwise a InputElementDiv. The * difference between these are handling of productions starting with a * forward slash. * * If strict_mode is set, recognizes additional future reserved words * specific to strict mode, and refuses to parse octal literals. * * The matching strategy below is to (currently) use a six character * lookup window to quickly determine which production is the -longest- * matching one, and then parse that. The top-level if-else clauses * match the first character, and the code blocks for each clause * handle -all- alternatives for that first character. Ecmascript * specification uses the "longest match wins" semantics, so the order * of the if-clauses matters. * * Misc notes: * * * Ecmascript numeric literals do not accept a sign character. * Consequently e.g. "-1.0" is parsed as two tokens: a negative * sign and a positive numeric literal. The compiler performs * the negation during compilation, so this has no adverse impact. * * * There is no token for "undefined": it is just a value available * from the global object (or simply established by doing a reference * to an undefined value). * * * Some contexts want Identifier tokens, which are IdentifierNames * excluding reserved words, while some contexts want IdentifierNames * directly. In the latter case e.g. "while" is interpreted as an * identifier name, not a DUK_TOK WHILE token. The solution here is * to provide both token types: DUK_TOK WHILE goes to 't' while * DUK_TOK_IDENTIFIER goes to '_nores', and 'slot1' always contains * the identifier / keyword name. * * * Directive prologue needs to identify string literals such as * "use strict" and 'use strict', which are sensitive to line * continuations and escape sequences. For instance, "use\u0020strict" * is a valid directive but is distinct from "use strict". The solution * here is to decode escapes while tokenizing, but to keep track of the * number of escapes. Directive detection can then check that the * number of escapes is zero. * * * Multi-line comments with one or more internal LineTerminator are * treated like a line terminator to comply with automatic semicolon * insertion.
18002. got lineterm preceding non-whitespace, non-lineterm token
18003. **comment:** XXX: this duplicates functionality in duk_regexp.c where a similar loop is * required anyway. We could use that BUT we need to update the regexp compiler * 'nranges' too. Work this out a bit more cleanly to save space.
label: code-design
18004. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
18005. Call with count == DUK_LEXER_WINDOW_SIZE to fill buffer initially.
18006. **comment:** XXX: shared parsing?
label: code-design
18007. CR LF again a special case
18008. **comment:** XXX: logic for handling character ranges is now incorrect, it will accept * e.g. [\d-z] whereas it should croak from it? SMJS accepts this too, though. * * Needs a read through and a lot of additional tests.
label: code-design
18009. * E5 Section 7.3: CR LF is detected as a single line terminator for * line numbers. Here we also detect it as a single line terminator * token.
18010. Track number of escapes: necessary for proper keyword * detection.
18011. (?=
18012. re-read to avoid spill / fetch
18013. Note: decimal number may start with a period, but must be followed by a digit
18014. (?:
18015. ')'
18016. **comment:** initialized to avoid warnings of unused var
label: code-design
18017. Three digit octal escape, digits validated.
18018. validation performed by duk_hexval
18019. validation of the regexp is caller's responsibility
18020. character represents itself
18021. '~'

18022. '['
18023. IdentifierStart is stricter than IdentifierPart, so if the first '*' character is escaped, must have a stricter check here.
18024. '%'
18025. **comment:** not strictly necessary, but avoids "uninitialized variable" warnings
label: code-design
18026. * To avoid creating a heavy intermediate value for the list of ranges, * only the start token ('[' or '[^') is parsed here. The regexp * compiler parses the ranges itself.
18027. Note: first character is checked against this. But because * IdentifierPart includes all IdentifierStart characters, and * the first character (if unescaped) has already been checked * in the if condition, this is OK.
18028. **comment:** init is unnecessary but suppresses "may be used uninitialized" warnings
label: code-design
18029. **comment:** temporary, must be signed and 32-bit to hold Unicode code points
label: code-design
18030. eat closing slash
18031. not strictly necessary because of lookahead '}' above
18032. DUK__L0 -> '\ char * DUK__L1 ... DUK__L5 -> more lookup
18033. '+'
18034. default: char escape (two chars)
18035. * Parse an identifier and then check whether it is: * - reserved word (keyword or other reserved word) * - "null" (NullLiteral) * - "true" (BooleanLiteral) * - "false" (BooleanLiteral) * - anything else => identifier * * This does not follow the E5 productions cleanly, but is * useful and compact. * * Note that identifiers may contain Unicode escapes, * see E5 Sections 6 and 7.6. They must be decoded first, * and the result checked against allowed characters. * The above if-clause accepts an identifier start and an * '\' character -- no other token can begin with a '\'. * * Note that "get" and "set" are not reserved words in E5 * specification so they are recognized as plain identifiers * (the tokens DUK_TOK_GET and DUK_TOK_SET are actually not * used now). The compiler needs to work around this. * * Strictly speaking, following EcmaScript longest match * specification, an invalid escape for the first character * should cause a syntax error. However, an invalid escape * for IdentifierParts should just terminate the identifier * early (longest match), and let the next tokenization * fail. For instance Rhino croaks with 'foo\z' when * parsing the identifier. This has little practical impact.
18036. adv = 2 default OK
18037. **comment:** XXX: the handling of character range detection is a bit convoluted. * Try to simplify and make smaller.
label: code-design
18038. * Special parser for character classes; calls callback for every * range parsed and returns the number of ranges present.
18039. fill window
18040. Failed to match the quantifier, restore lexer and parse * opening brace as a literal.
18041. lookup should prevent this
18042. '<'
18043. 10xx xxxx -> invalid
18044. * "/" followed by something in regexp mode. See E5 Section 7.8.5. * * RegExp parsing is a bit complex. First, the regexp body is delimited * by forward slashes, but the body may also contain forward slashes as * part of an escape sequence or inside a character class (delimited by * square brackets). A mini state machine is used to implement these. * * Further, an early (parse time) error must be thrown if the regexp * would cause a run-time error when used in the expression new RegExp(...). * Parsing here simply extracts the (candidate) regexp, and also accepts * invalid regular expressions (which are delimited properly). The caller * (compiler) must perform final validation and regexp compilation. * * RegExp first char may not be '/' (single line comment) or '*' (multi- * line comment). These have already been checked above, so there is no * need below for special handling of the first regexp character as in * the E5 productions. * * About unicode escapes within regexp literals: * * E5 Section 7.8.5 grammar does NOT accept '\uHHHH escapes. * However, Section 6 states that regexps accept the escapes, * see paragraph starting with "In string literals...". * The regexp grammar, which sees the decoded regexp literal * (after lexical parsing) DOES have a '\uHHHH unicode escape. * So, for instance: * * '\u1234/* * should first be parsed by the lexical grammar as: * * '\' 'u' RegularExpressionBackslashSequence * '1' RegularExpressionNonTerminator * '2' RegularExpressionNonTerminator * '3' RegularExpressionNonTerminator * '4' RegularExpressionNonTerminator * * and the escape itself is then parsed by the regexp engine. * This is the current implementation. * * Minor spec inconsistency: * * E5 Section 7.8.5 RegularExpressionBackslashSequence is: * * '\ NonTerminator * * The latter is not normative and a typo. *
18045. * Octal escape or zero escape: * '\0 (lookahead not DecimalDigit) * '\1 ... '\7 (lookahead not DecimalDigit) * '\ZeroToThree OctalDigit (lookahead not DecimalDigit) * '\FourToSeven OctalDigit (no lookahead restrictions) * '\ZeroToThree OctalDigit OctalDigit (no lookahead restrictions) * * Zero escape is part of the standard syntax. Octal escapes are * defined in E5 Section B.1.2, and are only allowed in non-strict mode. * Any other productions starting with a decimal digit are invalid.
18046. out_token->lineterm set by caller
18047. (?)
18048. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.
18049. E5 Section 15.10.2.11
18050. Keep current size
18051. 0=base, 1=esc, 2=class, 3=class+esc
18052. **comment:** Throwing an error this deep makes the error rather vague, but * saves hundreds of bytes of code.
label: code-design
18053. XXX: potential issue with signed pointers, p_end < p.
18054. lookup shorthands (note: assume context variable is named 'lex_ctx')
18055. **comment:** Zero 'count' is also allowed to make call sites easier. * Arithmetic in bytes generates better code in GCC.
label: code-design
18056. numeric value of a hex digit (also covers octal and decimal digits)
18057. DecimalEscape, only '\0 is allowed, no leading zeroes are allowed
18058. XXX: optimize by adding the token numbers directly into the * always interned duk_hstring objects (there should be enough * flag bits free for that)?
18059. ';'
18060. '/' and not in regexp mode
18061. * Lexing helpers
18062. EOF
18063. whether to use macros or helper function depends on call count
18064. **comment:** (advance << 8) + token_type, updated at function end, * init is unnecessary but suppresses "may be used uninitialized" warnings.
label: code-design
18065. first, parse regexp body roughly
18066. Production allows 'DecimalDigits', including leading zeroes
18067. Does not allow e.g. 2**31-1, but one more would allow overflows of u32.
18068. '*'
18069. **comment:** * DecimalLiteral, HexIntegerLiteral, OctalIntegerLiteral * "pre-parsing", followed by an actual, accurate parser step. * * Note: the leading sign character ('+' or '-') is -not- part of * the production in E5 grammar, and that the a DecimalLiteral * starting with a '0' must be followed by a non-digit. Leading * zeroes are syntax errors and must be checked for. * * XXX: the two step parsing process is quite awkward, it would * be more straightforward to allow numconv to parse the longest * valid prefix (it already does that, it only needs to indicate * where the input ended). However, the lexer decodes characters * using a lookup window, so this is not a trivial change.
label: code-design
18070. ']'
18071. **comment:** XXX: better coercion
label: code-design
18072. '!'
18073. check final character validity

18074. * The escapes are same as outside a character class, except that '\b' has a * different meaning, and '\B' and backreferences are prohibited (see E5 * Section 15.10.2.19). However, it's difficult to share code because we * handle e.g. "\n" very differently: here we generate a single character * range for it.

18075. * Init lexer context

18076. Here 'x' is a Unicode codepoint

18077. How much to advance before next loop; note that next loop * will advance by 1 anyway, so -1 from the total escape * length (e.g. len("\uXXXX") - 1 = 6 - 1). As a default, * 1 is good.

18078. '>'

18079. Note: '\b' in char class is different than outside (assertion), * '\B' is not allowed and is caught by the duk_unicode_is_identifier_part() * check below.

18080. PatternCharacter, all excluded characters are matched by cases above

18081. skip opening slash on first loop

18082. Reuse buffer as is unless buffer has grown large.

18083. **comment:** DUK__ADVANCECHARS(lex_ctx, 2) would be correct here, but it unnecessary
label: code-design

18084. Note: duk_uint8_t type yields larger code

18085. DUK_USE_LEXER_SLIDING_WINDOW

18086. '-' as a range indicator

18087. Two digit octal escape, digits validated. * * The if-condition is a bit tricky. We could catch e.g. *\039' in the three-digit escape and fail it there (by * validating the digits), but we want to avoid extra * additional validation code.

18088. not LS/PS

18089. ""

18090. Slow path.

18091. No other escape beginning with a digit in strict mode

18092. invalid codepoint encoding or codepoint

18093. IdentityEscape, with dollar added as a valid additional * non-standard escape (see test-regexp-identity-escape-dollar.js). * Careful not to match end-of-buffer (<0) here.

18094. '&'

18095. **comment:** packed advance/token number macro used by multiple functions
label: code-design

18096. fast paths for space and tab

18097. Although these could be parsed as PatternCharacters unambiguously (here), * E5 Section 15.10.1 grammar explicitly forbids these as PatternCharacters.

18098. clipped codepoint

18099. fall through to error

18100. * Shared exit path

18101. val2 = min count, val1 = max count

18102. ch is a literal character here or -1 if parsed entity was * an escape such as "\s".

18103. DUK__L0() cannot be a digit, because the loop doesn't terminate if it is

18104. line terminator will be handled on next round

18105. part of string

18106. **comment:** This would be nice, but parsing is faster without resetting the * value slots. The only side effect is that references to temporary * string values may linger until lexing is finished; they're then * freed normally.
label: code-design

18107. '!'
18108. '^'

18109. * Append a Unicode codepoint to the temporary byte buffer. Performs * CESU-8 surrogate pair encoding for codepoints above the BMP. * Existing surrogate pairs are allowed and also encoded into CESU-8.

18110. eat backslash on entry

18111. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.

18112. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.

18113. **comment:** Period followed by a digit can only start DecimalLiteral * (handled in slow path). We could jump straight into the * DecimalLiteral handling but should avoid goto to inside * a block.
label: code-design

18114. IdentityEscape

18115. could also just pop?

18116. ''

18117. adv = 2 - 1 default OK

18118. eat '{' on entry

18119. Zero 'count' is also allowed to make call sites easier.

18120. Note: 'e' and 'E' are also accepted here.

18121. Zero escape (also allowed in non-strict mode)

18122. **comment:** * Advance lookup window by N characters, filling in new characters as * necessary. After returning caller is guaranteed a character window of * at least DUK__LEXER_WINDOW_SIZE characters. * * The main function duk__advance_bytes() is called at least once per every * token so it has a major lexer/compiler performance impact. There are two * variants for the main duk__advance_bytes() algorithm: a sliding window * approach which is slightly faster at the cost of larger code footprint, * and a simple copying one. * * Decoding directly from the source string would be another lexing option. * But the lookup window based approach has the advantage of hiding the * source string and its encoding effectively which gives more flexibility * going forward to e.g. support chunked streaming of source from flash. * * Decodes UTF-8/CESU-8 leniently with support for code points from U+0000 to * U+10FFFF, causing an error if the input is unparseable. Leniency means: * * * Unicode code point validation is intentionally not performed, * except to check that the codepoint does not exceed 0x10ffff. * * * In particular, surrogate pairs are allowed and not combined, which * allows source files to represent all SourceCharacters with CESU-8. * Broken surrogate pairs are allowed, as Ecmascript does not mandate * their validation. * * * Allow non-shortest UTF-8 encodings. * * Leniency here causes few security concerns because all character data is * decoded into Unicode codepoints before lexer processing, and is then * re-encoded into CESU-8. The source can be parsed as strict UTF-8 with * a compiler option. However, Ecmascript source characters include -all- * 16-bit unsigned integer codepoints, so leniency seems to be appropriate. * * Note that codepoints above the BMP are not strictly SourceCharacters, * but the lexer still accepts them as such. Before ending up in a string * or an identifier name, codepoints above BMP are converted into surrogate * pairs and then CESU-8 encoded, resulting in 16-bit Unicode data as * expected by Ecmascript. * * An alternative approach to dealing with invalid or partial sequences * would be to skip them and replace them with e.g. the Unicode replacement * character U+FFFD. This has limited utility because a replacement character * will most likely cause a parse error, unless it occurs inside a string. * Further, Ecmascript source is typically pure ASCII. * * See: * * http://en.wikipedia.org/wiki=UTF-8 * http://en.wikipedia.org/wiki/CESU-8 * http://tools.ietf.org/html/rfc3629 * http://en.wikipedia.org/wiki=UTF-8#Invalid_byte_sequences * * Future work: * * * Reject other invalid Unicode sequences (see Wikipedia entry for examples) * in strict UTF-8 mode. * * * Size optimize. An attempt to use a 16-byte lookup table for the first * byte resulted in a code increase though. * * * Is checking against maximum 0x10ffff really useful? 4-byte encoding * imposes a certain limit anyway. * * * Support chunked streaming of source code. Can be implemented either * by streaming chunks of bytes or chunks of codepoints.
label: code-design

18123. * E5 Section 7.4, allow SourceCharacter (which is any 16-bit * code point).

18124. **comment:** not necessary to init, disabled for faster parsing
label: code-design

18125. Section 7.8.3 (note): NumericLiteral must be followed by something other than * IdentifierStart or DecimalDigit.

18126. unreachable

18127. **comment:** never reached, but avoids warnings of * potentially unused variables.

label: code-design

18128. switch

18129. Track number of escapes; count not really needed but directive * prologues need to detect whether there were any escapes or line * continuations or not.

18130. free some memory

18131. **comment:** XXX: naming is inconsistent: ATOM_END_GROUP ends an ASSERT_START_LOOKAHEAD**label:** code-design

18132. 'advtok' indicates how much to advance and which token id to assign * at the end. This shared functionality minimizes code size. All * code paths are required to set 'advtok' to some value, so no default * init value is used. Code paths calling DUK_ERROR() never return so * they don't need to set advtok.

18133. marker: copy t if not changed

18134. eat opening quote on first loop

18135. '

18136. Note: intentionally allow leading zeroes here, as the * actual parser will check for them.

18137. input offset tracking

18138. Constants for duk_lexer_ctx.buf.

18139. token type (with reserved word identification)

18140. valstack slot for 2nd token value

18141. Lexer context. Same context is used for Ecmascript and Regexp parsing.

18142. identifier names (E5 Section 7.6)

18143. token was preceded by a lineterm

18144. token type (with reserved words as DUK_TOK_IDENTIFIER)

18145. number of tokens parsed

18146. Regexp tokens

18147. Sanity checks for string and token defines

18148. A regexp token value.

18149. number of escapes and line continuations (for directive prologue)

18150. * Lexer defines.

18151. Lexer codepoint with additional info like offset/line number

18152. **comment:** "get" and "set" are tokens but NOT ReservedWords. They are currently * parsed and identifiers and these defines are actually now unused.**label:** code-design

18153. A token value. Can be memcpy()'d, but note that slot1/slot2 values are on the valstack. * Some fields (like num, str1, str2) are only valid for specific token types and may have * stale values otherwise.

18154. input linenum at input_offset (not window[0]), init to 1

18155. DUK_USE_REGEXP_SUPPORT

18156. returned after EOF (infinite amount)

18157. unicode code points, window[0] is always next, points to 'buffer'

18158. bufwriter for temp accumulation

18159. input offset for window leading edge (not window[0])

18160. Convert heap string index to a token (reserved words)

18161. token allows automatic semicolon insertion (eof or preceded by newline)

18162. named "arithmetic" because result is signed

18163. start byte offset of token in lexer input

18164. A structure for 'snapshotting' a point for rewinding

18165. temp accumulation buffer

18166. * Prototypes

18167. reserved words: additional future reserved words in strict mode

18168. valstack slot for 1st token value

18169. numeric value of token

18170. exclusive

18171. reserved words: keywords

18172. string 1 of token (borrowed, stored to ctx->slot1_idx)

18173. reserved words: future reserved words

18174. input byte length

18175. unicode code points, window[0] is always next

18176. "null", "true", and "false" are always reserved words. * Note that "get" and "set" are not!

18177. DUK_LEXER_H_INCLUDED

18178. valstack slot for temp buffer

18179. currently 6 characters of lookup are actually needed (duk_lexer.c)

18180. token type

18181. numeric value (character, count)

18182. punctuators (unlike the spec, also includes "/" and "=")

18183. inclusive

18184. maximum token count before error (sanity backstop)

18185. start line of token (first char)

18186. literals (E5 Section 7.8), except null, true, false, which are treated * like reserved words (above).

18187. input string (may be a user pointer)

18188. Sanity check

18189. * A token is interpreted as any possible production of InputElementDiv * and InputElementRegExp, see E5 Section 7 in its entirety. Note that * the E5 "Token" production does not cover all actual tokens of the * language (which is explicitly stated in the specification, Section 7.5). * Null and boolean literals are defined as part of both ReservedWord * (E5 Section 7.6.1) and Literal (E5 Section 7.8) productions. Here, * null and boolean values have literal tokens, and are not reserved * words. * * Decimal literal negative/positive sign is -not- part of DUK_TOK_NUMBER. * The number tokens always have a non-negative value. The unary minus * operator in "-1.0" is optimized during compilation to yield a single * negative constant. * * Token numbering is free except that reserved words are required to be * in a continuous range and in a particular order. See genstrings.py.

18190. string 2 of token (borrowed, stored to ctx->slot2_idx)

18191. thread; minimizes argument passing

18192. Pad significand with "virtual" zero digits so that Dragon4 will * have enough (apparent) precision to work with.

18193. need to normalize, may even cancel to 0

18194. Zero special case: fake requested number of zero digits; ensure * no sign bit is printed. Relative and absolute fixed format * require separate handling.

18195. This may happen even after the fast path check, if exponent is * not balanced (e.g. "0e1"). Remember to respect zero sign.

18196. * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * ieee value = 1.ffff... * 2^(e - 1023) (normal) * = 0.ffff... * 2^(-1022) (denormal) * * algorithm v = f * b^e

18197. Note: 'count' is currently not adjusted by rounding (i.e. the * digits are not "chopped off". That shouldn't matter because * the digit position (absolute or relative) is passed on to the * convert-and-push function.

18198. (>= e 0) AND (not (= f (expt b (- p 1)))) * * be <- (expt b e) == b^e * r <- (* f be 2) == 2 * f * b^e [if b==2 -> f * b^(e+1)] * s <- 2 * m+ <- be == b^e * m- <- be == b^e * k <- 0 * B <- B * low_ok <- round * high_ok <- round

18199. DUK_USE_64BIT_OPS

18200. zero handled by caller

18201. Don't check for Infinity unless the context allows it. * 'Infinity' is a valid integer literal in e.g. base-36: * * parseInt('Infinity', 36) * 1461559270678

18202. borrow literal Infinity from builtin string

18203. $d \leftarrow (\text{quotient } (* r B) s)$ (in range 0...B-1)
18204. exponential notation forced
18205. $mp \leftarrow b/(e+1)$
18206. input radix
18207. **comment:** * The string conversion here incorporates all the necessary Ecmascript * semantics without attempting to be generic. nc_ctx->digits contains * nc_ctx->count digits (≥ 1), with the topmost digit's 'position' * indicated by nc_ctx->k as follows: * * digits="123" count=3 k=0 --> 0.123 * digits="123" count=3 k=1 --> 1.23 * digits="123" count=3 k=5 --> 12300 * digits="123" count=3 k=-1 --> 0.0123 * * Note that the identifier names used for format selection are different * in Burger-Dybvig paper and Ecmascript specification (quite confusingly * so, because e.g. 'k' has a totally different meaning in each). See * documentation for discussion. * * Ecmascript doesn't specify any specific behavior for format selection * (e.g. when to use exponent notation) for non-base-10 numbers. * * The bigint space in the context is reused for string output, as there * is more than enough space for that (>1kB at the moment), and we avoid * allocating even more stack.
label: code-design
18208. 0 or -1
18209. **comment:** XXX: this could be optimized; there is only one call site now though
label: code-design
18210. $x \leftarrow y + z$
18211. exponent digit
18212. **comment:** XXX: this could be optimized
label: code-design
18213. * Convert and push final string.
18214. $mm \leftarrow b^e$
18215. e.g. 12.3 --> digits="123" k=2 --> 1.23e1
18216. **comment:** This is essentially the 'scale' algorithm, with recursion removed. * Note that 'k' is either correct immediately, or will move in one * direction in the loop. There's no need to do the low/high checks * on every round (like the Scheme algorithm does). * * The scheme algorithm finds 'k' and updates 's' simultaneously, * while the logical algorithm finds 'k' with 's' having its initial * value, after which 's' is updated separately (see the Burger-Dybvig * paper, Section 3.1, steps 2 and 3). * * The case where $m^+ == m^-$ (almost always) is optimized for, because * it reduces the bigint operations considerably and almost always * applies. The scale loop only needs to work with m^+ , so this works.
label: code-design
18217. **comment:** Currently about $7*152 = 1064$ bytes. The space for these * duk__bigints is used also as a temporary buffer for generating * the final string. This is a bit awkward; a union would be * more correct.
label: code-design
18218. $s \leftarrow 2$
18219. (quotient-remainder (* r B) s) using a dummy subtraction loop
18220. * Scan number and setup for Dragon4. * * The fast path case is detected during setup: an integer which * can be converted without rounding, no net exponent. The fast * path could be implemented as a separate scan, but may not really * be worth it: the multiplications for building 'f' are not * expensive when 'f' is small. * * The significand ('f') must contain enough bits of (apparent) * accuracy, so that Dragon4 will generate enough binary output digits. * For decimal numbers, this means generating a 20-digit significand, * which should yield enough practical accuracy to parse IEEE doubles. * In fact, the Ecmascript specification explicitly allows an * implementation to treat digits beyond 20 as zeroes (and even * to round the 20th digit upwards). For non-decimal numbers, the * appropriate number of digits has been precomputed for comparable * accuracy. * * Digit counts: * * [dig_lzero] * | * .+---[dig_prec]----. * || | *
0000123.45678901234567890e+123456 * | | | | * `---[dig_frac]-----`---+---| * [dig_whole] [dig_expt] * * dig_frac and dig_expt are -1 if not present * dig_lzero is only computed for whole number part * * Parsing state * * Parsing whole part dig_frac < 0 AND dig_expt < 0 * Parsing fraction part dig_frac >= 0 AND dig_expt < 0 * Parsing exponent part dig_expt >= 0 (dig_frac may be < 0 or >= 0) * * Note: in case we hit an implementation limit (like exponent range), * we should throw an error, NOT return NaN or Infinity. Even with * very large exponent (or significand) values the final result may be * finite, so NaN/Infinity would be incorrect.
18221. 12 to 21
18222. $r \leftarrow (2 * f) * b^{(e+1)}$
18223. 31 to 36
18224. This upper value has been experimentally determined; debug build will check * bigint size with assertions.
18225. **comment:** unused
label: code-design
18226. $x \leftarrow y * z$
18227. Leading zero is not counted towards precision digits; not * in the integer part, nor in the fraction part.
18228. $\text{expt} == -1023 \rightarrow \text{bitstart}=0$ (leading 1); * $\text{expt} == -1024 \rightarrow \text{bitstart}=-1$ (one left of leading 1), etc
18229. **comment:** $m^+ != m^-$ (very rarely)
label: code-design
18230. * Dragon4 slow path digit generation.
18231. free-form
18232. interpret e.g. '0x' and '0xg' as a NaN (= parse error)
18233. Maximum number of digits generated.
18234. * Exposed number-to-string API * * Input: [number] * Output: [string]
18235. http://en.wikipedia.org/wiki/Exponentiation_by_squaring
18236. empty ("") is allowed in some formats (e.g. Number()), as zero
18237. $r \leftarrow (\text{remainder } (* r B) s)$
18238. if 1, doing a fixed format output (not free format)
18239. $tc1 = \text{true}$, $tc2 = \text{false}$
18240. Leading zero.
18241. $m^+ \leftarrow (* m^+ B)$
18242. `toString()` conditions
18243. Digit generation
18244. significant from precision perspective
18245. Loop structure ensures that we don't compute $t1^2$ unnecessarily * on the final round, as that might create a bignum exceeding the * current DUK_BL_MAX_PARTS limit.
18246. Most common cases first.
18247. Ignore digits beyond a radix-specific limit, but note them * in expt_adj.
18248. Return value: $<0 \iff x < y \iff 0 \iff x == y \iff >0 \iff x > y$
18249. no net exponent
18250. needed for inf: causes mantissa to become zero, * and rounding to be skipped.
18251. carry
18252. add E
18253. A 32-bit unsigned integer formats to at most 32 digits (the * worst case happens with radix == 2). Output the digits backwards, * and use a memmove() to get them in the right place.
18254. Fast path the binary case
18255. Infinity
18256. low to high
18257. $x \leftarrow (1 << y)$
18258. absolute position for digit considered for rounding
18259. $r \leftarrow (* r B) * s \leftarrow s * m^+ \leftarrow (* m^+ B) * m^- \leftarrow (* m^- B) * k \leftarrow (-k 1)$
18260. $r \leftarrow (* f 2) * s \leftarrow (* (\text{expt } b (- e)) 2) == b^{(-e)} * 2$ [if $b == 2 \rightarrow b^{(1-e)}]$ * $m^+ \leftarrow 1 * m^- \leftarrow 1 * k \leftarrow 0 * B \leftarrow B * \text{low_ok} \leftarrow \text{round} * \text{high_ok} \leftarrow \text{round}$

18261. tc1 = false, tc2 = false
18262. r <- r * s <- (* s B) * m+ <- m+ * m- <- m- * k <- (+ k 1)
18263. + 1 for rounding
18264. t1 <- (* r B)
18265. Both inputs are zero; cases where only one is zero can go * through main algorithm.
18266. Exponent without a sign or with a +/- sign is accepted * by all call sites (even JSON.parse()).
18267. * Convert binary digits into an IEEE double. Need to handle * denormals and rounding correctly.
18268. allow e.g. '0x0009' and '00077'
18269. tc1 = false, tc2 = true
18270. add AC*2^32
18271. large context; around 2kB now
18272. mp <- b^e
18273. s <- b^(e-1) * 2
18274. Would be nice to bulk clear the allocation, but the context * is 1-2 kilobytes and nothing should rely on it being zeroed.
18275. x <- x + y, use t as temp
18276. setup many variables in nc_ctx
18277. add BC*2^16
18278. Note: if 'x' is zero, x->n becomes 0 here
18279. **comment:** * Dragon4 setup. * * Convert double from IEEE representation for conversion; * normal finite values have an implicit leading 1-bit. The * slow path algorithm doesn't handle zero, so zero is special * cased here but still creates a valid nc_ctx, and goes * through normal formatting in case special formatting has * been requested (e.g. forced exponential format: 0 -> "0e+0").
label: code-design
18280. * Multiply + add + carry for 32-bit components using only 16x16->32 * multiplies and carry detection based on unsigned overflow. * * 1st mult, 32-bit: (A*2^16 + B) * 2nd mult, 32-bit: (C*2^16 + D) * 3rd add, 32-bit: E * 4th add, 32-bit: F * * (AC*2^16 + B) * (C*2^16 + D) + E + F * = AC*2^32 + AD*2^16 + BC*2^16 + BD + E + F * = AC*2^32 + (AD + BC)*2^16 + (BD + E + F) * = AC*2^32 + AD*2^16 + BC*2^16 + (BD + E + F)
18281. (-Number.MAX_VALUE).toString(2).length == 1025, + spare
18282. position of highest digit changed
18283. * Round-up limit. * * For even values, divides evenly, e.g. 10 -> roundup_limit=5. * * For odd values, rounds up, e.g. 3 -> roundup_limit=2. * If radix is 3, 0/3 -> down, 1/3 -> down, 2/3 -> up.
18284. Note: computes with 'idx' in assertions, so caller beware. * 'idx' is preincremented, i.e. '1' on first call, because it * is more convenient for the caller.
18285. * Digit generation loop. * * Different termination conditions: * * 1. Free format output. Terminate when shortest accurate * representation found. * * 2. Fixed format output, with specific number of digits. * Ignore termination conditions, terminate when digits * generated. Caller requests an extra digit and rounds. * * 3. Fixed format output, with a specific absolute cut-off * position (e.g. 10 digits after decimal point). Note * that we always generate at least one digit, even if * the digit is below the cut-off point already.
18286. number of digits changed
18287. **comment:** No NUL term checks in this debug code.
label: code-design
18288. triggers garbage digit check below
18289. digit count
18290. impose a reasonable exponent limit, so that exp * doesn't need to get tracked using a bigint.
18291. interpret e.g. '09' as '0', not NaN
18292.toFixed()
18293. * Tables generated with src/gennumdigits.py. * * duk_str2num_digits_for_radix indicates, for each radix, how many input * digits should be considered significant for string-to-number conversion. * The input is also padded to this many digits to give the Dragon4 * conversion enough (apparent) precision to work with. * * duk_str2num_exp_limits indicates, for each radix, the radix-specific * minimum/maximum exponent values (for a Dragon4 integer mantissa) * below and above which the number is guaranteed to underflow to zero * or overflow to Infinity. This allows parsing to keep bigint values * bounded.
18294. Detect zero special case.
18295. Number and (minimum) size of bigints in the nc_ctx structure.
18296. Careful with borrow condition: * - If borrow not subtracted: 0x12345678 - 0 - 0xffffffff = 0x12345679 (> 0x12345678) * - If borrow subtracted: 0x12345678 - 1 - 0xffffffff = 0x12345678 (== 0x12345678)
18297. **comment:** Note: not honoring round-to-even should work but now generates incorrect * results. For instance, 1e23 serializes to "a000...", i.e. the first digit * equals the radix (10). Scaling stops one step too early in this case. * Don't know why this is the case, but since this code path is unused, it * doesn't matter.
label: code-design
18298. r <- (2 * f) * b^e
18299. -Infinity
18300. * Exponent notation for non-base-10 numbers isn't specified in EcmaScript * specification, as it never explicitly turns up: non-decimal numbers can * only be formatted with Number.prototype.toString([radix]) and for that, * behavior is not explicitly specified. * * Logical choices include formatting the exponent as decimal (e.g. binary * 100000 as 1e+5) or in current radix (e.g. binary 100000 as 1e+101). * The Dragon4 algorithm (in the original paper) prints the exponent value * in the target radix B. However, for radix values 15 and above, the * exponent separator 'e' is no longer easily parseable. Consider, for * instance, the number ".faecee+1c".
18301. Start position (inclusive) and end position (exclusive)
18302. * Preliminaries: trim, sign, Infinity check * * We rely on the interned string having a NUL terminator, which will * cause a parse failure wherever it is encountered. As a result, we * don't need separate pointer checks. * * There is no special parsing for 'NaN' in the specification although * 'Infinity' (with an optional sign) is allowed in some contexts. * Some contexts allow plus/minus sign, while others only allow the * minus sign (like JSON.parse()). * * Automatic hex number detection (leading '0x' or '0X') and octal * number detection (leading '0' followed by at least one octal digit) * is done here too.
18303. denormal
18304. max possible
18305. fixed-format
18306. Count is incremented before DUK_DRAGON4_OUTPUT_PREINC() call * on purpose, which is taken into account by the macro.
18307. Sometimes this assert is not true right now; it will be true after * rounding. See: test-bug-numconv-mantissa-assert.js.
18308. IEEE exp without bias
18309. Exponent handling: if exponent format is used, record exponent value and * fake k such that one leading digit is generated (e.g. digits=123 -> "1.23"). * * toFixed() prevents exponent use; otherwise apply a set of criteria to * match the other API calls (toString(), toPrecision, etc).
18310. low 32 bits is complete
18311. Validity checks for various fraction formats ("0.1", ".1", "1.", ".").
18312. generate mantissa with a single leading whole number digit
18313. requested number of output digits; 0 = free-format
18314. s <- 2 * b
18315. (>= e 0) AND (= f (expt b (- p 1))) * * be <- (expt b e) == b^e * be1 <- (* be b) == (expt b (+ e 1)) == b^(e+1) * r <- (* f be1 2) == 2 * f * b^(e+1) [if b==2 -> f * b^(e+2)] * s <- (* b 2) [if b==2 -> 4] * m+ <- be1 == b^(e+1) * m- <- be == b^e * k <- 0 * B <- B * low_ok <- round * high_ok <- round
18316. Perform fixed-format rounding.
18317. Overestimate required size; debug code so not critical to be tight.
18318. see algorithm
18319. t1 <- (+ r m+)
18320. denormal or zero
18321. **comment:** no special formatting
label: code-design
18322. Fast path check.

18323. caller handles sign change
18324. r <- (* f b 2) [if b==2 -> (* f 4)] * s <- (* (expt b (- 1 e)) 2) == b^(1-e) * 2 [if b==2 -> b^(2-e)] * m+ <- b == 2 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round
18325. 22 to 31
18326. **comment:** * Handle integers in 32-bit range (that is, $[-(2^{32}-1), 2^{32}-1]$) * specially, as they're very likely for embedded programs. This * is now done for all radix values. We must be careful not to use * the fast path when special formatting (e.g. forced exponential) * is in force. * * XXX: could save space by supporting radix 10 only and using * sprintf "%lu" for the fast path and for exponent formatting.
label: code-design
18327. x <- x - y, use t as temp
18328. Carry is detected based on wrapping which relies on exact 32-bit * types.
18329. current digit
18330. fixed precision and zero padding would be required
18331. t has high mantissa
18332. max size: radix=2 + sign
18333. We can't shortcut zero here if it goes through special formatting * (such as forced exponential notation).
18334. "." is not accepted in any format
18335. Maximum number of characters in formatted value.
18336. Terminating conditions. For fixed width output, we just ignore the * terminating conditions (and pretend that tc1 == tc2 == false). The * the current shortcut for fixed-format output is to generate a few * extra digits and use rounding (with carry) to finish the output.
18337. **comment:** Borrow is detected based on wrapping which relies on exact 32-bit * types.
label: code-design
18338. fixed-format termination conditions
18339. * Limited functionality bigint implementation. * * Restricted to non-negative numbers with less than 32 * DUK__BI_MAX_PARTS bits, * with the caller responsible for ensuring this is never exceeded. No memory * allocation (except stack) is needed for bigint computation. Operations * have been tailored for number conversion needs. * * Argument order is "assignment order", i.e. target first, then arguments: * x <- y * z --> duk__bi_mul(x, y, z);
18340. **comment:** * Handle special cases (NaN, infinity, zero).
label: code-design
18341. exponent for 'f'
18342. 32-bit value
18343. Buffer used for generated digits, values are in the range [0,B-1].
18344. absolute req_digits; e.g. digits = 1 -> last digit is 0, * but add an extra digit for rounding.
18345. fall through and continue for-loop
18346. * Create mantissa
18347. 0.000...
18348. r <- r (updated above: r <- (remainder (* r B) s) * s <- s * m+ <- m+ (updated above: m+ <- (* m+ B) * m- <- m- (updated above: m- <- (* m- B) * B, low_ok, high_ok are fixed
18349. skip leading digit
18350. count is already incremented, take into account
18351. t2 = (* (+ r m+) B)
18352. exponent non-negative (and thus not minimum exponent)
18353. true, because v[1] has at least one bit set
18354. **comment:** This seems to waste a lot of stack frame entries, but good compilers * will compute these as needed below. Some of these initial flags are * also modified in the code below, so they can't all be removed.
label: code-design
18355. s <- b^(1-e) * 2
18356. * Number-to-string and string-to-number conversions. * * Slow path number-to-string and string-to-number conversion is based on * a Dragon4 variant, with fast paths for small integers. Big integer * arithmetic is needed for guaranteeing that the conversion is correct * and uses a minimum number of digits. The big number arithmetic has a * fixed maximum size and does not require dynamic allocations. * * See: doc/number-conversion.rst.
18357. is valid size
18358. integer number in range
18359. no need to normalize
18360. Current size is about 152 bytes.
18361. build result as: (r << 32) + s: start with (BD + E + F)
18362. x <- b'y; use t1 and t2 as temps
18363. k > 0 -> k was too low, and cannot be too high
18364. * Exposed string-to-number API * * Input: [string] * Output: [number] * * If number parsing fails, a NaN is pushed as the result. If number parsing * fails due to an internal error, an InternalError is thrown.
18365. x <- y - z, require x >= y => z >= 0, i.e. y >= z
18366. m- <- (* m- B)
18367. Exponent
18368. x <- y
18369. terminating conditions
18370. t1 = (+ r m+)
18371. if 1, doing a string-to-number; else doing a number-to-string
18372. (e.g. k=3, digits=2 -> "12X")
18373. For string-to-number, pretend we never have the lowest mantissa as there * is no natural "precision" for inputs. Having lowest_mantissa == 0, we'll * fall into the base cases for both e >= 0 and e < 0.
18374. whole or fraction digit
18375. * A Dragon4 number-to-string variant, based on: * * Guy L. Steele Jr., Jon L. White: "How to Print Floating-Point Numbers * Accurately" * * Robert G. Burger, R. Kent Dybvig: "Printing Floating-Point Numbers * Quickly and Accurately" * * The current algorithm is based on Figure 1 of the Burger-Dybvig paper, * i.e. the base implementation without logarithm estimation speedups * (these would increase code footprint considerably). Fixed-format output * does not follow the suggestions in the paper; instead, we generate an * extra digit and round-with-carry. * * The same algorithm is used for number parsing (with b=10 and B=2) * by generating one extra digit and doing rounding manually. * * See doc/number-conversion.rst for limitations.
18376. essentially tracks digit position of lowest 'f' digit
18377. Some contexts don't allow fractions at all; this can't be a * post-check because the state ('f' and expt) would be incorrect.
18378. r <- 2 * f
18379. lead zero + 'digits' fractions + 1 for rounding
18380. A leading digit is not required in some cases, e.g. accept ".123". * In other cases (JSON.parse()) a leading digit is required. This * is checked for after the loop.
18381. Note: we don't parse back exponent notation for anything else * than radix 10, so this is not an ambiguous check (e.g. hex * exponent values may have 'e' either as a significand digit * or as an exponent separator). * * If the exponent separator occurs twice, 'e' will be interpreted * as a digit (= 14) and will be rejected as an invalid decimal * digit.
18382. "123.456"
18383. for
18384. Bigint is 2^{52} . Used to detect normalized IEEE double mantissa values * which are at the lowest edge (next floating point value downwards has * a different exponent). The lowest mantissa has the form: * * 1000.....000 (52 zeroes; only "hidden bit" is set)
18385. distinct bignums, easy mistake to make
18386. -> sets 'f' and 'e'
18387. **comment:** Leading / trailing whitespace is sometimes accepted and * sometimes not. After white space trimming, all valid input * characters are pure ASCII.

label: code-design

18388. NB: use 's' as temp on purpose

18389. **comment:** XXX: could accept numbers larger than 32 bits, e.g. up to 53 bits?**label:** code-design

18390. Significand ('f') padding.

18391. **comment:** XXX: join these ops (multiply-accumulate), but only if * code footprint decreases.**label:** code-design

18392. Exponent without digits (e.g. "1e" or "1e+"). If trailing garbage is * allowed, ignore exponent part as garbage (= parse as "1", i.e. exp 0).

18393. don't increase 'count'

18394. this is the case for normalized numbers

18395. arbitrary marker, outside valid exp range

18396. **comment:** XXX: union would be more correct**label:** code-design

18397. NaN sign bit is platform specific with unpacked, un-normalized NaNs

18398. * Conversion helpers

18399. add AD*2^16

18400. Fast path is triggered for no exponent and also for balanced exponent * and fraction parts, e.g. for "1.23e2" == "123". Remember to respect * zero sign.

18401. normal

18402. use bigint area as a temp

18403. ".123"

18404. is normalized

18405. lowest mantissa for this exponent

18406. mm <- mp

18407. Round up digits to a given position. If position is out-of-bounds, * does nothing. If carry propagates over the first digit, a '1' is * prepended to digits and 'k' will be updated. Return value indicates * whether carry propagated over the first digit. ** Note that nc_ctx->count is NOT updated based on the rounding position * (it is updated only if carry overflows over the first digit and an * extra digit is prepended).

18408. * Figure out how generated digits match up with the mantissa, * and then perform rounding. If mantissa overflows, need to * recompute the exponent (it is bumped and may overflow to * infinity). ** For normal numbers the leading '1' is hidden and ignored, * and the last bit is used for rounding; ** rounding pt * <-----52----->| * 1 x x x x ... x x x|x/y ==> x x x x ... x x x x * * For denormals, the leading '1' is included in the number, * and the rounding point is different: ** rounding pt * <-52 or less--->| * 1 x x x x ... x x|x x y ==> 0 0 ... 1 x x ... x x * * The largest denormals will have a mantissa beginning with * a '1' (the explicit leading bit); smaller denormals will * have leading zero bits. ** If the exponent would become too high, the result becomes * Infinity. If the exponent is so small that the entire * mantissa becomes zero, the result becomes zero. ** Note: the Dragon4 'k' is off-by-one with respect to the IEEE * exponent. For instance, k==0 indicates that the leading '1' * digit is at the first binary fraction position (0.1xxx...); * the corresponding IEEE exponent would be -1.

18409. ".123"

18410. no negative sign for zero

18411. (Number.MAX_VALUE).toString(2).length == 1024, + spare

18412. normal: implicit leading 1-bit

18413. **comment:** Should not be required because the code below always sets both high * and low parts, but at least gcc-4.4.5 fails to deduce this correctly * (perhaps because the low part is set (seemingly) conditionally in a * loop), so this is here to avoid the bogus warning.**label:** code-design

18414. 'roundpos' is relative to nc_ctx->k and increases to the right * (opposite of how 'k' changes).

18415. r <- (2 * b) * f

18416. digit position is absolute, not relative

18417. x <- x * y, use t as temp

18418. t1 <- t1 - s

18419. Assume IEEE round-to-even, so that shorter encoding can be used * when round-to-even would produce correct result. By removing * this check (and having low_ok == high_ok == 0) the results would * still be accurate but in some cases longer than necessary.

18420. When doing string-to-number, lowest_mantissa is always 0 so * the exponent check, while incorrect, won't matter.

18421. (< (* r 2) s)

18422. not minimum exponent

18423. add F

18424. output radix

18425. **comment:** Quick reject of too large or too small exponents. This check * would be incorrect for zero (e.g. "0e1000" is zero, not Infinity) * so zero check must be above.**label:** code-design

18426. * Dragon4 slow path (binary) digit generation. * An extra digit is generated for rounding.

18427. biased exp == 0 -> denormal, exp -1022

18428. 37x32 = 1184 bits

18429. **comment:** slow init, do only for slow path cases**label:** code-design

18430. ".123." is allowed in some formats

18431. **comment:** Corner case: see test-numconv-parse-mant-carry.js. We could * just bump the exponent and update bitstart, but it's more robust * to recompute (but avoid rounding twice).**label:** code-design

18432. tc1 = true, tc2 = true

18433. * round_idx points to the digit which is considered for rounding; the * digit to its left is the final digit of the rounded value. If round_idx * is zero, rounding will be performed; the result will either be an empty * rounded value or if carry happens a '1' digit is generated.

18434. 2 to 11

18435. x <- y - z

18436. **comment:** XXX: this algorithm could be optimized quite a lot by using e.g. * a logarithm based estimator for 'k' and performing B^n multiplication * using a lookup table or using some bit-representation based exp * algorithm. Currently we just loop, with significant performance * impact for very large and very small numbers.**label:** code-design

18437. Careful with carry condition: * - If carry not added: 0x12345678 + 0 + 0xffffffff = 0x12345677 (< 0x12345678) * - If carry added: 0x12345678 + 1 + 0xffffffff = 0x12345678 (= 0x12345678)

18438. Force exponential format. Used for toExponential().

18439. Allow naked fraction (e.g. ".123")

18440. Digit count indicates number of fractions (i.e. an absolute * digit index instead of a relative one). Used together with * DUK_N2S_FLAG_FIXED_FORMAT for toFixed().

18441. Allow leading zeroes (e.g. "0123" -> "123")

18442. Maximum exponent value when parsing numbers. This is not strictly * compliant as there should be no upper limit, but as we parse the * exponent without a bigint, impose some limit.

18443. * String-to-number conversion

18444. Output a specified number of digits instead of using the shortest * form. Used for toPrecision() and toFixed().

18445. Allow trailing garbage (e.g. treat "123foo" as "123")

18446. * Prototypes

18447. Allow exponent

18448. DUK_NUMCONV_H_INCLUDED

18449. Allow empty string to be interpreted as 0
18450. **comment:** If number would need zero padding (for whole number part), use * exponential format instead. E.g. if input number is 12300, 3 * digits are generated ("123"), output "1.23e+4" instead of "12300". * Used for toPrecision().
label: code-design
18451. Allow 'Infinity'
18452. * Number-to-string conversion. The semantics of these is very tightly * bound with the Ecmascript semantics required for call sites.
18453. Allow automatic detection of hex base ("0x" or "0X" prefix), * overrides radix argument and forces integer mode.
18454. Allow leading plus sign
18455. Allow leading minus sign
18456. Allow fraction part
18457. Trim white space (= allow leading and trailing whitespace)
18458. Allow empty fraction (e.g. "123.")
18459. Allow automatic detection of octal base, overrides radix * argument and forces integer mode.
18460. regexp opcodes
18461. maximum bytecode copies for {n,m} quantifiers
18462. * Regular expression structs, constants, and bytecode defines.
18463. 1e8
18464. regexp execution limits
18465. flags
18466. DUK_REGEX_H_INCLUDED
18467. internal temporary value, used for char classes
18468. 1e9
18469. * Prototypes
18470. regexp compilation limits
18471. **comment:** hacky helper for String.prototype.split()
label: code-design
18472. allocated from valstack (fixed buffer)
18473. highest capture number emitted so far (used as: ++captures)
18474. insert code for matching the remainder - infinite or finite
18475. expensive init, just want to fill window
18476. [... regexp_object escaped_source]
18477. record previous atom info in case next token is a quantifier
18478. DUK_USE_REGEX_SUPPORT
18479. retval (sub-atom char length) unused, tainted as complex above
18480. return '(?:)'
18481. two encoding attempts suffices
18482. * The remaining matches are emitted as sequence of SPLITs and atom * copies; the SPLITs skip the remaining copies and match the sequel. * This sequence needs to be emitted starting from the last copy * because the SPLITs are variable length due to the variable length * skip offset. This causes a lot of memory copying now. * * Example structure (greedy, match maximum # atoms): ** SPLIT1 LSEQ * (atom) * SPLIT1 LSEQ ; <- the byte length of this instruction is needed * (atom) ; to encode the above SPLIT1 correctly * ... * LSEQ:
18483. -> [... escaped_source bytecode]
18484. prefer direct execution
18485. flags always encodes to 1 byte
18486. **comment:** These are not needed to implement quantifier capture handling, * but might be needed at some point.
label: requirement
18487. * duk_re_range_callback for generating character class ranges. * * When ignoreCase is false, the range is simply emitted as is. * We don't, for instance, eliminate duplicates or overlapping * ranges in a character class. * * When ignoreCase is true, the range needs to be normalized through * canonicalization. Unfortunately a canonicalized version of a * continuous range is not necessarily continuous (e.g. [x-{] is * continuous but [X-{] is not). The current algorithm creates the * canonicalized range(s) space efficiently at the cost of compile * time execution time (see doc/regexp.rst for discussion). * * Note that the ctx->ranges is a context-wide temporary value * (this is OK because there cannot be multiple character classes * being parsed simultaneously).
18488. * Complex atom * * The original code is used as a template, and removed at the end * (this differs from the handling of simple quantifiers). * * NOTE: there is no current solution for empty atoms in complex * quantifiers. This would need some sort of a 'progress' instruction. * * XXX: impose limit on maximum result size, i.e. atom_code_len * atom_copies?
18489. mark as complex
18490. **comment:** unused
label: code-design
18491. insert (DUK_REGOP_WIPERANGE, start, count) in reverse order so the order ends up right
18492. reuse last emitted atom for remaining 'infinite' quantifier
18493. [... pattern flags escaped_source]
18494. [... pattern flags escaped_source bytecode]
18495. qmin and qmax will be 0 or 1
18496. bytecode start offset of the atom parsed in this loop * (allows quantifiers to copy the atom bytecode)
18497. [... regexp_object escaped_source bytecode]
18498. * Exposed regexp compilation primitive. * * Sets up a regexp compilation context, and calls duk_parse_disjunction() to do the * actual parsing. Handles generation of the compiled regexp header and the * "boilerplate" capture of the matching substring (save 0 and 1). Also does some * global level regexp checks after recursive compilation has finished. * * An escaped version of the regexp source, suitable for use as a RegExp instance * 'source' property (see E5 Section 15.10.3), is also left on the stack. * * Input stack: [pattern flags] * Output stack: [bytecode escaped_source] (both as strings)
18499. * Insert a jump offset at 'offset' to complete an instruction * (the jump offset is always the last component of an instruction). * The 'skip' argument must be computed relative to 'offset', * -without- taking into account the skip field being inserted. * * ... A B C ins X Y Z ... (ins may be a JUMP, SPLIT1/SPLIT2, etc) * => ... A B C ins SKIP X Y Z * * Computing the final (adjusted) skip value, which is relative to the * first byte of the next instruction, is a bit tricky because of the * variable length UTF-8 encoding. See doc/regexp.rst for discussion.
18500. * Encoding helpers * * Some of the typing is bytecode based, e.g. slice sizes are unsigned 32-bit * even though the buffer operations will use duk_size_t.
18501. insert range count
18502. * Create escaped RegExp source (E5 Section 15.10.3). * * The current approach is to special case the empty RegExp * (" -> '(?:')") and otherwise replace unescaped '/' characters * with '\/' regardless of where they occur in the regexp. * * Note that normalization does not seem to be necessary for * RegExp literals (e.g. '/foo/') because to be acceptable as * a RegExp literal, the text between forward slashes must * already match the escaping requirements (e.g. must not contain * unescaped forward slashes or be empty). Escaping IS needed * for expressions like 'new RegExp("...","")' however. * Currently, we re-escape in either case. * * Also note that we process the source here in UTF-8 encoded * form. This is correct, because any non-ASCII characters are * passed through without change.
18503. prefer direct
18504. **comment:** * Check for invalid backreferences; note that it is NOT an error * to back-reference a capture group which has not yet been introduced * in the pattern (as in /\1(foo)/); in fact, the backreference will * always match! It IS an error to back-reference a capture group * which will never be introduced in the pattern. Thus, we can check * for such references only after parsing is complete.
label: code-design
18505. * Init compilation context
18506. [... escaped_source bytecode]
18507. -1 if disjunction is complex, char length if simple
18508. insert ranges instruction, range count patched in later
18509. add a new pending split to the beginning of the entire disjunction

18510. never here
18511. wipe the capture range made by the atom (if any)
18512. negative -> complex atom
18513. -1 for opcode
18514. no change
18515. 'aint' result as complex
18516. * Helper macros
18517. * Disjunction struct: result of parsing a disjunction
18518. [... escape_source bytecode]
18519. [... pattern flags]
18520. -> [... flags escaped_source bytecode]
18521. **comment:** note: ctx-wide temporary
 label: code-design
18522. * Create normalized 'source' property (E5 Section 15.10.3).
18523. * Compilation
18524. set by atom case clauses
18525. value of re_ctx->captures at start of atom
18526. remove the original 'template' atom
18527. * Flags parsing (see E5 Section 15.10.4.1).
18528. * Init lexer
18529. -> [... escaped_source]
18530. Special case: original qmin was zero so there is nothing * to repeat. Emit an atom copy but jump over it here.
18531. insert the required matches (qmin) by copying the atom
18532. expect_eof
18533. coerce to string
18534. * Range parsing is done with a special lexer function which calls * us for every range parsed. This is different from how rest of * the parsing works, but avoids a heavy, arbitrary size intermediate * value type to hold the ranges. * * Another complication is the handling of character ranges when * case insensitive matching is used (see docs for discussion). * The range handler callback given to the lexer takes care of this * as well. * * Note that duplicate ranges are not eliminated when parsing character * classes, so that canonicalization of * * [0-9a-fA-Fx-{}] * * creates the result (note the duplicate ranges): * * [0-9A-FA-FX-Z-{}] * * where [x-{}] is split as a result of canonicalization. The duplicate * ranges are not a semantics issue: they work correctly.
18535. +1 for opcode
18536. * Finalize stack
18537. always at least the header
18538. Unescaped '/' ANYWHERE in the regexp (in disjunction, * inside a character class, ...) => same escape works.
18539. * Emit compiled regexp header: flags, ncaptures * (insertion order inverted on purpose)
18540. Note: successive characters could be joined into string matches * but this is not trivial (consider e.g. '/xyz+/'; see docs for * more discussion).
18541. complete 'sub atom'
18542. * Create a RegExp instance (E5 Section 15.10.7). * * Note: the output stack left by duk_regexp_compile() is directly compatible * with the input here. * * Input stack: [escaped_source bytecode] (both as strings) * Output stack: [RegExp]
18543. XXX: the insert helpers should ensure that the bytecode result is not * larger than expected (or at least assert for it). Many things in the * bytecode, like skip offsets, won't work correctly if the bytecode is * larger than say 2G.
18544. char length of the atom parsed in this loop
18545. * Regexp compilation. * * See doc/regexp.rst for a discussion of the compilation approach and * current limitations. * * Regexp bytecode assumes jumps can be expressed with signed 32-bit * integers. Consequently the bytecode size must not exceed 0x7fffffffL. * The implementation casts duk_size_t (buffer size) to duk_(u)int32_t * in many places. Although this could be changed, the bytecode format * limit would still prevent regexps exceeding the signed 32-bit limit * from working. * * XXX: The implementation does not prevent bytecode from exceeding the * maximum supported size. This could be done by limiting the maximum * input string size (assuming an upper bound can be computed for number * of bytecode bytes emitted per input byte) or checking buffer maximum * size when emitting bytecode (slower).
18546. silence scan-build warning
18547. **comment:** * Canonicalize a range, generating result ranges as necessary. * Needs to exhaustively scan the entire range (at most 65536 * code points). If 'direct' is set, caller (lexer) has ensured * that the range is already canonicalization compatible (this * is used to avoid unnecessary canonicalization of built-in * ranges like \W, which are not affected by canonicalization). * * NOTE: here is one place where we don't want to support chars * outside the BMP, because the exhaustive search would be * massively larger.
 label: code-design
18548. only advance if not tainted
18549. **comment:** XXX: solve into closed form (smaller code)
 label: code-design
18550. re_ctx->captures at the start of the atom parsed in this loop
18551. special helper for emitting u16 lists (used for character ranges for built-in char classes)
18552. mark as complex (capture handling)
18553. [... regexp_object]
18554. negative -> no atom matched on previous round
18555. * Parse regexp Disjunction. Most of regexp compilation happens here. * * Handles Disjunction, Alternative, and Term productions directly without * recursion. The only constructs requiring recursion are positive/negative * lookaheads, capturing parentheses, and non-capturing parentheses. * * The function determines whether the entire disjunction is a 'simple atom' * (see doc/regexp.rst discussion on 'simple quantifiers') and if so, * returns the atom character length which is needed by the caller to keep * track of its own atom character length. A disjunction with more than one * alternative is never considered a simple atom (although in some cases * that might be the case). * * Return value: simple atom character length or < 0 if not a simple atom. * Appends the bytecode for the disjunction matcher to the end of the temp * buffer. * * Regexp top level structure is: * * Disjunction = Term* * | Term* | Disjunction * * Term = Assertion * | Atom * | Atom Quantifier * * An empty Term sequence is a valid disjunction alternative (e.g. /\|/). * * Notes: * * * Tracking of the 'simple-ness' of the current atom vs. the entire * disjunction are separate matters. For instance, the disjunction * may be complex, but individual atoms may be simple. Furthermore, * simple quantifiers are used whenever possible, even if the * disjunction as a whole is complex. * * * The estimate of whether an atom is simple is conservative now, * and it would be possible to expand it. For instance, captures * cause the disjunction to be marked complex, even though captures * -can- be handled by simple quantifiers with some minor modifications. * * * Disjunction 'tainting' as 'complex' is handled at the end of the * main for loop collectively for atoms. Assertions, quantifiers, * and !' tokens need to taint the result manually if necessary. * Assertions cannot add to result char length, only atoms (and * quantifiers) can; currently quantifiers will taint the result * as complex though.
18556. TypeError if fails
18557. a complex (new) atom taints the result
18558. finish up pending jump and split for last alternative
18559. duplicate zeroing, expect for (possible) NULL init
18560. patch in range count later
18561. patch pending jump and split
18562. * Args validation
18563. atom_char_length, atom_start_offset, atom_start_offset reflect the * atom matched on the previous loop. If a quantifier is encountered * on this loop, these are needed to handle the quantifier correctly. * new_atom_char_length etc are for the atom parsed on this round; * they're written to atom_char_length etc at the end of the round.
18564. regexp support disabled
18565. * Simple atom * * If atom_char_length is zero, we'll have unbounded execution time for e.g. * /(0*x/.exec('x'). We can't just skip the match because it might have some * side effects (for instance, if we allowed captures in simple atoms, the * capture needs to happen). The simple solution below is to force the * quantifier to

match at most once, since the additional matches have no effect. ** With a simple atom there can be no capture groups, so no captures need * to be reset.

18566. offset is now target of the pending split (right after jump)

18567. re_ctx->captures at start and end of atom parsing. * Since 'captures' indicates highest capture number emitted * so far in a DUK_REOP_SAVE, the captures numbers saved by * the atom are:]start_captures,end_captures].

18568. add a new pending match jump for latest finished alternative

18569. [... pattern flags escaped_source buffer]

18570. parse ranges until character class ends

18571. Call sites don't need the result length so it's not accumulated.

18572. XXX: return type should probably be duk_size_t, or explicit checks are needed for * maximum size.

18573. 'taint' result as complex -- this is conservative, * as lookaheads do not backtrack.

18574. prefer jump

18575. Number of characters that the atom matches (e.g. 3 for 'abc'), * -1 if atom is complex and number of matched characters either * varies or is not known.

18576. * The handling here is a bit tricky. If a previous ']' has been processed, * we have a pending split1 and a pending jump (for a previous match). These * need to be back-patched carefully. See docs for a detailed example.

18577. Note: can be safely scanned as bytes (undecoded)

18578. **comment:** pre-check how many atom copies we're willing to make (atom_copies not needed below)

label: requirement

18579. match fail

18580. * Helpers for dealing with the input string

18581. split2: prefer jump execution (not direct)

18582. DUK_USE_REGEXP_SUPPORT

18583. non-global regexp: lastIndex never updated on match

18584. split1: prefer direct execution (no jump)

18585. [... re_obj input bc]

18586. must fit into duk_small_int_t

18587. note: caller 'sp' is intentionally not updated here

18588. * Byte-based matching would be possible for case-sensitive * matching but not for case-insensitive matching. So, we * match by decoding the input and bytecode character normally. ** Bytecode characters are assumed to be already canonicalized. * Input characters are canonicalized automatically by * duk_inp_get_cp() if necessary. ** There is no opcode for matching multiple characters. The * regexp compiler has trouble joining strings efficiently * during compilation. See doc/regexp.rst for more discussion.

18589. must have start and end

18590. * No match, E5 Section 15.10.6.2, step 9.a.i - 9.a.ii apply, regardless * of 'global' flag of the RegExp. In particular, if lastIndex is invalid * initially, it is reset to zero.

18591. * Regexp recursive matching function. ** Returns 'sp' on successful match (points to character after last matched one), * NULL otherwise. ** The C recursion depth limit check is only performed in this function, this * suffices because the function is present in all true recursion required by * regexp execution.

18592. **comment:** not really necessary

label: code-design

18593. E5 Sections 15.10.2.8, 7.3

18594. match: keep wiped/resaved values

18595. no regexp instance should exist without a non-configurable bytecode property

18596. Note: not necessary to check p against re_ctx->input_end: * the memory access is checked by duk_inp_get_cp(), while * valid compiled regexps cannot write a saved[] entry * which points to outside the string.

18597. **comment:** integer, but may be +/- Infinite, +/- zero (not NaN, though)

label: code-design

18598. Note: if encoding ends by hitting end of input, we don't check that * the encoding is valid, we just assume it is.

18599. advance by one character (code point) and one char_offset

18600. TypeError if wrong; class check, see E5 Section 15.10.6

18601. fall through

18602. Note: allow backtracking from p == ptr_end

18603. **comment:** * Note: ** - duk_match_regexp() is required not to longjmp() in ordinary "non-match" * conditions; a longjmp() will terminate the entire matching process. ** - Clearing saved[] is not necessary because backtracking does it ** - Backtracking also rewinds ctx.recursion back to zero, unless an * internal/limit error occurs (which causes a longjmp()) ** - If we supported anchored matches, we would break out here * unconditionally; however, ECMAScript regexps don't have anchored * matches. It might make sense to implement a fast bail-out if * the regexp begins with '^' and sp is not 0: currently we'll just * run through the entire input string, trivially failing the match * at every non-zero offset.

label: code-design

18604. never here

18605. * Regexp instance check, bytecode check, input coercion. ** See E5 Section 15.10.6.

18606. * Get starting character offset for match, and initialize 'sp' based on it. ** Note: lastIndex is non-configurable so it must be present (we check the * internal class of the object above, so we know it is). User code can set * its value to an arbitrary (garbage) value though; E5 requires that lastIndex * be coerced to a number before using. The code below works even if the * property is missing: the value will then be coerced to zero. ** Note: lastIndex may be outside Uint32 range even after ToInteger() coercion. * For instance, ToInteger(+Infinity) = +Infinity. We track the match offset * as an integer, but pre-check it to be inside the 32-bit range before the loop. * If not, the check in E5 Section 15.10.6.2, step 9.a applies.

18607. * Matching complete, create result array or return a 'null'. Update lastIndex * if necessary. See E5 Section 15.10.6.2. ** Because lastIndex is a character (not byte) offset, we need the character * length of the match which we conveniently get as a side effect of interning * the matching substring (0th index of result array). ** saved[0] start pointer (~ byte offset) of current match * saved[1] end pointer (~ byte offset) of current match (exclusive) * char_offset start character offset of current match (->.index of result) * char_end_offset end character offset (computed below)

18608. Assumes that saved[0] and saved[1] are always * set by regexp bytecode (if not, char_end_offset * will be zero). Also assumes clen reflects the * correct char length.

18609. char_offset in [0, h_input->clen] (both ends inclusive), checked before entry

18610. backref n -> saved indices [n*2, n*2+1]

18611. Note: ctx.steps is intentionally not reset, it applies to the entire unanchored match

18612. buffer is automatically zeroed

18613. [... res_obj re_obj input bc saved_buf]

18614. **comment:** Note: don't bail out early, we must read all the ranges from * bytecode. Another option is to skip them efficiently after * breaking out of here. Prefer smallest code.

label: code-design

18615. NB: 'length' property is automatically updated by the array setup loop

18616. * Regexp executor. ** Safety: the ECMAScript executor should prevent user from reading and * replacing regexp bytecode. Even so, the executor must validate all * memory accesses etc. When an invalid access is detected (e.g. a 'save' * opcode to invalid, unallocated index) it should fail with an internal * error but not cause a segmentation fault. ** Notes: ** - Backtrack counts are limited to unsigned 32 bits but should * technically be duk_size_t for strings longer than 4G chars. * This also requires a regexp bytecode change.

18617. avoid calling at end of input, will DUK_ERROR (above check suffices to avoid this)

18618. [... re_obj input] -> [... re_obj input bc]

18619. idx is unsigned, < 0 check is not necessary

18620. * Helpers for UTF-8 handling ** For bytecode readers the duk_uint32_t and duk_int32_t types are correct * because they're used for more than just codepoints.

18621. [... res_obj]

18622. **comment:** XXX: lastIndex handling produces a lot of asm

label: code-design

18623. * Exposed matcher function which provides the semantics of RegExp.prototype.exec(). ** RegExp.prototype.test() has the same semantics as exec() but does not return the * result object (which contains the matching string and capture groups). Currently * there is no separate test() helper, so a temporary result object is created and * discarded if test() is needed. This is intentional, to save code space. ** Input stack: [... re_obj input] * Output stack: [... result]

18624. canonicalized by compiler

18625. not a wordchar

18626. Note: if atom were to contain e.g. captures, we would need to * re-match the atom to get correct captures. Simply quantifiers * do not allow captures in their atom now, so this is not an issue.

18627. -> [... re_obj input bc saved_buf lastIndex]

18628. * Note: ** - Intentionally attempt (empty) match at char_offset == k_input->clen ** - Negative char_offsets have been eliminated and char_offset is duk_uint32_t * -> no need or use for a negative check

18629. fail: restore saves

18630. [... re_obj input bc saved_buf]

18631. ToInteger(lastIndex)

18632. **comment:** XXX: these last tricks are unnecessary if the function is made * a genuine native function.
label: code-design

18633. capture is 'undefined', always matches!

18634. **comment:** Wipe capture range and save old values for backtracking. ** XXX: this typically happens with a relatively small idx_count. * It might be useful to handle cases where the count is small * (say <= 8) by saving the values in stack instead. This would * reduce memory churn and improve performance, at the cost of a * slightly higher code footprint.
label: code-design

18635. **comment:** * Needs a save of multiple saved[] entries depending on what range * may be overwritten. Because the regexp parser does no such analysis, * we currently save the entire saved array here. Lookaheads are thus * a bit expensive. Note that the saved array is not needed for just * the lookahead sub-match, but for the matching of the entire sequel. ** The temporary save buffer is pushed on to the valstack to handle * errors correctly. Each lookahead causes a C recursion and pushes * more stuff on the value stack. If the C recursion limit is less * than the value stack spare, there is no need to check the stack. * We do so regardless, just in case.
label: code-design

18636. signed integer encoding needed to work with UTF-8

18637. Captures which are undefined have NULL pointers and are returned * as 'undefined'. The same is done when saved[] pointers are insane * (this should, of course, never happen in practice).

18638. * Byte matching for back-references would be OK in case- * sensitive matching. In case-insensitive matching we need * to canonicalize characters, so back-reference matching needs * to be done with codepoints instead. So, we just decode * everything normally here, too. ** Note: back-reference index which is 0 or higher than * NCapturingParens (= number of capturing parens in the * -entire- regexp) is a compile time error. However, a * backreference referring to a valid capture which has * not matched anything always succeeds! See E5 Section * 15.10.2.9, step 5, sub-step 3.

18639. Get a (possibly canonicalized) input character from current sp. The input * itself is never modified, and captures always record non-canonicalized * characters even in case-insensitive matching.

18640. force_global

18641. match: keep saves

18642. * Basic context initialization. ** Some init values are read from the bytecode header * whose format is (UTF-8 codepoints): * uint flags * uint nsaved (even, 2n+2 where n = num captures)

18643. regexp support disabled

18644. Backtrack utf-8 input and return a (possibly canonicalized) input character.

18645. [... re_obj input bc saved_buf res_obj val]

18646. lastIndex must be ignored for non-global regexps, but get the * value for (theoretical) side effects. No side effects can * really occur, because lastIndex is a normal property and is * always non-configurable for RegExp instances.

18647. **comment:** XXX: Array size is known before and (2 * re_ctx.nsaved) but not taken * advantage of now. The array is not compacted either, as regexp match * objects are usually short lived.
label: code-design

18648. and even number

18649. read header

18650. * Match loop. ** Try matching at different offsets until match found or input exhausted.

18651. * E5 Section 15.10.2.6. The previous and current character * should -not- be canonicalized as they are now. However, * canonicalization does not affect the result of IsWordChar() * (which depends on Unicode characters never canonicalizing * into ASCII characters) so this does not matter.

18652. [... re_obj input bc saved_buf res_obj]

18653. **comment:** This variant is needed by String.prototype.split(); it needs to perform * global-style matching on a cloned RegExp which is potentially non-global.
label: code-design

18654. dummy so sp won't get updated

18655. utf-8 continuation bytes have the form 10xx xxxx

18656. regexp compiler should catch these

18657. global regexp: lastIndex updated on match

18658. expt 0xffff is infinite/nan

18659. expt 0x000 is zero/subnormal

18660. expt values [0x001,0x7fe] = normal

18661. * Replacements for missing platform functions. ** Unlike the originals, fpclassify() and signbit() replacements don't * work on any floating point types, only doubles. The C typing here * mimics the standard prototypes.

18662. !DUK_SINGLE_FILE

18663. DUK_REPLACEMENTS_H_INCLUDED

18664. * Byte order. Important to self check, because on some exotic platforms * there is no actual detection but rather assumption based on platform * defines.

18665. Test without volatiles

18666. >>> struct.unpack('>d', '4000112233445566'.decode('hex')) * (2.008366013071895,)

18667. * Zero sign, see misc/tcc_zerosign2.c.

18668. Some internal code now assumes that all duk_uint_t values * can be expressed with a duk_size_t.

18669. * Union aliasing, see misc/clang_aliasing.c.

18670. little endian

18671. Test signaling NaN and alias assignment in all endianness combinations.

18672. Catch a double-to-int64 cast issue encountered in practice.

18673. * Casting

18674. **comment:** * Struct size/alignment if platform requires it ** There are some compiler specific struct padding pragmas etc in use, this * selftest ensures they're correctly detected and used.
label: code-design

18675. * Self tests to ensure execution environment is sane. Intended to catch * compiler/platform problems which cannot be detected at compile time.

18676. * Unions and structs for self tests

18677. big endian

18678. nop

18679. * Packed tval sanity

18680. DUK_USE_SELF_TESTS

18681. * 64-bit arithmetic ** There are some platforms/compilers where 64-bit types are available * but don't work correctly. Test for known cases.

18682. **comment:** Same test with volatiles
label: test

18683. * Self test main
18684. * This test fails on an exotic ARM target; double-to-uint * cast is incorrectly clamped to -signed- int highest value. * *
 <https://github.com/svaarala/duktape/issues/336>
18685. * Basic double / byte union memory layout.
18686. mixed endian
18687. Allow very small lenience because some compilers won't parse * exact IEEE double constants (happened in matrix testing with * Linux gcc-4.8 -m32 at least).
18688. * >>> struct.pack('>d', 102030405060).encode('hex') * '4237c17c6dc40000'
18689. Note that byte order doesn't affect this test: all bytes in * 'test' will be 0xFF for two's complement.
18690. Oxdeadbeef in decimal
18691. * Two's complement arithmetic.
18692. * DUK_BSswap macros
18693. * <https://github.com/svaarala/duktape/issues/127#issuecomment-77863473>
18694. no check
18695. This testcase fails when Emscripten-generated code runs on Firefox. * It's not an issue because the failure should only affect packed * duk_tval representation, which is not used with Emscripten.
18696. **comment:** * Various sanity checks for typing
 label: code-design
18697. * Selftest code
18698. DUK_SELFTEST_H_INCLUDED
18699. Object property access
18700. Compiler
18701. Misc
18702. Limits
18703. Mostly API and built-in method related
18704. **comment:** still in use with verbose messages
 label: code-design
18705. Regexp
18706. **comment:** * Shared error message strings * * To minimize code footprint, try to share error messages inside Duktape * code. Modern compilers will do this automatically anyway, this is mostly * for older compilers.
 label: code-design
18707. JSON
18708. DUK_ERRMSG_H_INCLUDED
18709. **comment:** still in use with verbose messages
 label: code-design
18710. !DUK_SINGLE_FILE
18711. **comment:** * Shared error messages: declarations and macros * * Error messages are accessed through macros with fine-grained, explicit * error message distinctions. Concrete error messages are selected by the * macros and multiple macros can map to the same concrete string to save * on code footprint. This allows flexible footprint/verbosity tuning with * minimal code impact. There are a few limitations to this approach: * (1) switching between plain messages and format strings doesn't work * conveniently, and (2) conditional strings are a bit awkward to handle. * * Because format strings behave differently in the call site (they need to * be followed by format arguments), they have a special prefix (DUK_STR_FMT_ * and duk_str_fmt_). * * On some compilers using explicit shared strings is preferable; on others * it may be better to use straight literals because the compiler will combine * them anyway, and such strings won't end up unnecessarily in a symbol table.
 label: code-design
18712. DUK_USE_FASTINT
18713. zero
18714. exponent 1070
18715. **comment:** unused
 label: code-design
18716. exponent 0
18717. positive
18718. * Manually optimized number-to-double conversion
18719. 0
18720. negative
18721. exponents 1023 to 1069
18722. Note: reject negative zero.
18723. **comment:** * Manually optimized double-to-fastint downgrade check. * * This check has a large impact on performance, especially for fastint * slow paths, so must be changed carefully. The code should probably be * optimized for the case where the result does not fit into a fastint, * to minimize the penalty for "slow path code" dealing with fractions etc. * * At least on one tested soft float ARM platform double-to-int64 coercion * is very slow (and sometimes produces incorrect results, see self tests). * This algorithm combines a fastint compatibility check and extracting the * integer value from an IEEE double for setting the tagged fastint. For * other platforms a more naive approach might be better. * * See doc/fastint.rst for details.
 label: code-design
18724. **comment:** XXX: optimize for packed duk_tval directly?
 label: code-design
18725. implicit leading one
18726. avoid unary minus on unsigned
18727. 1 << 52
18728. DUK_USE_FASTINT && DUK_USE_PACKED_TVAL
18729. **comment:** Note: not initializing all bytes is normally not an issue: Duktape won't * read or use the uninitialized bytes so valgrind won't issue warnings. * In some special cases a harmless valgrind warning may be issued though. * For example, the DumpHeap debugger command writes out a compiled function's * 'data' area as is, including any uninitialized bytes, which causes a * valgrind warning.
 label: code-design
18730. Sign extend: 0x0000##00 -> 0x##000000 -> sign extend to 0xssssss##
18731. * Portable 12-byte representation
18732. XXX: clumsy sign extend and masking of 16 topmost bits
18733. marker; not actual tagged type
18734. marker; not actual tagged value
18735. **comment:** This is performance critical because it's needed for every DECREF. * Take advantage of the fact that the first heap allocated tag is 8, * so that bit 3 is set for all heap allocated tags (and never set for * non-heap-allocated tags).
 label: code-design
18736. DUK_USE_64BIT_OPS
18737. Assumes that caller has normalized NaNs, otherwise trouble ahead.
18738. use duk_double_union as duk_tval directly
18739. This seems reasonable overall.
18740. **comment:** two casts to avoid gcc warning: "warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]"
 label: code-design
18741. DUK_TAG_NUMBER is intentionally first, as it is the default clause in code * to support the 8-byte representation. Further, it is a non-heap-allocated * type so it should come before DUK_TAG_STRING. Finally, it should not break * the tag value ranges covered by case-clauses in a switch-case.
18742. for convenience

18743. **comment:** avoid tag 0xffff0, no risk of confusion with negative infinity
label: code-design

18744. embed: integer value

18745. DUK_TAG_NUMBER would logically go here, but it has multiple 'tags'

18746. **comment:** XXX: fast int-to-double
label: code-design

18747. max, excl. varargs marker

18748. DUK_USE_PACKED_TVAL

18749. if present, forces 16-byte duk_tval

18750. varargs marker

18751. Note: masking is done for 'i' to deal with negative numbers correctly

18752. * Packed 8-byte representation

18753. in non-packed representation we don't care about which NaN is used

18754. embed: void ptr

18755. decoding

18756. 0xffff0 is -Infinity

18757. embed: duk_hobject ptr

18758. DUK_USE_FASTINT

18759. getters

18760. **comment:** This is performance critical because it appears in every DECREF.
label: code-design

18761. **comment:** the NaN variant we use
label: code-design

18762. Double casting for pointer to avoid gcc warning (cast from pointer to integer of different size)

18763. DUK_TVAL_H_INCLUDED

18764. embed: func ptr

18765. sanity

18766. fastint constants etc

18767. **comment:** * Tagged type definition (duk_tval) and accessor macros. ** Access all fields through the accessor macros, as the representation * is quite tricky. **
There are two packed type alternatives: an 8-byte representation * based on an IEEE double (preferred for compactness), and a 12-byte * representation (portability). The latter is needed also in e.g. * 64-bit environments (it usually pads to 16 bytes per value). ** Selecting the tagged type format involves many trade-offs (memory * use, size and performance of generated code, portability, etc), * see doc/types.rst for a detailed discussion (especially of how the * IEEE double format is used to pack tagged values). ** NB: because macro arguments are often expressions, macros should * avoid evaluating their argument more than once.
label: code-design

18768. embed: duk_hstring ptr

18769. Lightfunc flags packing and unpacking.

18770. first heap allocated, match bit boundary

18771. embed: 0 or 1 (false or true)

18772. setters

18773. not exposed

18774. embed: duk_hbuffer ptr

18775. tags

18776. embed: nothing

18777. * Convenience (independent of representation)

18778. =====

18779. * Unicode helpers

18780. zero-width joiner

18781. * Automatically generated by extract_chars.py, do not edit!

18782. * ASCII character constants ** C character literals like 'x' have a platform specific value and do * not match ASCII (UTF-8) values on e.g. EBCDIC platforms.
So, use * these (admittedly awkward) constants instead. These constants must * also have signed values to avoid unexpected coercions in comparisons. * *
<http://en.wikipedia.org/wiki/ASCII>

18783. up to 36 bit codepoints

18784. * Unicode tables

18785. duk_unicode_support.c

18786. !DUK_SINGLE_FILE

18787. all codepoints up to U+10FFFF

18788. DUK_UNICODE_H_INCLUDED

18789. * Automatically generated by extract_caseconv.py, do not edit!

18790. * Useful Unicode codepoints * * Integer constants must be signed to avoid unexpected coercions * in comparisons.

18791. * Prototypes

18792. all codepoints up to U+FFFF

18793. * Extern

18794. zero-width non-joiner

18795. * UTF-8 / XUTF-8 / CESU-8 constants

18796. http://en.wikipedia.org/wiki/Replacement_character#Replacement_character

18797. * E5 Section 15.10.2.6 "IsWordChar" abstract operation. Assume * x < 0 for characters read outside the string.

18798. does not fit into an uchar

18799. 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [26 bits]

18800. * UTF-8 decoder which accepts longer than standard byte sequences. * This allows full 32-bit code points to be used.

18801. DUK_USE_REGEXP_SUPPORT

18802. curr char

18803. 0x20...0x2f

18804. * Unicode letter check.

18805. * Replace valstack top with case converted version.

18806. 0x60...0x6f

18807. 0xxx xxxx [7 bits]

18808. 0x40...0x4f

18809. exposed because lexer needs these too

18810. * XUTF-8 and CESU-8 encoding/decoding

18811. 1110 xxxx 10xx xxxx 10xx xxxx [16 bits]

18812. without explicit non-BMP support, assume non-BMP characters * are always accepted as identifier characters.

18813. 110x xxxx 10xx xxxx [11 bits]

18814. * Unicode codepoints above U+FFFF are encoded as surrogate * pairs here. This ensures that all CESU-8 codepoints are * 16-bit values as expected in EcmaScript.
The surrogate * pairs always get a 3-byte encoding (each) in CESU-8. * See: http://en.wikipedia.org/wiki/Surrogate_pair ** 20-bit codepoint, 10 bits (A and B) per surrogate pair: * * x = 0b00000000 0000AAAA AAAAABBBBBBBB * sp1 = 0b110110AA AAAAaaaa (0xd800 + ((x >> 10) & 0x3ff)) * sp2 = 0b110111BB BBBBCCCC (0xdc00 + (x & 0x3ff)) * * Encoded into CESU-8: * * sp1 -> 0b11101101 (0xe0 + ((sp1 >> 12) & 0x0f)) * -> 0b1010AAAA (0x80 + ((sp1 >> 6) & 0x3f)) * -> 0b10AAAAAA (0x80 + (sp1 & 0x3f)) * sp2 -> 0b11101101 (0xe0 + ((sp2 >> 12) & 0x0f)) * -> 0b1011BBBB (0x80 + ((sp2 >> 6) &

0x3f)) * -> 0b10BBBBBB (0x80 + (sp2 & 0x3f)) * * Note that 0x10000 must be subtracted first. The code below * avoids the sp1, sp2 temporaries which saves around 20 bytes * of code.

18815. XXX: add 'language' argument when locale/language sensitive rule * support added.

18816. next does not exist or next is not a letter

18817. Encoded as surrogate pair, each encoding to 3 bytes for * 6 bytes total. Codepoints above U+10FFFF encode as 6 bytes * too, see duk_unicode_encode_cesu8().

18818. 11 bits

18819. 0x50...0x5f

18820. **comment:** Fast canonicalization lookup at the cost of 128kB footprint.

label: code-design

18821. DUK_USE_PREFER_SIZE

18822. fall through

18823. * "IdentifierPart" production check.

18824. fast path for ASCII

18825. prev exists and is not a letter

18826. 36 bits

18827. verified at beginning

18828. complex, multicharacter conversion

18829. DUK_USE_ASSERTIONS

18830. Note: masking of 'x' is not necessary because of * range check and shifting -> no bits overlapping * the marker should be set.

18831. * Various Unicode help functions for character classification predicates, * case conversion, decoding, etc.

18832. Compute (extended) utf-8 length without codepoint encoding validation, * used for string interning. * * NOTE: This algorithm is performance critical, more so than string hashing * in some cases. It is needed when interning a string and needs to scan * every byte of the string with no skipping. Having an ASCII fast path * is useful if possible in the algorithm. The current algorithms were * chosen from several variants, based on x64 gcc -O2 testing. See: * <https://github.com/svaaraala/duktape/pull/422> * * NOTE: must match src/dukutil.py:duk_unicode_unvalidated_utf8_length().

18833. Fall through to handle the rest.

18834. check pointer at end

18835. Full, aligned 4-byte reads.

18836. * "IdentifierStart" production check.

18837. **comment:** XXX: turkish / azeri, lowercase rules

label: code-design

18838. without explicit non-BMP support, assume non-BMP characters * are always accepted as letters.

18839. 0x00...0x0f

18840. **comment:** XXX: there are language sensitive rules for the ASCII range. * If/when language/locale support is implemented, they need to * be implemented here for the fast path. There are no context * sensitive rules for ASCII range.

label: code-design

18841. NULL is allowed, no output

18842. invalidates h_buf pointer

18843. Decode helper. Return zero on error.

18844. * Unicode range matcher * * Matches a codepoint against a packed bitstream of character ranges. * Used for slow path Unicode matching.

18845. 31 bits

18846. * Canonicalize() abstract operation needed for canonicalization of individual * codepoints during regexp compilation and execution, see E5 Section 15.10.2.8. * Note that codepoints are canonicalized one character at a time, so no context * specific rules can apply. Locale specific rules can apply, though.

18847. * Fast path tables

18848. * Regexp range tables

18849. 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [31 bits]

18850. Small variant; roughly 150 bytes smaller than the fast variant.

18851. cp == -1 (EOF) never matches and causes return value 0

18852. * Complex case conversion helper which decodes a bit-packed conversion * control stream generated by unicode/extract_caseconv.py. The conversion * is very slow because it runs through the conversion data in a linear * fashion to save space (which is why ASCII characters have a special * fast path before arriving here). * * The particular bit counts etc have been determined experimentally to * be small but still sufficient, and must match the Python script * (src/extract_caseconv.py). * * The return value is the case converted codepoint or -1 if the conversion * results in multiple characters (this is useful for regexp Canonicalization * operation). If 'buf' is not NULL, the result codepoint(s) are also * appended to the hbuffer. * * Context and locale specific rules must be checked before consulting * this function.

18853. Encode to CESU-8; 'out' must have space for at least * DUK_UNICODE_MAX_CESU8_LENGTH bytes; codepoints above U+10FFFF * will encode to garbage but won't overwrite the output buffer.

18854. default: no change

18855. * Final sigma context specific rule. This is a rather tricky * rule and this handling is probably not 100% correct now. * The rule is not locale/language specific so it is supported.

18856. Must match src/extract_chars.py, generate_match_table3()

18857. * E5 Section 7.3 * * A LineTerminatorSequence essentially merges <CR> <LF> sequences * into a single line terminator. This must be handled by the caller.

18858. 1:1 or special conversions, but not locale/context specific: script generated rules

18859. 10xx xxxx -> invalid

18860. * Unicode letter is now taken to be the categories: * * Lu, Ll, Lt, Lm, Lo * * (Not sure if this is exactly correct.) * * The ASCII fast path consists of: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z']

18861. Capital sigma occurred at "end of word", lowercase to * U+03C2 = GREEK SMALL LETTER FINAL SIGMA. Otherwise * fall through and let the normal rules lowercase it to * U+03C3 = GREEK SMALL LETTER SIGMA.

18862. 0x10...0x1f

18863. Align 'p' to 4; the input data may have arbitrary alignment. * End of string check not needed because blen >= 16.

18864. This seems like a good overall approach. Fast path for ASCII in 4 byte * blocks.

18865. on first round, skip

18866. surrogate pairs get encoded here

18867. **comment:** XXX: could add a fast path to process chunks of input codepoints, * but relative benefit would be quite small.

label: code-design

18868. end marker

18869. used by e.g. duk_regex_executor.c, string built-ins

18870. prev char

18871. ASCII fast path

18872. 0x70...0x7f

18873. **comment:** * E5 Section 7.6: * * IdentifierPart: * IdentifierStart * UnicodeCombiningMark * UnicodeDigit * UnicodeConnectorPunctuation * <ZWNJ> [U+200C] * <ZWJ> [U+200D] * * IdentifierPart production has one multi-character production * as part of its IdentifierStart alternative. The '\ character * of an escape sequence is not matched here, see discussion in * duk_unicode_is_identifier_start(). * * To match non-ASCII characters (codepoints >= 0x80), a very slow * linear range-by-range scan is used. The codepoint is first compared * to the IdentifierStart ranges, and if it doesn't match, then to a * set consisting of code points in IdentifierPart but not in * IdentifierStart. This is done to keep the unicode range data small, * at the expense of speed. * * The ASCII fast path consists of: * * 0x0030 ... 0x0039 ['0' ... '9', UnicodeDigit] * 0x0041 ... 0x005a ['A' ... 'Z', IdentifierStart] * 0x0061 ... 0x007a ['a' ... 'z', IdentifierStart] * 0x0024 ['\$', IdentifierStart] * 0x005f ['_', IdentifierStart and * UnicodeConnectorPunctuation] * * UnicodeCombiningMark has no code points <= 0x7f. * * The matching code reuses the "identifier start" tables, and then * consults a separate range set for characters in "identifier part" * but not in "identifier start". These can be extracted with the * "src/extract_chars.py" script. * * UnicodeCombiningMark -> categories Mn, Mc * UnicodeDigit -> categories Nd * UnicodeConnectorPunctuation -> categories Pc

label: code-design

18874. **comment:** XXX: lithuanian, explicit dot rules

label: code-design

18875. **comment:** * Note: the description in E5 Section 15.10.2.6 has a typo, it * contains 'A' twice and lacks 'a'; the intent is [0-9a-zA-Z_].

label: documentation

18876. 0x30...0x3f

18877. **comment:** * E5 Section 7.6: ** IdentifierStart: * UnicodeLetter * \$ * _ * \ UnicodeEscapeSequence ** IdentifierStart production has one multi-character production: ** \ UnicodeEscapeSequence ** The '\ character is -not- matched by this function. Rather, the caller * should decode the escape and then call this function to check whether the * decoded character is acceptable (see discussion in E5 Section 7.6). ** The "UnicodeLetter" alternative of the production allows letters * from various Unicode categories. These can be extracted with the * "src/extract_chars.py" script. ** Because the result has hundreds of Unicode codepoint ranges, matching * for any values >= 0x80 are done using a very slow range-by-range scan * and a packed range format. ** The ASCII portion (codepoints 0x00 ... 0x7f) is fast-patched below because * it matters the most. The ASCII related ranges of IdentifierStart are: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z'] * 0x0024 ['\$'] * 0x005f['_']

label: code-design

18878. 26 bits

18879. 64-bit OK because always >= 0

18880. **comment:** XXX: lithuanian not implemented

label: requirement

18881. [r1,r2] is the range

18882. ASCII (and EOF) fast path -- quick accept and reject

18883. Encode to extended UTF-8; 'out' must have space for at least * DUK_UNICODE_MAX_XUTF8_LENGTH bytes. Allows encoding of any * 32-bit (unsigned) codepoint.

18884. always >= 0

18885. number of continuation (non-initial) bytes in [0x80,0xbff]

18886. 1:1 conversion

18887. * "LineTerminator" production check.

18888. **comment:** 8-byte format could be: * 1111 1111 10xx xxxx [41 bits] ** However, this format would not have a zero bit following the * leading one bits and would not allow 0xFF to be used as an * "invalid xutf-8" marker for internal keys. Further, 8-byte * encodings (up to 41 bit code points) are not currently needed.

label: code-design

18889. multiple codepoint conversion or non-ASCII mapped to ASCII * --> leave as is.

18890. * E5 Section 7.2 specifies six characters specifically as * white space: * * 0009;<control>;Cc;0;S;;;;N;CHARACTER TABULATION;;;,* 000B;
<control>;Cc;0;S;;;;N;LINE TABULATION;;;,* 000C;<control>;Cc;0;WS;;;;N;FORM FEED (FF);;;,* 0020;SPACE;Zs;0;WS;;;;N;;;;* 00A0;NO-BREAK
SPACE;Zs;0;CS;<noBreak> 0020;;;N;NON-BREAKING SPACE;;;,* FFFF;ZERO WIDTH NO-BREAK SPACE;Cf;0;BN;;;N;BYTE ORDER MARK;;;,* It
also specifies any Unicode category 'Zs' characters as white * space. These can be extracted with the "src/extract_chars.py" script. * Current result: * * RAW
OUTPUT: * ===== * 0020;SPACE;Zs;0;WS;;;;N;;;;* 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;N;NON-BREAKING SPACE;;;,*
1680;OGHAM SPACE MARK;Zs;0;WS;;;;N;;;;* 180E;MONGOLIAN VOWEL SEPARATOR;Zs;0;WS;;;;N;;;;* 2000;EN QUAD;Zs;0;WS;2002;;;N;;;;*
2001;EM QUAD;Zs;0;WS;2003;;;N;;;;* 2002;EN SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 2003;EM SPACE;Zs;0;WS;<compat> 0020;;;N;;;;*
2004;THREE-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 2005;FOUR-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 2006;SIX-PER-EM
SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 2007;FIGURE SPACE;Zs;0;WS;<noBreak> 0020;;;N;;;;* 2008;PUNCTUATION SPACE;Zs;0;WS;<compat>
0020;;;N;;;;* 2009;THIN SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 200A;HAIR SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 202F;NARROW NO-BREAK
SPACE;Zs;0;CS;<noBreak> 0020;;;N;;;;* 205F;MEDIUM MATHEMATICAL SPACE;Zs;0;WS;<compat> 0020;;;N;;;;* 3000;IDEOGRAPHIC
SPACE;Zs;0;WS;<wide> 0020;;;N;;;;* * RANGES: * ===== * 0x0020 * 0x00a0 * 0x1680 * 0x180e * 0x2000 ... 0x200a * 0x202f * 0x205f * 0x3000 * * A
manual decoder (below) is probably most compact for this.

18891. end of input and last char has been processed

18892. 16 bits

18893. **comment:** DUK_USE_REGEXP_CANON_WORKAROUND

label: code-design

18894. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx [21 bits]

18895. **comment:** XXX: turkish / azeri

label: code-design

18896. U+03A3 = GREEK CAPITAL LETTER SIGMA

18897. 1111 1110 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [36 bits]

18898. Ensures space for maximum multi-character result; estimate is overkill.

18899. [...] input buffer]

18900. uppercase

18901. * Case conversion helper, with context/local sensitivity. * For proper case conversion, one needs to know the character * and the preceding and following
characters, as well as * locale/language.

18902. range conversion with a "skip"

18903. next char

18904. 7 bits

18905. * "WhiteSpace" production check.

18906. **comment:** Flip highest bit of each byte which changes * the bit pattern 10xxxxxx into 00xxxxxx which * allows an easy bit mask test.

label: test

18907. 21 bits

18908. **comment:** unused now, not needed until Turkish/Azeri

label: requirement

18909. **comment:** context and locale specific rules which cannot currently be represented * in the caseconv bitstream: hardcoded rules in C

label: code-design

18910. **comment:** Non-ASCII slow path (range-by-range linear comparison), very slow

label: code-design

18911. **comment:** XXX: 'internal error' is a bit of a misnomer

label: code-design

18912. 0: not IdentifierStart or IdentifierPart * 1: IdentifierStart and IdentifierPart * -1: IdentifierPart only

18913. * Automatically generated by extract_chars.py, do not edit!

18914. * Unicode support tables automatically generated during build.

18915. IdentifierStart production with Letter, ASCII, and non-BMP excluded

18916. duk_unicode_ids_m_let_noabmp[]

18917. duk_unicode_ids_m_let_noa[]

18918. duk_unicode_ids_noabmp[]

18919. IdentifierStart production with ASCII excluded

18920. duk_unicode_caseconv_lc[]

18921. IdentifierPart production with IdentifierStart and ASCII excluded

18922. duk_unicode_ids_noa[]

18923. IdentifierStart production with ASCII and non-BMP excluded

18924. IdentifierStart production with Letter and ASCII excluded

18925. IdentifierPart production with IdentifierStart, ASCII, and non-BMP excluded

18926. duk_unicode_idp_m_ids_noa[]

18927. * Case conversion tables generated using src/extract_caseconv.py.
18928. duk_unicode_idp_m_ids_noabmp[]
18929. * Automatically generated by extract_caseconv.py, do not edit!
18930. duk_unicode_caseconv_uc[]

18931. **comment:** * Unicode tables containing ranges of Unicode characters in a * packed format. These tables are used to match non-ASCII * characters of complex productions by resorting to a linear * range-by-range comparison. This is very slow, but is expected * to be very rare in practical Ecmascript source code, and thus * compactness is most important. * * The tables are matched using uni_range_match() and the format * is described in src/extract_chars.py.
label: code-design

18932. !DUK_SINGLE_FILE
18933. Insert bytes in the middle of the buffer from an external buffer.
18934. **comment:** * Buffer writer (dynamic buffer only) * * Helper for writing to a dynamic buffer with a concept of a "spare" area * to reduce resizes. You can ensure there is enough space beforehand and * then write for a while without further checks, relying on a stable data * pointer. Spare handling is automatic so call sites only indicate how * much data they need right now. * * There are several ways to write using bufwriter. The best approach * depends mainly on how much performance matters over code footprint. * The key issues are (1) ensuring there is space and (2) keeping the * pointers consistent. Fast code should ensure space for multiple writes * with one ensure call. Fastest inner loop code can temporarily borrow * the 'p' pointer but must write it back eventually. * * Be careful to ensure all macro arguments (other than static pointers like * 'thr' and 'bw_ctx') are evaluated exactly once, using temporaries if * necessary (if that's not possible, there should be a note near the macro). * Buffer write arguments often contain arithmetic etc so this is * particularly important here.
label: code-design

18935. * Utilities
18936. * Endian conversion
18937. Ensuring (reserving) space.
18938. Evaluates to (duk_uint8_t *) pointing to start of area.
18939. * Externs and prototypes
18940. NOTE: Multiple evaluation of 'ptr' in this macro.
18941. For now only needed by the debugger.
18942. Pointers may be NULL for a while when 'buf' size is zero and before any * ENSURE calls have been made. Once an ENSURE has been made, the pointers * are required to be non-NULL so that it's always valid to use memcpy() and * memmove(), even for zero size.
18943. **comment:** XXX: Migrate bufwriter and other read/write helpers to its own header?
label: code-design

18944. 2/4 -> 1/16 = 6.25% spare
18945. Working with the pointer and current size.
18946. **comment:** * Raw write/read macros for big endian, unaligned basic values. * Caller ensures there's enough space. The macros update the pointer * argument automatically on resizes. The idiom seems a bit odd, but * leads to compact code.
label: code-design

18947. DUK_UTIL_H_INCLUDED
18948. **comment:** XXX: Rework to use an always-inline function?
label: code-design

18949. Insert a reserved area somewhere in the buffer; caller fills it. * Evaluates to a (duk_uint_t *) pointing to the start of the reserved * area for convenience.
18950. No difference between raw/ensure because the buffer shrinks.
18951. Initialization and finalization (compaction), converting to other types.
18952. Miscellaneous.
18953. **comment:** XXX: add temporary duk__p pointer here too; sharing
label: code-design

18954. 'ptr' is evaluated both as LHS and RHS.
18955. Safe write calls which will ensure space first.
18956. must match genhashsizes.py
18957. * Bitstream decoder
18958. Append bytes from a slice already in the buffer.
18959. Note: assumes that duk_util_probe_steps size is 32
18960. * Bitstream encoder
18961. Fast write calls which assume you control the spare beforehand. * Multibyte write variants exist and use a temporary write pointer * because byte writes alias with anything: with a stored pointer * explicit pointer load/stores get generated (e.g. gcc -Os).
18962. Insert bytes in the middle of the buffer from a slice already * in the buffer. Source offset is interpreted "before" the operation.
18963. Make underlying buffer compact to match DUK_BW_GET_SIZE().
18964. No duk_bw_remove_ensure_slice(), functionality would be identical.
18965. Reset to zero size, keep current limit.
18966. Remove a slice from inside buffer.
18967. remaining
18968. **comment:** Note: cannot read more than 24 bits without possibly shifting top bits out. * Fixable, but adds complexity.
label: code-design

18969. * Bitstream decoder.
18970. If ctx->offset >= ctx->length, we "shift zeroes in" * instead of croaking.
18971. Decode a one-bit flag, and if set, decode a value of 'bits', otherwise return * default value. Return value is signed so that negative marker value can be * used by caller as a "not present" value.
18972. Extract 'top' bits of curval; note that the extracted bits do not need * to be cleared, we just ignore them on next round.
18973. Decode 'bits' bits from the input stream (bits must be 1...24). * When reading past bitstream end, zeroes are shifted in. The result * is signed to match duk_bd_decode_flagged.
18974. If buffer has been exhausted, truncate bitstream
18975. **comment:** This limitation would be fixable but adds unnecessary complexity.
label: code-design

18976. * Bitstream encoder.
18977. buffer size is >= 1
18978. * Fast buffer writer with spare management.
18979. overflow
18980. for manual torture testing: tight allocation, useful with valgrind
18981. Target is before source. Source offset is expressed as * a "before change" offset. Account for the memmove.
18982. point to start of 'reserved area'
18983. not reachable
18984. Don't support "straddled" source now.
18985. **comment:** * Macro support functions for reading/writing raw data. * * These are done using memcpy to ensure they're valid even for unaligned * reads/writes on platforms where alignment counts. On x86 at least gcc * is able to compile these into a bswap+mov. "Always inline" is used to * ensure these macros compile to minimal code. * * Not really bufwriter related, but currently used together.
label: code-design

18986. Make buffer compact, matching current written size.
18987. **comment:** * Macro support functions (use only macros in calling code)
label: code-design

18988. We could do this operation without caller updating bw_ctx->ptr, * but by writing it back here we can share code better.

18989. This is important to ensure dynamic buffer data pointer is not * NULL (which is possible if buffer size is zero), which in turn * causes portability issues with e.g. memmove() and memcpy().

18990. Resize target buffer for requested size. Called by the macro only when the * fast path test (= there is space) fails.

18991. **comment:** Portability workaround is required for platforms without * unaligned access. The replacement code emulates little * endian access even on big endian architectures, which is * OK as long as it is consistent for a build.
label: code-design

18992. DUK_USE_STRHASH_DENSE

18993. * Hash function duk_util_hashbytes(). ** Currently, 32-bit MurmurHash2. ** Don't rely on specific hash values; hash function may be endianness * dependent, for instance.

18994. 'magic' constants for Murmurhash2

18995. prediction corrections for prime list (see genhashsizes.py)

18996. hash size ratio goal, must match genhashsizes.py

18997. Awkward inclusion condition: drop out of compilation if not needed by any * call site: object hash part or probing stringtable.

18998. * Round a number upwards to a prime (not usually the nearest one). ** Uses a table of successive 32-bit primes whose ratio is roughly * constant. This keeps the relative upwards 'rounding error' bounded * and the data size small. A simple 'predict-correct' compression is * used to compress primes to one byte per prime. See genhashsizes.py * for details. ** The minimum prime returned here must be coordinated with the possible * probe sequence steps in duk_hobject and duk_heap stringtable.

18999. prediction: portable variant using doubles if 64-bit values not available

19000. minimum prime

19001. correction

19002. floor(1.15 * (1 << 10))

19003. 32-bit x 11-bit = 43-bit, fits accurately into a double

19004. may happen if size is very close to 2^32-1

19005. probe steps (see genhashsizes.py), currently assumed to be 32 entries long * (DUK_UTIL_GET_HASH_PROBE_STEP macro).

19006. DUK_USE_HOBJECT_HASH_PART || DUK_USE_STRTAB_PROBE

19007. 0x10-0x1f

19008. 0xf0-0xff

19009. 0xb0-0xbf

19010. 0xc0-0xcf

19011. 0x90...0x9f

19012. 0x90-0x9f

19013. 0x10...0x1f

19014. 0xd0...0xdf

19015. 0x20-0x2f

19016. Preshifted << 4. Must use 16-bit entry to allow negative value signaling.

19017. * Misc util stuff

19018. -1 = error, -2 = allowed whitespace, -3 = padding ('='), 0...63 decoded bytes

19019. * Arbitrary byteswap for potentially unaligned values ** Used to byteswap pointers e.g. in debugger code.

19020. For now only needed by the debugger.

19021. 0x60-0x6f

19022. 0x20...0x2f

19023. 0xa0-0xaf

19024. 0x60...0x6f

19025. 0x40...0x4f

19026. 0xe0-0xef

19027. * Table for base-64 decoding

19028. DUK_USE_HEX_FASTPATH

19029. A...P

19030. -1 if invalid

19031. * Table for hex decoding ASCII hex digits

19032. 0x30-0x3f

19033. 0xc0...0xcf

19034. 0x00...0x0f

19035. Lookup to encode one byte directly into 2 characters: ** def genhextab(bswap): * for i in xrange(256): * t = chr(i).encode('hex') * if bswap: * t = t[1] + t[0] * print('0x' + t.encode('hex') + 'U') * print('big endian'); genhextab(False) * print('little endian'); genhextab(True)

19036. 0x80-0x8f

19037. 0xb0...0xbf

19038. Q..f

19039. 0xa0...0xaf

19040. 0x70...0x7f

19041. * Table for hex encoding bytes

19042. 0x40-0x4f

19043. * Table for base-64 encoding

19044. 0x30...0x3f

19045. 0x00-0x0f

19046. 0x80...0x8f

19047. DUK_USE_INTEGER_BE

19048. g...v

19049. 0xf0...0xff

19050. 0x70-0x7f

19051. 0xd0...0xdf

19052. 0x50-0x5f

19053. * Lowercase digits for radix values 2 to 36. Also doubles as lowercase * hex nybble table.

19054. DUK_USE_BASE64_FASTPATH

19055. 0x50...0x5f

19056. 0xe0...0xef

19057. w.../

19058. enough to cover the whole mantissa

19059. if duk_uint32_l is exactly 32 bits, this is a NOP

19060. only use the highest bit

19061. **comment:** * XXX: could make this a lot faster if we create the double memory * representation directly. Feasible easily (must be uniform random).
label: code-design

19062. * A tiny random number generator. ** Currently used for Math.random(). ** http://www.woodmann.com/forum/archive/index.php/t-3100.html

19063. * Context management

19064. Return codes for protected calls (duk_safe_call(), duk_pcall())

19065. RangeError

19066. set/clear writable

19067. DUK_USE_64BIT_OPS

19068. Ecmascript date range is 100 million days from Epoch: $* > 100e6 * 24 * 60 * 60 * 1000 // 100M$ days in millisecs $* 8640000000000000 * (= 8.64e15)$
19069. * Coercion operations: in-place coercion, return coerced value where * applicable. If index is invalid, throw error. Some coercions may * throw an expected error
(e.g. from a `toString()` or `valueOf()` call) * or an internal error (e.g. from out of memory).

19070. packed tval
19071. (internal) catch compilation errors
19072. * END PUBLIC API
19073. E == 0x7ff, topmost four bits of F != 0 => assume NaN
19074. APIError
19075. * BEGIN PUBLIC API
19076. timeval breakdown: internal time value NaN -> RangeError (`toISOString()`)
19077. internal flag: external buffer
19078. enumerate internal properties (regardless of enumerability)
19079. Ecmascript E5 specification error codes
19080. * Debugger (debug protocol)
19081. lstring
19082. timeval breakdown: internal time value NaN -> zero
19083. include time part in string conversion result
19084. prefer number
19085. Note: parentheses are required so that the comma expression works in assignments.
19086. * Duktape public API for Duktape 1.5.2. * * See the API reference for documentation on call semantics. * The exposed API is inside the
DUK_API_PUBLIC_H_INCLUDED * include guard. Other parts of the header are Duktape * internal and related to platform/compiler/feature detection. * * Git
commit cad34ae155acb0846545ca6bf2d29f9463b22bbb (v1.5.2). * Git branch HEAD. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and *
licensing information.
19087. internal flag value: throw if mask doesn't match
19088. set getter (given on value stack)
19089. no error if file does not exist
19090. * Memory management * * Raw functions have no side effects (cannot trigger GC).
19091. Coercion hints
19092. is_copy
19093. * Variable access
19094. weekday: 0 to 6, 0=sunday, 1=monday, etc
19095. DUK_DBUNION_H_INCLUDED
19096. lightweight function pointer
19097. * String manipulation
19098. getter: subtract 1900 from year when getting year part
19099. Value mask types, used by e.g. `duk_get_type_mask()`
19100. Ecmascript undefined
19101. file
19102. Log levels
19103. Ecmascript boolean: 0 or 1
19104. * Public API specific typedefs * * Many types are wrapped by Duktape for portability to rare platforms * where e.g. 'int' is a 16-bit type. See practical typing
discussion * in Duktape web documentation.
19105. flags
19106. AssertionError
19107. only enumerate array indices
19108. * Stack management
19109. Flags for `duk_def_prop()` and its variants
19110. NOTE: when writing a Date provider you only need a few specific * flags from here, the rest are internal. Avoid using anything you * don't need.
19111. * Date provider related constants * * NOTE: These are "semi public" - you should only use these if you write * your own platform specific Date provider, see
doc/datetime.rst.
19112. internal: request fixed buffer result
19113. Number of value stack entries (in addition to actual call arguments) * guaranteed to be allocated on entry to a Duktape/C function.
19114. * Indexes of various types with respect to big endian (logical) layout
19115. no error (e.g. from `duk_get_error_code()`)
19116. LICENSE.txt
19117. SyntaxError
19118. * Duktape/C function magic value
19119. internal: request dynamic buffer result
19120. * Avoid C++ name mangling
19121. **comment:** XXX: replace with `TypeError?`
 label: code-design
19122. **comment:** XXX: to be removed?
 label: code-design
19123. Git commit, describe, and branch for Duktape build. Useful for * non-official snapshot builds so that application code can easily log * which Duktape snapshot
was used. Not available in the Ecmascript * environment.
19124. DUK_API_PUBLIC_H_INCLUDED
19125. Compilation flags for `duk_compile()` and `duk_eval()`
19126. timeval breakdown: convert month and day-of-month parts to one-based (default is zero-based)
19127. Duktape version, (major * 10000) + (minor * 100) + patch. Allows C code * to #ifdef against Duktape API version. The same value is also available * to
Ecmascript code in `Duktape.version`. Unofficial development snapshots * have 99 for patch level (e.g. 0.10.99 would be a development version * after 0.10.0 but
before the next official release).
19128. * Buffer
19129. set enumerable (effective if DUK_DEFPROP_HAVE_ENUMERABLE set)
19130. set configurable (effective if DUK_DEFPROP_HAVE_CONFIGURABLE set)
19131. don't walk prototype chain, only check own properties
19132. UnimplementedError
19133. (internal) omit eval result
19134. **comment:** * If no variadic macros, `__FILE__` and `__LINE__` are passed through globals * which is ugly and not thread safe.
 label: requirement
19135. last value is func pointer, arguments follow in parens
19136. set/clear configurable
19137. Return codes for C functions (shortcut for throwing an error)
19138. * Pop operations
19139. * Require operations: no coercion, throw error if index or type * is incorrect. No defaulting.
19140. * Bytecode load/dump
19141. compile function code (instead of global code)
19142. enumerate a proxy object itself without invoking proxy behavior
19143. **comment:** XXX: native 64-bit byteswaps when available

label: code-design

19144. **comment:** XXX: These calls are incomplete and not usable now. They are not (yet) * part of the public API.

label: code-design

19145. Error

19146. DUK_USE_FILE_IO

19147. ReferenceError

19148. timeval breakdown: replace year with equivalent year in the [1971,2037] range for DST calculations

19149. * Ecmascript operators

19150. args

19151. * Some defines forwarded from feature detection

19152. (internal) take strlen() of src_buffer (avoids double evaluation in macro)

19153. set/clear enumerable

19154. internal use

19155. * Error handling

19156. EvalError

19157. * Object prototype

19158. setter: perform 2-digit year fixup (00...99 -> 1900...1999)

19159. nop: no need to normalize

19160. * Compilation and evaluation

19161. or is a normalized NaN

19162. AllocError

19163. month: 0 to 11

19164. internal flag: create backing ArrayBuffer; keep in one byte

19165. additional values begin at bit 12

19166. * Constants

19167. **comment:** XXX: replace with plain Error?

label: code-design

19168. * Object finalizer

19169. Reverse operation is the same.

19170. duk_context is now defined in duk_config.h because it may also be * referenced there by prototypes.

19171. * C++ name mangling

19172. UncaughtError

19173. Ecmascript year range: * > new Date(100e6 * 24 * 3600e3).toISOString() * '+275760-09-13T00:00:00.000Z' * > new Date(-100e6 * 24 * 3600e3).toISOString() * '-271821-04-20T00:00:00.000Z'

19174. create a new global environment

19175. * Global object

19176. DUKTAPE_H_INCLUDED

19177. E == 0x7ff, F != 0 => NaN

19178. DUK_USE_PACKED_TVAL

19179. * Logging

19180. safe variants of a few coercion operations

19181. Indicates that a native function does not have a fixed number of args, * and the argument stack should not be capped/extended at all.

19182. * Get operations: no coercion, returns default value for invalid * indices and invalid value types. * * duk_get_undefined() and duk_get_null() would be pointless and * are not included.

19183. **comment:** use locale specific formatting if available

label: code-design

19184. External duk_config.h provides platform/compiler/OS dependent * typedefs and macros, and DUK_USE_xxx config options so that * the rest of Duktape doesn't need to do any feature detection.

19185. internal: don't care about fixed/dynamic nature

19186. * Property access * * The basic function assumes key is on stack. The _string variant takes * a C string as a property name, while the _index variant takes an array * index as a property name (e.g. 123 is equivalent to the key "123").

19187. * Debugging

19188. 64-bit byteswap, same operation independent of target endianness.

19189. * Type checks * * duk_is_none(), which would indicate whether index is outside of stack, * is not needed; duk_is_valid_index() gives the same information.

19190. sort array indices, use with DUK_ENUM_ARRAY_INDICES_ONLY

19191. force change if possible, may still fail for e.g. virtual properties

19192. Millisecond count constants.

19193. used by packed duk_tval, assumes sizeof(void *) == 4

19194. set value (given on value stack)

19195. Used to represent invalid index; if caller uses this without checking, * this index will map to a non-existent stack entry. Also used in some * API calls as a marker to denote "no value".

19196. * Module helpers: put multiple function or constant properties

19197. raw void pointer

19198. Flags for duk_push_thread_raw()

19199. * ===== * Duktape authors * ===== * * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang \u00f3 <llango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6tzte (<https://github.com/jaseg>) * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6man * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstrukchtrup> * * Michael Drake (<https://github.com/tlsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn \u00e1s \u00e1s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.

19200. InternalError

19201. Value types, used by e.g. duk_get_type()

19202. **comment:** Duktape specific error codes (must be 8 bits at most, see duk_error.h)

label: code-design

19203. no value, e.g. invalid index

19204. * Stack manipulation (other than push/pop)

19205. Byteswap an IEEE double in the duk_double_union from host to network * order. For a big endian target this is a no-op.

19206. prefer string
19207. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2016 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

19208. external use
19209. Flags for duk_push_string_file_raw()
19210. Duktape debug protocol version used by this build.
19211. **comment:** DUK_COMPILE_xxx bits 0-2 are reserved for an internal 'nargs' argument * (the nargs value passed is direct stack arguments + 1 to account for an * internal extra argument).
label: code-design
19212. fixed or dynamic, garbage collected byte buffer
19213. * Thread management
19214. * Double NaN manipulation macros related to NaN normalization needed when * using the packed duk_tval representation. NaN normalization is necessary * to keep double values compatible with the duk_tval format. * * When packed duk_tval is used, the NaN space is used to store pointers * and other tagged values in addition to NaNs. Actual NaNs are normalized * to a specific quiet NaN. The macros below are used by the implementation * to check and normalize NaN values when they might be created. The macros * are essentially NOPs when the non-packed duk_tval representation is used. * * A FULL check is exact and checks all bits. A NOTFULL check is used by * the packed duk_tval and works correctly for all NaNs except those that * begin with 0x7ff0. Since the 'normalized NaN' values used with packed * duk_tval begin with 0x7ff8, the partial check is reliable when packed * duk_tval is used. The 0x7ff8 prefix means the normalized NaN will be a * quiet NaN regardless of its remaining lower bits. * * The ME variant below is specifically for ARM byte order, which has the * feature that while doubles have a mixed byte order (32107654), unsigned * long long values has a little endian byte order (76543210). When writing * a logical double value through a ULL pointer, the 32-bit words need to be * swapped; hence the #ifdefs below for ULL writes with DUK_USE_DOUBLE_ME. * This is not full ARM support but suffices for some environments.
19215. either not a NaN
19216. convert time value to local time
19217. not directly applicable, byte order differs from a double
19218. end 'extern "C"' wrapper
19219. * Object operations
19220. set writable (effective if DUK_DEFPROP_HAVE_WRITABLE set)
19221. UnsupportedError
19222. **comment:** Internal API call flags, used for various functions in this file. * Certain flags are used by only certain functions, but since the flags * don't overlap, a single flags value can be passed around to multiple * functions. * * The unused top bits of the flags field are also used to pass values * to helpers (duk_get_part_helper() and duk_set_part_helper()). * * (Must be in-sync with genbuiltins.py.)
label: code-design
19223. There are currently no native functions to yield/resume, due to the internal * limitations on coroutine handling. These will be added later.
19224. Concrete macros for NaN handling used by the implementation internals. * Chosen so that they match the duk_tval representation: with a packed * duk_tval, ensure NaNs are properly normalized; with a non-packed duk_tval * these are essentially NOPs.
19225. (internal) no source string on stack
19226. * Function (method) calls
19227. plain
19228. Ecmascript string: CESU-8 / extended UTF-8 encoded
19229. internal flag: don't zero allocated buffer
19230. use strict (outer) context for global, eval, or function code
19231. in non-packed representation we don't care about which NaN is used
19232. * Union for accessing double parts, also serves as packed duk_tval
19233. include date part in string conversion result
19234. internal flag: dynamic buffer
19235. Ecmascript null
19236. enumerate non-enumerable properties in addition to enumerable
19237. **comment:** XXX: replace with RangeError?
label: code-design
19238. * Push operations * * Push functions return the absolute (relative to bottom of frame) * position of the pushed value for convenience. * * Note: duk_dup() is technically a push.
19239. string
19240. * ROM pointer compression
19241. * Union to access IEEE double memory representation, indexes for double * memory representation, and some macros for double manipulation. * * Also used by packed duk_tval. Use a union for bit manipulation to * minimize aliasing issues in practice. The C99 standard does not * guarantee that this should work, but it's a very widely supported * practice for low level manipulation. * * IEEE double format summary: * * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * See http://en.wikipedia.org/wiki/Double_precision_floating-point_format. * * NaNs are represented as exponent 0x7ff and mantissa != 0. The NaN is a * signaling NaN when the highest bit of the mantissa is zero, and a quiet * NaN when the highest bit is set. * * At least three memory layouts are relevant here: * * A B C D E F G H Big endian (e.g. 68k) DUK_USE_DOUBLE_BE * H G F E D C B A Little endian (e.g. x86) DUK_USE_DOUBLE_LE * D C B A H G F E Mixed/cross endian (e.g. ARM) DUK_USE_DOUBLE_ME * * ARM is a special case: ARM double values are in mixed/cross endian * format while ARM duk_uint64_t values are in standard little endian * format (H G F E D C B A). When a double is read as a duk_uint64_t * from memory, the register will contain the (logical) value * E F G H A B C D. This requires some special handling below. * * Indexes of various types (8-bit, 16-bit, 32-bit) in memory relative to * the logical (big endian) order: * * byte order duk_uint8_t duk_uint16_t duk_uint32_t * BE 01234567 0123 01 * LE 76543210 3210 10 * ME (ARM) 32107654 1032 01 * * Some processors may alter NaN values in a floating point load+store. * For instance, on X86 a FLD + FSTP may convert a signaling NaN to a * quiet one. This is catastrophic when NaN space is used in packed * duk_tval values. See: misc/clang_aliasing.c.
19242. * Helper macros for reading/writing memory representation parts, used * by duk_numconv.c and duk_tval.h.
19243. string conversion: use 'T' instead of '' as a separator
19244. Ecmascript number: double
19245. TypeError
19246. Support array for ROM pointer compression. Only declared when ROM * pointer compression is active.
19247. **comment:** Although extra/top could be an unsigned type here, using a signed type * makes the API more robust to calling code calculation errors or corner * cases (where caller might occasionally come up with negative values). * Negative values are treated as zero, which is better than casting them * to a large unsigned number. (This principle is used elsewhere in the * API too.)
label: code-design
19248. * Misc conversion
19249. Ecmascript object: includes objects, arrays, functions, threads
19250. URIError
19251. all doubles are considered normalized
19252. AUTHORS.rst
19253. day within month: 0 to 30

19254. setter: call is a time setter (affects hour, min, sec, ms); otherwise date setter (affects year, month, day-in-month)
19255. (internal) no filename on stack
19256. compile eval code (instead of global code)
19257. One problem with this macro is that expressions like the following fail * to compile: "(void) duk_error(...)". But because duk_error() is noreturn, * they make little sense anyway.
19258. set setter (given on value stack)
19259. prefer number, unless input is a Date, in which * case prefer string (E5 Section 8.12.8)
19260. Part indices for internal breakdowns. Part order from DUK_DATE_IDX_YEAR * to DUK_DATE_IDX_MILLISECOND matches argument ordering of EcmaScript API * calls (like Date constructor call). Some functions in duk.bi_date.c * depend on the specific ordering, so change with care. 16 bits are not * enough for all parts (year, specifically). * * (Must be in-sync with genbuiltins.py.)
19261. E == 0x7ff, F == 8 => normalized NaN
19262. Enumeration flags for duk_enum()
19263. year
19264. * Other state related functions
19265. This should not really happen, but would indicate x64.
19266. **comment:** Minimize warnings for unused internal functions with GCC >= 3.1.1 and * Clang. Based on documentation it should suffice to have the attribute * in the declaration only, but in practice some warnings are generated unless * the attribute is also applied to the definition.
 label: code-design
19267. Emscripten (provided explicitly by user), improve if possible
19268. Both MinGW and MSVC have a 64-bit type.
19269. * Platform autodetection
19270. --- TinyC ---
19271. mint clib is missing these
19272. SuperH
19273. --- x64 ---
19274. MSVC does not have sys/param.h
19275. --- Motorola 68k ---
19276. since gcc-2.5
19277. --- MIPS 64-bit ---
19278. **comment:** XXX: DUK_UNREACHABLE for msvc?
 label: code-design
19279. OpenBSD
19280. NetBSD
19281. Apple OSX, iOS
19282. Cannot determine byte order; __ORDER_PDP_ENDIAN__ is related to 32-bit * integer ordering and is not relevant.
19283. Type for public API calls.
19284. **comment:** Rely as little as possible on compiler behavior for NaN comparison, * signed zero handling, etc. Currently never activated but may be needed * for broken compilers.
 label: code-design
19285. --- Generic BSD ---
19286. uclibc may be missing these
19287. FreeBSD
19288. not defined by default
19289. **comment:** Windows, both 32-bit and 64-bit
 label: code-design
19290. --- OpenBSD ---
19291. **comment:** * Alternative customization header * * If you want to modify the final DUK_USE_xxx flags directly (without * using the available DUK_OPT_xxx flags), define DUK_OPT_HAVE_CUSTOM_H * and tweak the final flags there.
 label: code-design
19292. Cygwin
19293. integer endianness is little on purpose
19294. e.g. getdate_r
19295. * Autogenerated defaults
19296. Atari Mint
19297. **comment:** GCC older than 4.6: avoid overflow warnings related to using INFINITY
 label: code-design
19298. Most portable, wastes space
19299. GCC/clang inaccurate math would break compliance and probably duk_tval, * so refuse to compile. Relax this if -ffast-math is tested to work.
19300. --- Generic ---
19301. VS2012+ has stdint.h, < VS2012 does not (but it's available for download).
19302. **comment:** * Alignment requirement and support for unaligned accesses * * Assume unaligned accesses are not supported unless specifically allowed * in the target platform. Some platforms may support unaligned accesses * but alignment to 4 or 8 may still be desirable.
 label: requirement
19303. Clang
19304. AmigaOS. Neither AMIGA nor __amigaos__ is defined on VBCC, so user must * define 'AMIGA' manually when using VBCC.
19305. --- GCC ---
19306. Duktape/C function return value, platform int is enough for now to * represent 0, 1, or negative error code. Must be compatible with * assigning truth values (e.g. duk_ret_t rc = (foo == bar);).
19307. Low memory algorithm: separate chaining using arrays, fixed size hash
19308. __OVERRIDE_DEFINES__
19309. <http://stackoverflow.com/questions/5919996/how-to-detect-reliably-mac-os-x-ios-linux-windows-in-c-preprocessor>
19310. vbcc + AmigaOS has C99 but no inttypes.h
19311. MSVC
19312. * Check whether or not a packed duk_tval representation is possible. * What's basically required is that pointers are 32-bit values * (sizeof(void *) == 4). Best effort check, not always accurate. * If guess goes wrong, crashes may result; self tests also verify * the guess.
19313. --- Linux ---
19314. --- AmigaOS ---
19315. BCC, assume we're on x86.
19316. Pointer size determination based on __WORDSIZE or architecture when * that's not available.
19317. Unsigned index variant.
19318. External provider already defined.
19319. Byte order varies, so rely on autodetect.
19320. DUK_USE_UNION_INITIALIZERS: required from compilers, so no fill-in.
19321. **comment:** Special naming to avoid conflict with e.g. DUK_FREE() in duk_heap.h * (which is unfortunately named). May sometimes need replacement, e.g. * some compilers don't handle zero length or NULL correctly in realloc().
 label: code-design
19322. empty
19323. --- MSVC ---

19324. We're generally assuming that we're working on a platform with a 32-bit * address space. If DUK_SIZE_MAX is a typecast value (which is necessary * if SIZE_MAX is missing), the check must be avoided because the * preprocessor can't do a comparison.

19325. 64-bit constants. Since LL / ULL constants are not always available, * use computed values. These values can't be used in preprocessor * comparisons; flag them as such.

19326. --- x86 ---

19327. --- MIPS 32-bit ---

19328. XXX: add feature options to force basic types from outside?

19329. Array index values, could be exact 32 bits. * Currently no need for signed duk_arriidx_t.

19330. SIZE_MAX may be missing so use an approximate value for it.

19331. DUK_F_BCC

19332. Complex condition broken into separate parts.

19333. POSIX

19334. If not provided, use safe default for alignment.

19335. * Date provider selection * * User may define DUK_USE_DATE_GET_NOW() etc directly, in which case we'll * rely on an external provider. If this is not done, revert to previous * behavior and use Unix/Windows built-in provider.

19336. Cannot determine byte order.

19337. --- Cygwin ---

19338. !defined(DUK_USE_BYTEORDER) && defined(__BYTE_ORDER__)

19339. More or less standard endianness predefines provided by header files. * The ARM hybrid case is detected by assuming that __FLOAT_WORD_ORDER * will be big endian, see: <http://lists.mysql.com/internals/443>. * On some platforms some defines may be present with an empty value which * causes comparisons to fail: <https://github.com/svaarala/duktape/issues/453>.

19340. !defined(DUK_USE_BYTEORDER)

19341. **comment:** For custom platforms allow user to define byteorder explicitly. * Since endianness headers are not standardized, this is a useful * workaround for custom platforms for which endianness detection * is not directly supported. Perhaps custom hardware is used and * user cannot submit upstream patches.
label: code-design

19342. **comment:** Don't know how to declare unreachable point, so don't do it; this * may cause some spurious compilation warnings (e.g. "variable used * uninitialized").
label: code-design

19343. Good default is a bit arbitrary because alignment requirements * depend on target. See <https://github.com/svaarala/duktape/issues/102>.

19344. --- Windows ---

19345. User provided InitJS.

19346. These have been tested from VS2008 onwards; may work in older VS versions * too but not enabled by default.

19347. Error codes are represented with platform int. High bits are used * for flags and such, so 32 bits are needed.

19348. Intermediate define for 'have inttypes.h'

19349. build is not C99 or C++11, play it safe

19350. GCC: test not very accurate; enable only in relatively recent builds * because of bugs in gcc-4.4 (<http://lists.debian.org/debian-gcc/2010/04/msg00000.html>)

19351. Intel x86 (32-bit), x64 (64-bit) or x32 (64-bit but 32-bit pointers), * define only one of DUK_F_X86, DUK_F_X64, DUK_F_X32. *
<https://sites.google.com/site/x32abi/>

19352. Enabled with debug/assertions just so that any issues can be caught.

19353. C99 or compatible

19354. --- Mac OSX, iPhone, Darwin ---

19355. C99 or above

19356. MinGW. Also GCC flags (DUK_F_GCC) are enabled now.

19357. This detection is not very reliable.

19358. Support for 48-bit signed integer duk_tval with transparent semantics.

19359. GCC and Clang provide endianness defines as built-in predefines, with * leading and trailing double underscores (e.g. __BYTE_ORDER__). See * output of "make gcpredefs" and "make clangpredefs". Clang doesn't * seem to provide __FLOAT_WORD_ORDER__; assume not mixed endian for clang. *
<http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html>

19360. 64-bit type detection is a bit tricky. * * ULLONG_MAX is a standard define. __LONG_LONG_MAX__ and __ULONGLONG_MAX__ * are used by at least GCC (even if system headers don't provide ULLONG_MAX). * Some GCC variants may provide __LONG_LONG_MAX__ but not __ULONGLONG_MAX__. * * ULL / LL constants are rejected / warned about by some compilers, even if * the compiler has a 64-bit type and the compiler/system headers provide an * unsupported constant (ULL/LL)! Try to avoid using ULL / LL constants. * As a side effect we can only check that e.g. ULONG_MAX is larger than 32 * bits but can't be sure it is exactly 64 bits. Self tests will catch such * cases.

19361. Convert any input pointer into a 'void **', losing a const qualifier. * This is not fully portable because casting through duk_uintptr_t may * not work on all architectures (e.g. those with long, segmented pointers).

19362. Small integers (16 bits or more) can fall back to the 'int' type, but * have a typedef so they are marked "small" explicitly.

19363. **comment:** Most portable, potentially wastes space
label: code-design

19364. PowerPC

19365. --- Solaris ---

19366. **comment:** Technically C99 (C++11) but found in many systems. On some systems * __STDC_LIMIT_MACROS and __STDC_CONSTANT_MACROS must be defined before * including stdint.h (see above).
label: code-design

19367. * duk_config.h configuration header generated by genconfig.py. * * Git commit: cad34ae155acb0846545ca6bf2d29f9463b22bbb * Git describe: v1.5.2 * Git branch: HEAD * * Supported platforms: * - Mac OSX, iPhone, Darwin * - OpenBSD * - Generic BSD * - Atari ST TOS * - AmigaOS * - Windows * - Flashplayer (Crossbridge) * - QNX * - TI-Nspire * - Emscripten * - Linux * - Solaris * - Generic POSIX * - Cygwin * - Generic UNIX * - Generic fallback * * Supported architectures: * - x86 * - x64 * - x32 * - ARM 32-bit * - ARM 64-bit * - MIPS 32-bit * - MIPS 64-bit * - PowerPC 32-bit * - PowerPC 64-bit * - SPARC 32-bit * - SPARC 64-bit * - SuperH * - Motorola 68k * - Emscripten * - Generic * * Supported compilers: * - Clang * - GCC * - MSVC * - Emscripten * - TinyC * - VBCC * - Bruce's C compiler * - Generic *

19368. Missing some obvious constants.

19369. No provider for DUK_USE_DATE_PARSE_STRING(), fall back to ISO 8601 only.

19370. no endian.h or stdint.h

19371. --- Atari ST TOS ---

19372. GCC: assume we have __va_copy() in non-C99 mode.

19373. User forced alignment to 4 or 8.

19374. C99 / C++11 and above: rely on va_copy() which is required.

19375. <http://bellard.org/tcc/tcc-doc.html#SEC9>

19376. Check that architecture is two's complement, standard C allows e.g. * INT_MIN to be -2**31+1 (instead of -2**31).

19377. Based on 'make checkalign' there are no alignment requirements on * Linux MIPS except for doubles, which need align by 4. Alignment * requirements vary based on target though.

19378. AmigaOS on M68k

19379. not configured for DLL build

19380. * You may add overriding #define/#undef directives below for * customization. You of course cannot un-#include or un-typedef * anything; these require direct changes above.

19381. **comment:** cannot detect 64-bit type, not always needed so don't error
label: code-design

19382. --- Generic UNIX ---

19383. std::exception

19384. A few types are assumed to always exist.

19385. Float word order not known, assume not a hybrid.
19386. --- Generic fallback ---
19387. **comment:** --- Generic POSIX ---
 label: code-design
19388. no parsing (not an error)
19389. **comment:** ANSI C (various versions) and some implementations require that the * pointer arguments to memset(), memcpy(), and memmove() be valid values * even when byte size is 0 (even a NULL pointer is considered invalid in * this context). Zero-size operations as such are allowed, as long as their * pointer arguments point to a valid memory area. The DUK_MEMSET(), * DUK_MEMCPY(), and DUK_MEMMOVE() macros require this same behavior, i.e.: * (1) pointers must be valid and non-NULL, (2) zero size must otherwise be * allowed. If these are not fulfilled, a macro wrapper is needed. ** http://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0 * http://lists.cs.uiuc.edu/pipermail/lvmddev/2007-October/011065.html ** Not sure what's the required behavior when a pointer points just past the * end of a buffer, which often happens in practice (e.g. zero size memmoves). * For example, if allocation size is 3, the following pointer would not * technically point to a valid memory byte: * * <-- alloc --> * | 0 | 1 | 2 | * ^ - p=3, points after last valid byte (2)
 label: code-design
19390. C++11 or above
19391. SPARC byte order varies so rely on autodetection.
19392. DUK_F_PACKED_TVAL_PROVIDED
19393. http://www.monkey.org/openbsd/archive/ports/0401/msg00089.html
19394. VBCC
19395. sigsetjmp() alternative
19396. GCC. Clang also defines __GNUC__ so don't detect GCC if using Clang.
19397. --- SuperH ---
19398. Pre-C99: va_list type is implementation dependent. This replacement * assumes it is a plain value so that a simple assignment will work. * This is not the case on all platforms (it may be a single-array element, * for instance).
19399. **comment:** The best type for an "all around int" in Duktape internals is "at least * 32 bit signed integer" which is most convenient. Same for unsigned type. * Prefer 'int' when large enough, as it is almost always a convenient type.
 label: code-design
19400. uclibc
19401. Explicit marker needed; may be 'defined', 'undefined', 'or 'not provided'.
19402. No provider for DUK_USE_DATE_FORMAT_STRING(), fall back to ISO 8601 only.
19403. **comment:** Many platforms are missing fpclassify() and friends, so use replacements * if necessary. The replacement constants (FP_NAN etc) can be anything but * match Linux constants now.
 label: code-design
19404. * Checks for config option consistency (DUK_USE_xxx)
19405. autodetect compiler
19406. Rely on C89 headers only; time.h must be here.
19407. These functions don't currently need replacement but are wrapped for * completeness. Because these are used as function pointers, they need * to be defined as concrete C functions (not macros).
19408. --- TI-Nspire ---
19409. **comment:** We need va_copy() which is defined in C99 / C++11, so an awkward * replacement is needed for pre-C99 / pre-C++11 environments. This * will quite likely need portability hacks for some non-C99 * environments.
 label: code-design
19410. AmigaOS on M68K or PPC is always big endian.
19411. Feature option forcing.
19412. --- ARM 64-bit ---
19413. * Feature option handling
19414. **comment:** * Wrapper typedefs and constants for integer types, also sanity check types. * * C99 typedefs are quite good but not always available, and we want to avoid * forcibly redefining the C99 typedefs. So, there are Duktape wrappers for * all C99 typedefs and Duktape code should only use these typedefs. Type * detection when C99 is not supported is best effort and may end up detecting * some types incorrectly. ** Pointer sizes are a portability problem: pointers to different types may * have a different size and function pointers are very difficult to manage * portably. ** http://en.wikipedia.org/wiki/C_data_types#Fixed-width_integer_types ** Note: there's an interesting corner case when trying to define minimum * signed integer value constants which leads to the current workaround of * defining e.g. -0x80000000 as (-0x7fffffffL - 1L). See doc/code-issues.txt * for a longer discussion. ** Note: avoid typecasts and computations in macro integer constants as they * can then no longer be used in macro relational expressions (such as * #if DUK_SIZE_MAX < 0xffffffffUL). There is internal code which relies on * being able to compare DUK_SIZE_MAX against a limit.
 label: code-design
19415. Not standard but common enough
19416. --- QNX ---
19417. **comment:** Object property allocation layout has implications for memory and code * footprint and generated code size/speed. The best layout also depends * on whether the platform has alignment requirements or benefits from * having mostly aligned accesses.
 label: code-design
19418. (v)snprintf() is missing before MSVC 2015. Note that -(v)snprintf() does * NOT NUL terminate on truncation, but Duktape code never assumes that. * http://stackoverflow.com/questions/2915672/snprintf-and-visual-studio-2010
19419. Motorola 68K. Not defined by VBCC, so user must define one of these * manually when using VBCC.
19420. http://msdn.microsoft.com/en-us/library/aa235362(VS.60).aspx
19421. Non-C99 case, still relying on DUK_UINTPTR_MAX, as long as it is not a computed value
19422. http://bellard.org/tcc/tcc-doc.html#SEC7
19423. * Intermediate helper defines
19424. **comment:** Macro hackery to convert e.g. __LINE__ to a string without formatting, * see: http://stackoverflow.com/questions/240353/convert-a-preprocessor-token-to-a-string
 label: code-design
19425. Flash player (e.g. Crossbridge)
19426. no endian.h
19427. Shared includes: C89
19428. integer byte order
19429. float word order
19430. * Convert DUK_USE_BYTEORDER, from whatever source, into currently used * internal defines. If detection failed, #error out.
19431. e.g. ptrdiff_t
19432. **comment:** XXX: DUK_NOINLINE, DUK_INLINE, DUK_ALWAYS_INLINE for msvc?
 label: code-design
19433. SPARC
19434. VBCC supports C99 so check only for C99 for union initializer support. * Designated union initializers would possibly work even without a C99 check.
19435. C++11 apparently ratified stdint.h
19436. QNX gcc cross compiler seems to define e.g. __LITTLEENDIAN__ or __BIGENDIAN__: * \$ /opt/qnx650/host/linux/x86/usr/bin/i486-pc-nto-qnx6.5.0-gcc -dM -E - </dev/null | grep -ni endian * 67:#define __LITTLEENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/mips-unknown-nto-qnx6.5.0-gcc -dM -E - </dev/null | grep -ni endian * 81:#define __BIGENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/arm-unknown-nto-qnx6.5.0-gcc -dM -E - </dev/null | grep -ni endian * 70:#define __LITTLEENDIAN__ 1
19437. At least some uclibc versions have broken floating point math. For * example, fpclassify() can incorrectly classify certain NaN formats. * To be safe, use replacements.

19438. Placeholder fix for (detection is wider than necessary): * http://llvm.org/bugs/show_bug.cgi?id=17788
19439. snprintf() is technically not part of C89 but usually available.
19440. --- x32 ---
19441. Basic integer typedefs and limits, preferably from inttypes.h, otherwise * through automatic detection.
19442. **comment:** Only include when compiling Duktape to avoid polluting application build * with a lot of unnecessary defines.
 label: code-design
19443. --- VBCC ---
19444. On some systems SIZE_MAX can be smaller than max unsigned 32-bit value * which seems incorrect if size_t is (at least) an unsigned 32-bit type. * However, it doesn't seem useful to error out compilation if this is the * case.
19445. **comment:** On other platforms use layout 2, which requires some padding but * is a bit more natural than layout 3 in ordering the entries. Layout * 3 is currently not used.
 label: code-design
19446. VS2005+ should have variadic macros even when they're not C99.
19447. illumos / Solaris
19448. DUK_COMPILING_DUKTAPE
19449. DUK_OPT_FORCE_BYTEORDER
19450. QNX
19451. since gcc-4.5
19452. **comment:** Windows 32-bit and 64-bit are currently the same.
 label: code-design
19453. This is optionally used by panic handling to cause the program to segfault * (instead of e.g. abort()) on panic. Valgrind will then indicate the C * call stack leading to the panic.
19454. IEEE float/double typedef.
19455. --- ARM 32-bit ---
19456. **comment:** XXX: This is technically not guaranteed because it's possible to configure * an x86 to require aligned accesses with Alignment Check (AC) flag.
 label: code-design
19457. TinyC
19458. **comment:** Note: the funny looking computations for signed minimum 16-bit, 32-bit, and * 64-bit values are intentional as the obvious forms (e.g. -0x80000000L) are * -not- portable. See code-issues.txt for a detailed discussion.
 label: code-design
19459. These are necessary wild guesses.
19460. Already provided above
19461. Atari ST TOS. __TOS__ defined by PureC. No platform define in VBCC * apparently, so to use with VBCC user must define __TOS__ manually.
19462. Byte order is big endian but cannot determine IEEE double word order.
19463. --- Flashplayer (Crossbridge) ---
19464. MIPS byte order varies so rely on autodetection.
19465. stdint.h not available
19466. **comment:** The most portable way to figure out local time offset is gmtime(), * but it's not thread safe so use with caution.
 label: requirement
19467. Codepoint type. Must be 32 bits or more because it is used also for * internal codepoints. The type is signed because negative codepoints * are used as internal markers (e.g. to mark EOF or missing argument). * (X)UTF-8/CESU-8 encode/decode take and return an unsigned variant to * ensure duk_uint32_t casts back and forth nicely. Almost everything * else uses the signed one.
19468. DUK_USE_VARIADIC_MACROS: required from compilers, so no fill-in.
19469. TOS on M68K is always big endian.
19470. don't use strftime() for now
19471. **comment:** not sure, not needed with C99 anyway
 label: requirement
19472. C++ doesn't have standard designated union initializers ({ .foo = 1 }).
19473. --- Emscripten ---
19474. nop
19475. DLL build detection
19476. **comment:** Avoid custom date parsing and formatting for portability.
 label: code-design
19477. Workaround for older C++ compilers before including <inttypes.h>, * see e.g.: https://sourceware.org/bugzilla/show_bug.cgi?id=15366
19478. * Compiler autodetection
19479. --- SPARC 32-bit ---
19480. --- Clang ---
19481. On some platforms int is 16-bit but long is 32-bit (e.g. PureC)
19482. **comment:** * Byte order and double memory layout detection ** Endianness detection is a major portability hassle because the macros * and headers are not standardized. There's even variance across UNIX * platforms. Even with "standard" headers, details like underscore count * varies between platforms, e.g. both __BYTE_ORDER and _BYTE_ORDER are used * (Crossbridge has a single underscore, for instance). ** The checks below are structured with this in mind: several approaches are * used, and at the end we check if any of them worked. This allows generic * approaches to be tried first, and platform/compiler specific hacks tried * last. As a last resort, the user can force a specific endianness, as it's * not likely that automatic detection will work on the most exotic platforms. ** Duktape supports little and big endian machines. There's also support * for a hybrid used by some ARM machines where integers are little endian * but IEEE double values use a mixed order (12345678 -> 43218765). This * byte order for doubles is referred to as "mixed endian".
 label: code-design
19483. **comment:** Check whether we should use 64-bit integers or not. ** Quite incomplete now. Use 64-bit types if detected (C99 or other detection) * unless they are known to be unreliable. For instance, 64-bit types are * available on VBCC but seem to misbehave.
 label: code-design
19484. VS2013+ supports union initializers but there's a bug involving union-inside-struct: * <https://connect.microsoft.com/VisualStudio/feedback/details/805981> * The bug was fixed (at least) in VS2015 so check for VS2015 for now: * <https://blogs.msdn.microsoft.com/vcblog/2015/07/01/c-compiler-front-end-fixes-in-vs2015/> * Manually tested using VS2013, CL reports 18.00.31101, so enable for VS2013 too.
19485. no user declarations
19486. DUK_CONFIG_H_INCLUDED
19487. BCC (Bruce's C compiler): this is a "torture target" for compilation
19488. Generic Unix (includes Cygwin)
19489. There was a curious bug where test-bi-date-canceling.js would fail e.g. * on 64-bit Ubuntu, gcc-4.8.1, -m32, and no -std=c99. Some date computations * using doubles would be optimized which then broke some corner case tests. * The problem goes away by adding 'volatile' to the datetime computations. * Not sure what the actual triggering conditions are, but using this on * non-C99 systems solves the known issues and has relatively little cost * on other platforms.
19490. C99 / C++11 and above: rely on va_copy() which is required. * Omit parenthesis on macro right side on purpose to minimize differences * to direct use.
19491. C99 types
19492. DUK_SIZE_MAX (= SIZE_MAX) is often reliable
19493. --- PowerPC 64-bit ---
19494. See: /opt/qnx650/target/qnx6/usr/include/sys/platform.h
19495. --- SPARC 64-bit ---
19496. On platforms without any alignment issues, layout 1 is preferable * because it compiles to slightly less code and provides direct access * to property keys.
19497. Strict C99 case: DUK_UINTPTR_MAX (= UINTPTR_MAX) should be very reliable
19498. autodetect platform

19499. **comment:** NetBSD 6.0 x86 (at least) has a few problems with pow() semantics, * see test-bug-netbsd-math-pow.js. Use NetBSD specific workaround. * (This might be a wider problem; if so, generalize the define name.)
label: code-design
19500. autodetect architecture
19501. In VBCC (0.0 / 0.0) results in a warning and 0.0 instead of NaN. * In MSVC (VS2010 Express) (0.0 / 0.0) results in a compile error. * Use a computed NaN (initialized when a heap is created at the * latest).
19502. Clang: assume we have __va_copy() in non-C99 mode.
19503. --- Bruce's C compiler ---
19504. Same as 'duk_int_t' but guaranteed to be a 'fast' variant if this * distinction matters for the CPU. These types are used mainly in the * executor where it might really matter.
19505. vbcc + AmigaOS may be missing these
19506. C99 or C++11, no known issues
19507. **comment:** XXX: DUK_LIKELY, DUK_UNLIKELY for msvc?
label: code-design
19508. Initial fix: disable secure CRT related warnings when compiling Duktape * itself (must be defined before including Windows headers). Don't define * for user code including duktape.h.
19509. varargs
19510. * Fill-ins for platform, architecture, and compiler
19511. e.g. strptime
19512. **comment:** Note: PRS and FMT are intentionally left undefined for now. This means * there is no platform specific date parsing/formatting but there is still * the ISO 8601 standard format.
label: code-design
19513. vsnprintf() is technically not part of C89 but usually available.
19514. Rely on autodetection for byte order, alignment, and packed tval.
19515. AmigaOS + M68K seems to have math issues even when using GCC cross * compilation. Use replacements for all AmigaOS versions on M68K * regardless of compiler.
19516. Byte order is little endian but cannot determine IEEE double word order.
19517. C++
19518. MIPS. Related defines: __MIPSEB__, __MIPSEL__, __mips_isa_rev, __LP64__
19519. VBCC is missing the built-ins even in C99 mode (perhaps a header issue).
19520. __MSC_VER
19521. **comment:** Macro for suppressing warnings for potentially unreferenced variables. * The variables can be actually unreferenced or unreferenced in some * specific cases only; for instance, if a variable is only debug printed, * it is unreferenced when debug printing is disabled.
label: code-design
19522. Byte order varies, rely on autodetection.
19523. Use __setjmp() on Apple by default, see GH-55.
19524. TI-Nspire (using Ndless)
19525. --- PowerPC 32-bit ---
19526. Some math functions are C99 only. This is also an issue with some * embedded environments using uclibc where uclibc has been configured * not to provide some functions. For now, use replacements whenever * using uclibc.
19527. Most portable
19528. Boolean values are represented with the platform 'int'.
19529. Shared includes: stdint.h is C99
19530. BSD variant
19531. Old uclibcs have a broken memcpy so use memmove instead (this is overly wide * now on purpose): <http://lists.uclibc.org/pipermail/uclibc-cvs/2008-October/025511.html>
19532. date.h is omitted, and included per platform
19533. * Architecture autodetection
19534. The most portable current time provider is time(), but it only has a * one second resolution.
19535. Convenience, e.g. gcc 4.5.1 == 40501; <http://stackoverflow.com/questions/6031819/emulating-gccs-built-in-unreachable>
19536. defined(DUK_USE_BYTEORDER)
19537. In VBCC (1.0 / 0.0) results in a warning and 0.0 instead of infinity. * Use a computed infinity (initialized when a heap is created at the * latest).
19538. For now, hash part is dropped if and only if 16-bit object fields are used.
19539. MSVC dllexport/dllimport: appropriate __declspec depends on whether we're * compiling Duktape or the application.
19540. **comment:** On Windows, assume we're little endian. Even Itanium which has a * configurable endianness runs little endian in Windows.
label: code-design
19541. **comment:** Compiler specific hackery needed to force struct size to match alignment, * see e.g. duk_hbuffer.h. * * <http://stackoverflow.com/questions/11130109/c-struct-size-alignment>
label: code-design
19542. Index values must have at least 32-bit signed range.
19543. **comment:** Workaround for GH-323: avoid inlining control when compiling from * multiple sources, as it causes compiler portability trouble.
label: code-design
19544. byte order
19545. Linux
19546. Based on 'make checkalign' there are no alignment requirements on * Linux SH4, but align by 4 is probably a good basic default.
19547. MSVC preprocessor defines: <http://msdn.microsoft.com/en-us/library/b0084kay.aspx> * __MSC_FULL_VER includes the build number, but it has at least two formats, see e.g. * BOOST_MSVC_FULL_VER in http://www.boost.org/doc/libs/1_52_0/boost/config/compiler/visualc.hpp
19548. ARM
19549. Fast variants of small integers, again for really fast paths like the * executor.
19550. **comment:** When C99 types are not available, we use heuristic detection to get * the basic 8, 16, 32, and (possibly) 64 bit types. The fast/least * types are then assumed to be exactly the same for now: these could * be improved per platform but C99 types are very often now available. * 64-bit types are not available on all platforms; this is OK at least * on 32-bit platforms. * * This detection code is necessarily a bit hacky and can provide typedefs * and defines that won't work correctly on some exotic platform.
label: code-design
19551. **comment:** * Cleanups (all statement parsing flows through here). * * Pop label site and reset labels. Reset 'next temp' to value at * entry to reuse temps.
label: code-design
19552. **comment:** * Reference counting helper macros. The macros take a thread argument * and must thus always be executed in a specific thread context. The * thread argument is needed for features like finalization. Currently * it is not required for INCREF, but it is included just in case. * * Note that 'raw' macros such as DUK_HEAPHDR_GET_REFCOUNT() are not * defined without DUK_USE_REFERENCE_COUNTING, so caller must #ifdef * around them.
label: code-design
19553. alloc external with size zero
19554. i' is the first entry we'll keep
19555. If blen <= 0xffffUL, clen is also guaranteed to be <= 0xffffUL.
19556. Set catcher regs: idx_base+0 = value, idx_base+1 = lj_type.
19557. resend state next time executor is about to run
19558. **comment:** XXX: avoid this check somehow
label: code-design
19559. Should match Function.prototype.toString()

19560. * Main struct
19561. '/
19562. * Unwind debugger state. If we unwind while stepping * (either step over or step into), pause execution.
19563. Duktape.modLoaded[] module cache
19564. [obj handler trap]
19565. Not safe to use 'reg_varbind' as assignment expression * value, so go through a temp.
19566. **comment:** size for formatting buffers
 label: code-design
19567. default case exists: go there if no case matches
19568. registers are mutable, non-deletable
19569. '{
19570. If the debugger is active we need to force an interrupt so that * debugger breakpoints are rechecked. This is important for function * calls caused by side effects (e.g. when doing a DUK_OP_GETPROP), see * GH-303. Only needed for success path, error path always causes a * breakpoint recheck in the executor. It would be enough to set this * only when returning to an Ecmascript activation, but setting the flag * on every return should have no ill effect.
19571. get or set a range of flags; m=first bit number, n=number of bits
19572. [level obj func pc line]
19573. * Ecmascript compliant [[GetProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * May cause arbitrary side effects and invalidate (most) duk_tval * pointers.
19574. [arg1 ... argN this]
19575. * Matching order: * * Punctuator first chars, also covers comments, regexps * LineTerminator * Identifier or reserved word, also covers null/true/false literals * NumericLiteral * StringLiteral * EOF * * The order does not matter as long as the longest match is * always correctly identified. There are order dependencies * in the clauses, so it's not trivial to convert to a switch.
19576. Allocator functions.
19577. next does not exist or next is not a letter
19578. * Create and push an error object onto the top of stack. * If a "double error" occurs, use a fixed error instance * to avoid further trouble.
19579. 'enum'
19580. SameValue(NaN, NaN) = true, regardless of NaN sign or extra bits
19581. just making sure
19582. needed for inf: causes mantissa to become zero, * and rounding to be skipped.
19583. **comment:** XXX: string not shared because it is conditional
 label: code-design
19584. carry
19585. Initial bytes for markers.
19586. queue back to heap_allocated
19587. XXX: We can't resize the value stack to a size smaller than the * current top, so the order of the resize and adjusting the stack * top depends on the current vs. final size of the value stack. * The operations could be combined to avoid this, but the proper * fix is to only grow the value stack on a function call, and only * shrink it (without throwing if the shrink fails) on function * return.
19588. print provider
19589. form: { DecimalDigits , }, val1 = min count
19590. * Figure out the final, non-bound constructor, to get "prototype" * property.
19591. **comment:** * Note: * * - duk__match_regexp() is required not to longjmp() in ordinary "non-match" * conditions; a longjmp() will terminate the entire matching process. * * - Clearing saved[] is not necessary because backtracking does it * * - Backtracking also rewinds ctx.recursion back to zero, unless an * internal/limit error occurs (which causes a longjmp()) * * - If we supported anchored matches, we would break out here * unconditionally; however, Ecmascript regexps don't have anchored * matches. It might make sense to implement a fast bail-out if * the regexp begins with '^' and sp is not 0: currently we'll just * run through the entire input string, trivially failing the match * at every non-zero offset.
 label: code-design
19592. DUK_TOK_LPAREN
19593. caller ensures; rom objects are never bufferobjects now
19594. Would be nice to bulk clear the allocation, but the context * is 1-2 kilobytes and nothing should rely on it being zeroed.
19595. 1 -> needs a PUTVAR
19596. * Post-increment/decrement will return the original value as its * result value. However, even that value will be coerced using * ToNumber() which is quite awkward. Specific bytecode opcodes * are used to handle these semantics. * * Note that post increment/decrement has a "no LineTerminator here" * restriction. This is handled by duk__expr_lbp(), which forcibly terminates * the previous expression if a LineTerminator occurs before '++'/'--'.
19597. leave out trailing 'unused' elements
19598. **comment:** TODO: implement Proxy object support here
 label: requirement
19599. * Multiply + add + carry for 32-bit components using only 16x16->32 * multiples and carry detection based on unsigned overflow. * * 1st mult, 32-bit: (A*2^16 + B) * 2nd mult, 32-bit: (C*2^16 + D) * 3rd add, 32-bit: E * 4th add, 32-bit: F * * (AC*2^16 + B) * (C*2^16 + D) + E + F * = AC*2^32 + AD*2^16 + BC*2^16 + BD + E + F * = AC*2^32 + (AD + BC)*2^16 + (BD + E + F) * = AC*2^32 + AD*2^16 + BC*2^16 + (BD + E + F)
19600. args start at index 2
19601. * Various Unicode help functions for character classification predicates, * case conversion, decoding, etc.
19602. eat colon
19603. always set; points to a reserved valstack slot
19604. Track number of escapes: necessary for proper keyword * detection.
19605. * Node.js Buffer.prototype.write(string, [offset], [length], [encoding])
19606. IdentifierPart production with IdentifierStart, ASCII, and non-BMP excluded
19607. should never happen but default here
19608. 'env'
19609. Preliminaries, required by setjmp() handler. Must be careful not * to throw an unintended error here.
19610. -> [val]
19611. emitted
19612. resume delete to target
19613. -> [... obj]
19614. success
19615. +0.0
19616. replacer is a mutation risk
19617. parse key and value
19618. **comment:** * Arithmetic operations other than '+' have number-only semantics * and are implemented here. The separate switch-case here means a * "double dispatch" of the arithmetic opcode, but saves code space. * * E5 Sections 11.5, 11.5.1, 11.5.2, 11.5.3, 11.6, 11.6.1, 11.6.2, 11.6.3.
 label: code-design
19619. No duk_bw_remove_ensure_slice(), functionality would be identical.
19620. if boolean matches A, skip next inst
19621. ($\geq e 0$) AND ($= f (\text{expt } b (- p 1))$) * * be <- (expt b e) == b^e * be1 <- (* be b) == (expt b (+ e 1)) == b^(e+1) * r <- (* f be1 2) == 2 * f * b^(e+1) [if b==2 -> f * b^(e+2)] * s <- (* b 2) [if b==2 -> 4] * m+ <- be1 == b^(e+1) * m- <- be == b^e * k <- 0 * B <- B * low_ok <- round * high_ok <- round
19622. Perform fixed-format rounding.

19623. If not within Ecmascript range, some integer time calculations * won't work correctly (and some asserts will fail), so bail out * if so. This fixes test-bug-date-insane-setyear.js. There is * a +/- 24h leeway in this range check to avoid a test262 corner * case documented in test-bug-date-timeval-edges.js.
19624. Failed to match the quantifier, restore lexer and parse * opening brace as a literal.
19625. [...] val]
19626. 'true'
19627. **comment:** If we don't have a jmpbuf_ptr, there is little we can do * except panic. The caller's expectation is that we never * return. ** With C++ exceptions we now just propagate an uncaught error * instead of invoking the fatal error handler. Because there's * a dummy jmpbuf for C++ exceptions now, this could be changed.
label: code-design
19628. For now shared handler is fine.
19629. may be less, since DELETED entries are NULLed by rehash
19630. Handle single character fills as memset() even when * the fill data comes from a one-char argument.
19631. OK
19632. **comment:** * Bytecode dump/load ** The bytecode load primitive is more important performance-wise than the * dump primitive. ** Unlike most Duktape API calls, bytecode dump/load is not guaranteed to be * memory safe for invalid arguments - caller beware! There's little point * in trying to achieve memory safety unless bytecode instructions are also * validated which is not easy to do with indirect register references etc.
label: code-design
19633. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
19634. maximum loopcount for peephole optimization
19635. cleared before entering finally
19636. flags for duk_hobject_get_own_propdesc() and variants
19637. **comment:** Execute finalizers before freeing the heap, even for reachable * objects, and regardless of whether or not mark-and-sweep is * enabled. This gives finalizers the chance to free any native * resources like file handles, allocations made outside Duktape, * etc. This is quite tricky to get right, so that all finalizer * guarantees are honored. ** XXX: this perhaps requires an execution time limit.
label: code-design
19638. useful for patching jumps later
19639. avoid attempts to add/remove object keys
19640. The get/set pointers could be 16-bit pointer compressed but it * would make no difference on 32-bit platforms because duk_tval is * 8 bytes or more anyway.
19641. * Local defines
19642. **comment:** XXX: compression
label: code-design
19643. already NULLed (by unwind)
19644. Don't allow actual chars after equal sign.
19645. -> [Object defineProperty undefined obj key desc]
19646. whether to use macros or helper function depends on call count
19647. DUK_USE_FASTINT
19648. 7: toISOString
19649. **comment:** XXX: "read only object"?
label: code-design
19650. no action
19651. **comment:** XXX: There is currently no support for writing buffer object * indexed elements here. Attempt to do so will succeed and * write a concrete property into the buffer object. This should * be fixed at some point but because buffers are a custom feature * anyway, this is relatively unimportant.
label: code-design
19652. must hold DUK_VARARGS
19653. capture is 'undefined', always matches!
19654. **comment:** * XXX: could make this a lot faster if we create the double memory * representation directly. Feasible easily (must be uniform random).
label: code-design
19655. 'raw'
19656. DUK_TOK_RPAREN
19657. request to create catch binding
19658. for object-bound identifiers
19659. **comment:** XXX: compression (as an option)
label: code-design
19660. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** This handling is now identical for C and Ecmascript functions. * C functions always have the 'NEWENV' flag set, so their * environment record initialization is delayed (which is good). ** Delayed creation (on demand) is handled in duk_js_var.c.
19661. **comment:** * Declaration binding instantiation conceptually happens when calling a * function; for us it essentially means that function prologue. The * conceptual process is described in E5 Section 10.5. ** We need to keep track of all encountered identifiers to (1) create an * identifier-to-register map ("varmap"); and (2) detect duplicate * declarations. Identifiers which are not bound to registers still need * to be tracked for detecting duplicates. Currently such identifiers * are put into the varmap with a 'null' value, which is later cleaned up. ** To support functions with a large number of variable and function * declarations, registers are not allocated beyond a certain limit; * after that limit, variables and functions need slow path access. * Arguments are currently always register bound, which imposes a hard * (and relatively small) argument count limit. ** Some bindings in E5 are not configurable (= deletable) and almost all * are mutable (writable). Exceptions are: ** - The 'arguments' binding, established only if no shadowing argument * or function declaration exists. We handle 'arguments' creation * and binding through an explicit slow path environment record. ** - The "name" binding for a named function expression. This is also * handled through an explicit slow path environment record.
label: code-design
19662. first coerce to a plain value
19663. -> [... func varmap enum]
19664. * Replace lexical environment for global scope ** Create a new object environment for the global lexical scope. * We can't just reset the _Target property of the current one, * because the lexical scope is shared by other threads with the * same (initial) built-ins.
19665. force_global
19666. NB: use 's' as temp on purpose
19667. Not odd, or y == -Infinity
19668. * Global object built-ins
19669. A bunch of helpers (for size optimization) that combine duk_expr()/duk_exprtop() * and result conversions. ** Each helper needs at least 2-3 calls to make it worth while to wrap.
19670. Value stack: these are expressed as pointers for faster stack manipulation. * [valstack, valstack_top[is GC-reachable, [valstack_top, valstack_end[is * not GC-reachable but kept initialized as 'undefined'.
19671. Output a specified number of digits instead of using the shortest * form. Used for toPrecision() andtoFixed().
19672. [body formals], formals is comma separated list that needs to be parsed
19673. bytecode execution
19674. thisArg
19675. * Not found
19676. XXX: duk_uarridx_t?
19677. E5 Section 9.4, ToInteger()
19678. force_new
19679. duk_js_call.c is required to restore the stack reserve * so we only need to reset the top.

19680. * __FILE__ / __LINE__ entry, here 'pc' is line number directly. * Sometimes __FILE__ / __LINE__ is reported as the source for * the error (fileName, lineNumber), sometimes not.

19681. avoid dereference after potential callstack realloc

19682. * Pass 2

19683. compensate for eval() call

19684. set values into ret array

19685. * Pick an object from the head (don't remove yet).

19686. fast path

19687. **comment:** Fast exit if indices are identical. This is valid for a non-existent property, * for an undefined value, and almost always for ToString() coerced comparison of * arbitrary values (corner cases where this is not the case include e.g. a an * object with varying ToString() coercion). ** The specification does not prohibit "caching" of values read from the array, so * assuming equality for comparing an index with itself falls into the category of * "caching". ** Also, compareFn may be inconsistent, so skipping a call to compareFn here may * have an effect on the final result. The specification does not require any * specific behavior for inconsistent compare functions, so again, this fast path * is OK.

label: code-design

19688. **comment:** XXX: this could be a DUK__CONSTP instead

label: code-design

19689. [... this tracedata sep this]

19690. * After arguments, allocate special registers (like shuffling temps)

19691. [ToObject(this) item1 ... itemN arr item(i)]

19692. The implementation for computing of start_pos and end_pos differs * from the standard algorithm, but is intended to result in the exactly * same behavior. This is not always obvious.

19693. This behavior mostly mimics Node.js now.

19694. **comment:** XXX: typing (duk_hcompiledfunction has duk_uint32_t)

label: code-design

19695. **comment:** XXX: because we're dealing with 'own' properties of a fresh array, * the array initializer should just ensure that the array has a large * enough array part and write the values directly into array part, * and finally set 'length' manually in the end (as already happens now).

label: code-design

19696. This may happen even after the fast path check, if exponent is * not balanced (e.g. "0e1"). Remember to respect zero sign.

19697. Encode a fastint from duk_tval ptr, no value stack effects.

19698. 'has'

19699. DUK_TOK_COMMA

19700. 'void'

19701. bufwriter for code

19702. * Abandon array failed, need to decref keys already inserted * into the beginning of new_e_k before unwinding valstack.

19703. * UTF-8 / XUTF-8 / CESU-8 constants

19704. when going backwards, we decrement cpos 'early'; * 'p' may point to a continuation byte of the char * at offset 'cpos', but that's OK because we'll * backtrack all the way to the initial byte.

19705. DUK_TOK_IDENTIFIER

19706. * Regexp recursive matching function. ** Returns 'sp' on successful match (points to character after last matched one), * NULL otherwise. ** The C recursion depth limit check is only performed in this function, this * suffices because the function is present in all true recursion required by * regexp execution.

19707. **comment:** Use recursion_limit to ensure we don't overwrite js_ctx->visiting[] * array so we don't need two counter checks in the fast path. The * slow path has a much larger recursion limit which we'll use if * necessary.

label: code-design

19708. [... func varmap enum key value this]

19709. 1: toDateString

19710. **comment:** Fill offset handling is more lenient than in Node.js.

label: code-design

19711. Positive if local time ahead of UTC.

19712. With this check in place fast paths won't need read-only * object checks. This is technically incorrect if there are * setters that cause no writes to ROM objects, but current * built-ins don't have such setters.

19713. save stack top

19714. Insert an empty jump in the middle of code emitted earlier. This is * currently needed for compiling for-in.

19715. Treat like a debugger statement: ignore when not attached.

19716. 'import'

19717. Duktape/C API guaranteed entries (on top of args)

19718. avoid warning (unsigned)

19719. Helper which can be called both directly and with duk_safe_call().

19720. [... template]

19721. -> [buffer]

19722. **comment:** XXX: Here a "slice copy" would be useful.

label: code-design

19723. Must ensure result is 64-bit (no overflow); a * simple and sufficient fast path is to allow only * 32-bit inputs. Avoid zero inputs to avoid * negative zero issues (-1 * 0 = -0, for instance).

19724. duk_unicode_ids_m_let_noa[]

19725. Don't intercept a DoubleError, we may have caused the initial double * fault and attempting to intercept it will cause us to be called * recursively and exhaust the C stack.

19726. **comment:** * Node.js Buffer.prototype.equals() * Node.js Buffer.prototype.compare() * Node.js Buffer.compare()

label: requirement

19727. indirect allocs

19728. Lightfuncs are always considered strict.

19729. implies DUK_HOBJECT_IS_BUFFEROBJECT

19730. safe, because matched (NUL causes a break)

19731. * Packed tval sanity

19732. longjmp type

19733. should be first on 64-bit platforms

19734. **comment:** * XXX: duk_uint_fast32_t should probably be used in many places here.

label: code-design

19735. [... | (crud) errobj]

19736. expensive flag

19737. validation performed by duk_hexval

19738. non-strict equality from here on

19739. allocate catcher and populate it (should be atomic)

19740. since duk_abandon_array_checked() causes a resize, there should be no gaps in keys

19741. **comment:** DUK_TVAL_SET_TVAL_UPDREF() is used a lot in executor, property lookups, * etc, so it's very important for performance. Measure when changing. ** NOTE: the source and destination duk_tval pointers may be the same, and * the macros MUST deal with that correctly.

label: code-design

19742. if 0, 'str16' used, if > 0, 'strlist16' used

19743. probe sequence (open addressing)

19744. token was preceded by a lineterm
19745. -> [...]
19746. * Store entry state.
19747. Perform an intermediate join when this many elements have been pushed * on the value stack.
19748. duk_unicode_ids_m_let_noabmp[]
19749. * Parse variant 3 or 4. ** For variant 3 (e.g. "for (A in C) D;") the code for A (except the * final property/variable write) has already been emitted. The first * instruction of that code is at pc_v34_lhs; a JUMP needs to be inserted * there to satisfy control flow needs. ** For variant 4, if the variable declaration had an initializer * (e.g. "for (var A = B in C) D;") the code for the assignment * (B) has already been emitted. ** Variables set before entering here: ** pc_v34_lhs: insert a "JUMP L2" here (see doc/compiler.rst example). * reg_temps + 0: iteration target value (written to LHS) * reg_temps + 1: enumerator object
19750. 'return'
19751. **comment:** * String table algorithm: fixed size string table with array chaining ** The top level string table has a fixed size, with each slot holding * either NULL, string pointer, or pointer to a separately allocated * string pointer list. ** This is good for low memory environments using a pool allocator: the * top level allocation has a fixed size and the pointer lists have quite * small allocation size, which further matches the typical pool sizes * needed by objects, strings, property tables, etc.
label: code-design
19752. **comment:** XXX: remove this native function and map 'stack' accessor * to the toString() implementation directly.
label: code-design
19753. * Create a RegExp instance (E5 Section 15.10.7). ** Note: the output stack left by duk_regexp_compile() is directly compatible * with the input here. ** Input stack: [escaped_source bytecode] (both as strings) * Output stack: [RegExp]
19754. **comment:** log level could be popped but that's not necessary
label: code-design
19755. thread currently running (only one at a time)
19756. numargs
19757. already defined, good
19758. * Currently only allowed only if yielding thread has only * Ecmascript activations (except for the Duktape.Thread.yield() * call at the callstack top) and none of them constructor * calls. ** This excludes the 'entry' thread which will always have * a preventcount > 0.
19759. Copy term name until end or '/'.
19760. -> [... trap handler]
19761. Non-BMP characters within valid UTF-8 range: encode as is. * They'll decode back into surrogate pairs if the escaped * output is decoded.
19762. [... v1 v2 name filename str] -> [... str v2 name filename]
19763. DUK_TOK_DEBUGGER
19764. Init newly allocated slots (only).
19765. a complex (new) atom taints the result
19766. [this value]
19767. See: test-bug-tailcall-preventyield-assert.c.
19768. check_object_coercible
19769. **comment:** XXX: source code property
label: code-design
19770. **comment:** XXX: Would be nice to share the fast path loop from duk_hex_decode() * and avoid creating a temporary buffer. However, there are some * differences which prevent trivial sharing: ** - Pipe char detection * - EOF detection * - Unknown length of input and output ** The best approach here would be a bufwriter and a reasonably sized * safe inner loop (e.g. 64 output bytes at a time).
label: code-design
19771. **comment:** XXX: unnecessary, handle in adjust
label: code-design
19772. [... key trap handler]
19773. **comment:** * XXX: As noted above, a protected API call won't be counted as a * catcher. This is usually convenient, e.g. in the case of a top- * level duk_pcall(), but may not always be desirable. Perhaps add an * argument to treat them as catchers?
label: code-design
19774. const
19775. even for zero-length string
19776. eat 'if'
19777. 14: getDay
19778. Custom behavior: plain buffer is used as internal buffer * without making a copy (matches Duktape.Buffer).
19779. function: create an arguments object on function call
19780. -> [... target]
19781. * Nice debug log.
19782. **comment:** XXX: copy flags using a mask
label: code-design
19783. [... func/retval] -> [...]
19784. **comment:** XXX: optimize loops
label: code-design
19785. up to 36 bit codepoints
19786. no fractions in internal time
19787. ditto
19788. [key] -> []
19789. Track number of escapes; count not really needed but directive * prologues need to detect whether there were any escapes or line * continuations or not.
19790. [buf? res]
19791. **comment:** Duktape 0.11.0 and prior tried to optimize the resize by not * counting the number of actually used keys prior to the resize. * This worked mostly well but also caused weird leak-like behavior * as in: test-bug-object-prop-alloc-unbounded.js. So, now we count * the keys explicitly to compute the new entry part size.
label: code-design
19792. undefined -> skip (replaced with empty)
19793. eat opening quote on first loop
19794. **comment:** * Array built-ins * Note that most Array built-ins are intentionally generic and work even * when the 'this' binding is not an Array instance. To ensure this, * Array algorithms do not assume "magical" Array behavior for the "length" * property, for instance. ** XXX: the "Throw" flag should be set for (almost?) all [[Put]] and * [[Delete]] operations, but it's currently false throughout. Go through * all put/delete cases and check throw flag use. Need a new API primitive * which allows throws flag to be specified. ** XXX: array lengths above 2G won't work reliably. There are many places * where one needs a full signed 32-bit range ([-0xffffffff, 0xffffffff], * i.e. -33- bits). Although array 'length' cannot be written to be outside * the unsigned 32-bit range (E5.1 Section 15.4.5.1 throws a RangeError if so) * some intermediate values may be above 0xffffffff and this may not be always * correctly handled now (duk_uint32_t is not enough for all algorithms). * * For instance, push() can legitimately write entries beyond length 0xffffffff * and cause a RangeError only at the end. To do this properly, the current * push() implementation tracks the array index using a 'double' instead of a * duk_uint32_t (which is somewhat awkward). See test-bi-array-push-maxlen.js. * * On using "put" vs. "def" prop * ===== * * Code below must be careful to use the appropriate primitive as it matters * for compliance. When using "put" there may be inherited properties in * Array.prototype which cause side effects when values are written. When * using "define" there are no such side effects, and many test262 test cases * check for this (for real world code, such side effects are very rare). * Both "put" and "define" are used in the E5.1 specification; as a rule, * "put" is used when modifying an existing array (or a non-array 'this' * binding) and "define" for setting values into a fresh result array. * * Also note that Array instance 'length' should be writable, but not * enumerable and definitely not configurable: even Duktape code internally * assumes that an Array instance will always have a 'length' property. * Preventing deletion of the property is critical.
label: code-design

19795. no need to decref
19796. 18: getMinutes
19797. 'exports'
19798. * Unicode letter check.
19799. DUK_USE_MATH_BUILTIN
19800. No field needed when strings are in ROM.
19801. value was undefined/unsupported
19802. eat 'function'
19803. keep in valstack
19804. * Must guarantee all actually used array entries will fit into * new entry part. Add one growth step to ensure we don't run out * of space right away.
19805. step 11
19806. IdentifierPart production with IdentifierStart and ASCII excluded
19807. Internal timevalue is already NaN, so don't touch it.
19808. flag for later write
19809. GH-303
19810. Popping one element is called so often that when footprint is not an issue, * compile a specialized function for it.
19811. DUK_TOK_SEMICOLON
19812. return value from Duktape.Thread.yield()
19813. Causes a ToNumber() coercion, but doesn't break coercion order since * year is coerced first anyway.
19814. Parts are in local time, convert when setting.
19815. * E5 Section 7.4. If the multi-line comment contains a newline, * it is treated like a single line terminator for automatic * semicolon insertion.
19816. Note: if target_pc1 == i, we'll optimize a jump to itself. * This does not need to be checked for explicitly; the case * is rare and max iter breaks us out.
19817. Backup 'caller' property and update its value.
19818. assume exactly 1 arg, which is why '*' is forbidden; arg size still * depends on type though.
19819. Add function object.
19820. finalization
19821. [ToObject(this) item1 ... itemN arr]
19822. 0x50...0x5f
19823. Rule table: first matching rule is used to determine what to do next.
19824. CATCH flag may be enabled or disabled here; it may be enabled if * the statement has a catch block but the try block does not throw * an error.
19825. * Init the heap object
19826. Inherit from ROM-based global object: less RAM usage, less transparent.
19827. Note: 0xff != DUK_BC_C_MAX
19828. COMMA
19829. Node.js return value for noAssert out-of-bounds reads is * usually (but not always) NaN. Return NaN consistently.
19830. **comment:** function must not be tail called
 label: code-design
19831. stage 3: update length (done by caller), decide return code
19832. idx_space
19833. statement does not terminate directive prologue
19834. [requested_id require require.id resolved_id last_comp Duktape.Duktape.modLoaded undefined fresh_require exports module mod_func exports fresh_require exports module]
19835. **comment:** XXX: optimize for buffer inputs: no need to coerce to a string * which causes an unnecessary interning.
 label: code-design
19836. * Table for hex decoding ASCII hex digits
19837. function: create new environment when called (see duk_hcompiledfunction)
19838. pop regexp res_obj or match string
19839. P
19840. required if NAMEBINDING set
19841. slow path decode
19842. don't care value, year is mandatory
19843. really 'not applicable' anymore, should not be referenced after this
19844. skip jump conditionally
19845. * Flags parsing (see E5 Section 15.10.4.1).
19846. for updating (all are set to < 0 for virtual properties)
19847. * Run (object) finalizers in the "to be finalized" work list.
19848. 'warn'
19849. DUK_USE_FILE_IO
19850. [... constructor arg1 ... argN final_cons fallback]
19851. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur.
19852. 'Array' object, array length and index exotic behavior
19853. a statement following a label cannot be a source element * (a function declaration).
19854. 24: setMilliseconds
19855. '\ufffVarenv'
19856. **comment:** DUK_USE_DATE_GET_LOCAL_TZOFFSET() needs to be called with a * time value computed from UTC parts. At this point we only * have 'd' which is a time value computed from local parts, so * it is off by the UTC-to-local time offset which we don't know * yet. The current solution for computing the UTC-to-local * time offset is to iterate a few times and detect a fixed * point or a two-cycle loop (or a sanity iteration limit), * see test-bi-date-local-parts.js and test-bi-date-tzoffset-basic-fi.js. * * E5.1 Section 15.9.1.9: * UTC(t) = t - LocalTZA - DaylightSavingTA(t - LocalTZA) * * For NaN/inf, DUK_USE_DATE_GET_LOCAL_TZOFFSET() returns 0.
 label: code-design
19857. DUK_TOK_BREAK
19858. We're conceptually between two opcodes; act->pc indicates the next * instruction to be executed. This is usually the correct pc/line to * indicate in Status. (For the 'debugger' statement this now reports * the pc/line after the debugger statement because the debugger opcode * has already been executed.)
19859. DUK_TOK_MOD_EQ
19860. **comment:** * Write to 'length' of an array is a very complex case * handled in a helper which updates both the array elements * and writes the new 'length'. The write may result in an * unconditional RangeError or a partial write (indicated * by a return code). * * Note: the helper has an unnecessary writability check * for 'length', we already know it is writable.
 label: code-design
19861. **comment:** * Allocation size for 'curr_alloc' is alloc_size. There is no * automatic NUL terminator for buffers (see above for rationale). * * 'curr_alloc' is explicitly allocated with heap allocation * primitives and will thus always have alignment suitable for * e.g. duk_tval and an IEEE double.
 label: code-design
19862. * ctx->prev_token token to process with duk__expr_led() * ctx->curr_token updated by caller
19863. Note: target registers a and a+1 may overlap with DUK_REGP(b). * Careful here.
19864. * Mark unreachable, finalizable objects. * * Such objects will be moved aside and their finalizers run later. They have * to be treated as reachability roots for their properties etc to remain * allocated. This marking is only done for unreachable values which would * be swept later (refzero_list is thus excluded). * * Objects are first marked FINALIZABLE and only then marked as reachability * roots; otherwise circular references might be handled inconsistently.
19865. **comment:** XXX: accept any duk_hbufferobject type as an input also?

label: code-design

19866. Note: DST adjustment is determined using UTC time.

19867. caller must check

19868. 2

19869. **comment:** Fresh require: require.id is left configurable (but not writable) * so that is not easy to accidentally tweak it, but it can still be * done with Object.defineProperty(). * * XXX: require.id could also be just made non-configurable, as there * is no practical reason to touch it.**label:** code-design

19870. log this with a normal debug level because this should be relatively rare

19871. * Ecmascript execution, support primitives.

19872. * replace()

19873. out_clamped=NULL, RangeError if outside range

19874. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC

19875. idx_args = idx_func + 2

19876. handle comma and closing bracket

19877. * Avoid a GC if GC is already running. See duk_heap_mem_alloc().

19878. Note: this allows creation of internal strings.

19879. DUK_FLD_DOUBLE

19880. **comment:** * Logging * * Current logging primitive is a sprintf-style log which is convenient * for most C code. Another useful primitive would be to log N arguments * from value stack (like the Ecmascript binding does).**label:** code-design

19881. caller and arguments must use the same thrower, [[ThrowTypeError]]

19882. bound function 'length' property is interesting

19883. DUK_REPLACEMENTS_H_INCLUDED

19884. [... error func]

19885. extp 0x000 is zero/subnormal

19886. There is a caller; it MUST be an Ecmascript caller (otherwise it would * match entry level check)

19887. restore PC

19888. allow negative PCs, behave as a no-op

19889. '\xffValue'

19890. * Activation defines

19891. Step 15: insert itemCount elements into the hole made above

19892. No need to replace the 'enum_target' value in stack, only the * enum_target reference. This also ensures that the original * enum target is reachable, which keeps the proxy and the proxy * target reachable. We do need to replace the internal _Target.

19893. XXX: ARRAY_PART for Array prototype?

19894. Coerce value to a number before computing check_length, so that * the field type specific coercion below can't have side effects * that would invalidate check_length.

19895. string is a valid array index

19896. * Fast path: assume no mutation, iterate object property tables * directly; bail out if that assumption doesn't hold.

19897. 'Number'

19898. buffer is behind a pointer, dynamic or external

19899. * in

19900. allow caller to give a const number with the DUK_CONST_MARKER

19901. DUK_USE_JSON_DECNUMBER_FASTPATH

19902. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack (e.g. if * lval is already a string).

19903. XXX: use DUK_HSTRING_FLAG_INTERNAL?

19904. func support for [[HasInstance]] checked in the beginning of the loop

19905. %p

19906. weak refs should be handled here, but no weak refs for * any non-string objects exist right now.

19907. pop key

19908. implementation specific

19909. * Log level check

19910. **comment:** XXX: unify handling with native call.**label:** code-design

19911. internal extra elements assumed on function entry, * always added to user-defined 'extra' for e.g. the * duk_check_stack() call.

19912. * Complex atom * * The original code is used as a template, and removed at the end * (this differs from the handling of simple quantifiers). * * NOTE: there is no current solution for empty atoms in complex * quantifiers. This would need some sort of a 'progress' instruction. * * XXX: impose limit on maximum result size, i.e. atom_code_len * atom_copies?

19913. Leading zero is not counted towards precision digits; not * in the integer part, nor in the fraction part.

19914. 39: setYear

19915. **comment:** * ToString() (E5 Section 9.8) * * ==> implemented in the API.**label:** requirement

19916. E5.1 Section 15.9.1.6

19917. DUK_USE_JSON_QUOTESTRING_FASTPATH

19918. always in executor

19919. because of valstack init policy

19920. embed: func ptr

19921. local

19922. Sanity check

19923. * Object environment record. * * Binding (target) object is an external, uncontrolled object. * Identifier may be bound in an ancestor property, and may be * an accessor. Target can also be a Proxy which we must support * here.

19924. * Manipulate callstack for the call.

19925. Currently the bytecode executor and executor interrupt * instruction counts are off because we don't execute the * interrupt handler when we're about to exit from the initial * user call into Duktape. * * If we were to execute the interrupt handler here, the counts * would match. You can enable this block manually to check * that this is the case.

19926. **comment:** Note: combining comparison ops must be done carefully because * of uncomparable values (NaN): it's not necessarily true that * $(x \geq y) == !(x < y)$. Also, evaluation order matters, and * although it would only seem to affect the compiler this is * actually not the case, because there are also run-time coercions * of the arguments (with potential side effects). * * XXX: can be combined; check code size.**label:** code-design

19927. typeof

19928. Execute bytecode until returned or longjmp().

19929. overflow not possible, buffer limits

19930. 'interface'

19931. zero everything unless requested not to do so

19932. E5 Section 15.12.3, main algorithm, step 4.b.ii steps 1-4.

19933. * Node.js Buffer.isBuffer()

19934. DUK_TOK_COLON

19935. B -> target register for next key * C -> enum register

19936. -> [...] flags escaped_source bytecode]

19937. backref n -> saved indices [n*2, n*2+1]
19938. value1 -> error object
19939. remove key
19940. **comment:** * String hash computation (interning). ** String hashing is performance critical because a string hash is computed * for all new strings which are candidates to be added to the string table. * However, strings actually added to the string table go through a codepoint * length calculation which dominates performance because it goes through * every byte of the input string (but only for strings added). ** The string hash algorithm should be fast, but on the other hand provide * good enough hashes to ensure both string table and object property table * hash tables work reasonably well (i.e., there aren't too many collisions * with real world inputs). Unless the hash is cryptographic, it's always * possible to craft inputs with maximal hash collisions. ** NOTE: The hash algorithms must match src/dukutil.py:duk_heap_hashstring() * for ROM string support!
label: code-design
19941. **comment:** XXX: should the API call handle this directly, i.e. attempt * to duk_push_hobject(ctx, null) would push a null instead? * (On the other hand 'undefined' would be just as logical, but * not wanted here.)
label: code-design
19942. **comment:** For short strings, use a fixed temp buffer.
label: code-design
19943. label limits
19944. marker values for hash part
19945. DUK_TOK_GE
19946. -> [...] closure
19947. NULL pointer is required to encode to zero, so memset is enough.
19948. else an array initializer element
19949. * E5 Section 7.3 ** A LineTerminatorSequence essentially merges <CR> <LF> sequences * into a single line terminator. This must be handled by the caller.
19950. 35: setUTCMonth
19951. [...] retval]
19952. Input shuffling happens before the actual operation, while output * shuffling happens afterwards. Output shuffling decisions are still * made at the same time to reduce branch clutter; output shuffle decisions * are recorded into X_out variables.
19953. 'new' MemberExpression Arguments
19954. assume plain values
19955. duk_regconst_t is unsigned, so use 0 as dummy value (ignored by caller)
19956. [...] val obj]
19957. -> [...] re_obj input bc saved_buf lastIndex]
19958. **comment:** (advance << 8) + token_type, updated at function end, * init is unnecessary but suppresses "may be used uninitialized" warnings.
label: code-design
19959. first, parse regexp body roughly
19960. Duktape/C (nativefunction) object, exotic 'length'
19961. DUK_HOBJECT_FLAG_EXOTIC_ARRAY varies
19962. 'this' binding
19963. [...] val key] -> [...] key val]
19964. isAccessor
19965. Cannot be compressed as a heap pointer because may point to * an arbitrary address.
19966. skip opening slash on first loop
19967. transfer flags
19968. DUK_TOK_INTERFACE
19969. may be undefined
19970. accept anything, expect first value (EOF will be * caught by duk_dec_value() below.
19971. iteration statements allow continue
19972. Clamp so that values at 'clamp_top' and above are wiped and won't * retain reachable garbage. Then extend to 'nregs' because we're * returning to an Ecmascript function.
19973. 'while'
19974. **comment:** Checking this here rather than in memory alloc primitives * reduces checking code there but means a failed allocation * will go through a few retries before giving up. That's * fine because this only happens during debugging.
label: code-design
19975. **comment:** * Array abandon check; abandon if: * * new_used / new_size < limit * new_used < limit * new_size || limit is 3 bits fixed point * new_used < limit' / 8 * new_size || *8 * 8*new_used < limit' * new_size || :8 * new_used < limit' * (new_size / 8) ** Here, new_used = a_used, new_size = a_size. ** Note: some callers use approximate values for a_used and/or a_size * (e.g. dropping a '+1' term). This doesn't affect the usefulness * of the check, but may confuse debugging.
label: code-design
19976. Target is before source. Source offset is expressed as * a "before change" offset. Account for the memmove.
19977. object is constructable
19978. regexp support disabled
19979. This should be equivalent to match() algorithm step 8.f.iii.2: * detect an empty match and allow it, but don't allow it twice.
19980. assume keys are compacted
19981. currently implicitly also DUK_USE_DOUBLE_LINKED_HEAP
19982. advance, whatever the current token is; parse next token in regexp context
19983. *ToInt32(), ToUint32(), ToUint16() (E5 Sections 9.5, 9.6, 9.7)
19984. rego to allocate
19985. preallocated temporaries (2) for variants 3 and 4
19986. [regexp input]
19987. function: function object is strict
19988. DUK_TOK_ALSHIFT
19989. **comment:** XXX: what to do if _Formals is not empty but compiler has * optimized it away -- read length from an explicit property * then?
label: code-design
19990. work list for objects whose refcounts are zero but which have not been * "finalized"; avoids recursive C calls when refcounts go to zero in a * chain of objects.
19991. **comment:** context and locale specific rules which cannot currently be represented * in the caseconv bitstream: hardcoded rules in C
label: code-design
19992. * Heap string representation. ** Strings are byte sequences ordinarily stored in extended UTF-8 format, * allowing values larger than the official UTF-8 range (used internally) * and also allowing UTF-8 encoding of surrogate pairs (CESU-8 format). * Strings may also be invalid UTF-8 altogether which is the case e.g. with * strings used as internal property names and raw buffers converted to * strings. In such cases the 'clen' field contains an inaccurate value. ** Ecmascript requires support for 32-bit long strings. However, since each * 16-bit codepoint can take 3 bytes in CESU-8, this representation can only * support about 1.4G codepoint long strings in extreme cases. This is not * really a practical issue.
19993. call is a direct eval call
19994. varname is still reachable
19995. Leave 'setter' on stack
19996. seconds
19997. **comment:** * Ecmascript compiler. ** Parses an input string and generates a function template result. * Compilation may happen in multiple contexts (global code, eval * code, function code). ** The parser uses a traditional top-down recursive parsing for the * statement level, and an operator precedence based top-down approach * for the expression level. The attempt is to minimize the C stack * depth. Bytecode is generated directly without an intermediate * representation (tree), at the cost of needing two passes over each * function. ** The top-down recursive parser functions are named "duk_parse_XXX". ** Recursion limits are in key functions to prevent arbitrary C recursion: * function body parsing, statement parsing, and expression parsing. ** See doc/compiler.rst for discussion on the

design. ** A few typing notes: ** - duk_regconst_t: unsigned, no marker value for "none" * - duk_reg_t: signed, < 0 = none * - PC values: duk_int_t, negative values used as markers

label: code-design

19998. Append a "(line NNN)" to the "message" property of any error * thrown during compilation. Usually compilation errors are * SyntaxErrors but they can also be out-of-memory errors and * the like.

19999. **comment:** XXX: any way to detect faster whether something needs to be closed? * We now look up _Callee and then skip the rest.

label: requirement

20000. DUK_USE_VARIADIC_MACROS

20001. * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H ** s sign bit * eee... exponent field * fff... fraction ** ieee value = 1.ffff... * 2^(e - 1023) (normal) * = 0.ffff... * 2^{(-1022)} (denormal) ** algorithm v = f * b^e

20002. key is now reachable in the valstack

20003. MakeDate

20004. Request callback should push values for reply to client onto valstack

20005. inserted jump

20006. [enum_target res]

20007. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize or a forced relocating realloc?

label: code-design

20008. For Ecmascript strings, this check can only match for * initial UTF-8 bytes (not continuation bytes). For other * strings all bets are off.

20009. Then reuse the unwound activation; callstack was not shrunk so there is always space

20010. DUK_USE_DEBUGGER_DUMPHEAP || DUK_USE_DEBUGGER_INSPECT

20011. for storing/restoring the varmap binding for catch variable

20012. r <- (2 * f) * b^(e+1)

20013. [...]'

20014. **comment:** Careful here and with other duk_put_prop_xxx() helpers: the * target object and the property value may be in the same value * stack slot (unusual, but still conceptually clear).

label: code-design

20015. function is strict

20016. must work for nargs <= 0

20017. Ignore digits beyond a radix-specific limit, but note them * in expt_adj.

20018. don't run finalizers; leave finalizable objects in finalize_list for next round

20019. [this value] -> [value this]

20020. Update 'buffer_length' to be the effective, safe limit which * takes into account the underlying buffer. This value will be * potentially invalidated by any side effect.

20021. 2: toTimeString

20022. **comment:** No need for constants pointer (= same as data). ** When using 16-bit packing alignment to 4 is nice. 'funcs' will be * 4-byte aligned because 'constants' are duk_tvals. For now the * inner function pointers are not compressed, so that 'bytecode' will * also be 4-byte aligned.

label: code-design

20023. * Initialize function state for a zero-argument function

20024. Guaranteed by recursion_limit setup so we don't have to * check twice.

20025. line continuation

20026. IsGenericDescriptor(desc) == true; this means in practice that 'desc' * only has [[Enumerable]] or [[Configurable]] flag updates, which are * allowed at this point.

20027. '-Infinity'

20028. * Resume a thread. ** The thread must be in resumable state, either (a) new thread which hasn't * yet started, or (b) a thread which has previously yielded. This method * must be called from an Ecmascript function. ** Args: * - thread * - value * - isError (defaults to false) ** Note: yield and resume handling is currently asymmetric.

20029. type tag (public)

20030. **comment:** It'd be nice to do something like this - but it doesn't * work for closures created inside the catch clause.

label: code-design

20031. don't re-enter e.g. during Eval

20032. mp <- b^e

20033. special handling of fmt==NULL

20034. Ensure space for final configuration (idx_rebase + num_stack_rets) * and intermediate configurations.

20035. parse value

20036. **comment:** Note: not honoring round-to-even should work but now generates incorrect * results. For instance, 1e23 serializes to "a000...", i.e. the first digit * equals the radix (10). Scaling stops one step too early in this case. * Don't know why this is the case, but since this code path is unused, it * doesn't matter.

label: code-design

20037. **comment:** The 'strict' flag is copied to get the special [[Get]] of E5.1 * Section 15.3.5.4 to apply when a 'caller' value is a strict bound * function. Not sure if this is correct, because the specification * is a bit ambiguous on this point but it would make sense.

label: code-design

20038. function may call direct eval

20039. **comment:** _Varmap: omitted if function is guaranteed not to do slow path identifier * accesses or if it would turn out to be empty of actual register mappings * after a cleanup. When debugging is enabled, we always need the varmap to * be able to lookup variables at any point.

label: code-design

20040. [array totalLength]

20041. Opcode interval for a Date-based status/peek rate limit check. Only * relevant when debugger is attached. Requesting a timestamp may be a * slow operation on some platforms so this shouldn't be too low. On the * other hand a high value makes Duktape react to a pause request slowly.

20042. -> [... key val val']

20043. [... re_obj input] -> [... re_obj input bc]

20044. valstack slot for 1st token value

20045. side effects, possibly errors

20046. NOTE: Multiple evaluation of 'ptr' in this macro.

20047. Overflow of the execution counter is fine and doesn't break * anything here.

20048. mark-and-sweep: finalized (on previous pass)

20049. curr is accessor, desc is data

20050. [func thisArg arg1 ... argN]

20051. XXX: the insert helpers should ensure that the bytecode result is not * larger than expected (or at least assert for it). Many things in the * bytecode, like skip offsets, won't work correctly if the bytecode is * larger than say 2G.

20052. this function

20053. **comment:** 'tv' becomes invalid

label: code-design

20054. is an actual function (not global/eval code)

20055. Catch attempts to use out-of-range reg/const. Without this * check Duktape 0.12.0 could generate invalid code which caused * an assert failure on execution. This error is triggered e.g. * for functions with a lot of constants and a try-catch statement. * Shuffling or opcode semantics change is needed to fix the issue. * See: test-bug-trycatch-many-constants.js

20056. 'callee'

20057. Initial stack size satisfies the stack spare constraints so there * is no need to require stack here.

20058. 'byteOffset'

20059. **comment:** duk_handle_call_xxx: constructor call (i.e. called as 'new Foo()')

label: code-design

20060. no operators
20061. Strict: never allow function declarations outside top level.
20062. [... arg1 ... argN envobj]
20063. * Field access
20064. 0x30...0x3f
20065. value1 -> label number, pseudo-type to indicate a continue continuation (for ENDFIN)
20066. true
20067. **comment:** XXX: common reg allocation need is to reuse a sub-expression's temp reg, * but only if it really is a temp. Nothing fancy here now.
 label: code-design
20068. only use the highest bit
20069. Function values are handled completely above (including * coercion results):
20070. distinct bignums, easy mistake to make
20071. **comment:** XXX: this is now a very unoptimal implementation -- this can be * made very simple by direct manipulation of the object internals, * given the guarantees above.
 label: code-design
20072. Split time value into parts. The time value is assumed to be an internal * one, i.e. finite, no fractions. Possible local time adjustment has already * been applied when reading the time value.
20073. If flags != 0 (strict or SameValue), thr can be NULL. For loose * equals comparison it must be != NULL.
20074. DUK_TOK_INSTANCEOF
20075. time
20076. update entry, allocating if necessary
20077. A structure for 'snapshotting' a point for rewinding
20078. * Ecmascript call
20079. cannot be arguments exotic
20080. **comment:** this is not strictly necessary, but helps debugging
 label: code-design
20081. * Prototypes for built-in functions not automatically covered by the * header declarations emitted by genbuiltins.py.
20082. push initial function call to new thread stack; this is * picked up by resume().
20083. 31: setUTCHours
20084. Decode a one-bit flag, and if set, decode a value of 'bits', otherwise return * default value. Return value is signed so that negative marker value can be * used by caller as a "not present" value.
20085. format plus something to avoid just missing
20086. **comment:** XXX: post-checks (such as no duplicate keys)
 label: code-design
20087. if property already exists, overwrites silently
20088. **comment:** 'val' is never NaN, so no need to normalize
 label: code-design
20089. * Lowercase digits for radix values 2 to 36. Also doubles as lowercase * hex nybble table.
20090. may be reg or const
20091. decl name
20092. rnd_state for duk_util_tinyrandom.c
20093. Note: heap->heap_thread, heap->curr_thread, and heap->heap_object * are on the heap allocated list.
20094. jump is inserted here
20095. misc
20096. 'length' and other virtual properties are not * enumerable, but are included if non-enumerable * properties are requested.
20097. only flag now
20098. [... func this arg1 ... argN]
20099. record previous atom info in case next token is a quantifier
20100. '-'
20101. * Ecmascript compliant [[GetOwnProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * Notes: * * - Getting a property descriptor may cause an allocation (and hence * GC) to take place, hence reachability and refcount of all related * values matter. Reallocation of value stack, properties, etc may * invalidate many duk_tval pointers (concretely, those which reside * in memory areas subject to reallocation). However, heap object * pointers are never affected (heap objects have stable pointers). * * - The value of a plain property is always reachable and has a non-zero * reference count. * * - The value of a virtual property is not necessarily reachable from * elsewhere and may have a refcount of zero. Hence we push it onto * the valstack for the caller, which ensures it remains reachable * while it is needed. * * - There are no virtual accessor properties. Hence, all getters and * setters are always related to concretely stored properties, which * ensures that the get/set functions in the resulting descriptor are * reachable and have non-zero refcounts. Should there be virtual * accessor properties later, this would need to change.
20102. not enumerable
20103. %f and %lf both consume a 'long'
20104. * Catcher defines
20105. never executed
20106. **comment:** At least 'magic' has a significant impact on function * identity.
 label: code-design
20107. '
20108. important to do this *after* pushing, to make the thread reachable for gc
20109. [offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
20110. Should not happen.
20111. init pointer fields to null
20112. * toString(), valueOf()
20113. success, fixup pointers
20114. **comment:** XXX: several pointer comparison issues here
 label: code-design
20115. insert (DUK_REOP_WIPERANGE, start, count) in reverse order so the order ends up right
20116. Check that value is a duk_hbufferobject and return a pointer to it.
20117. Note: DUK_VALSTACK_INITIAL_SIZE must be >= DUK_VALSTACK_API_ENTRY_MINIMUM * + DUK_VALSTACK_INTERNAL_EXTRA so that the initial stack conforms to spare * requirements.
20118. **comment:** XXX: may need a 'length' filter for forEach()
 label: code-design
20119. Note: assumes that duk_util_probe_steps size is 32
20120. guaranteed when building arguments
20121. updates thread state, minimizes its allocations
20122. '{"_inf":true}'
20123. **comment:** Without variadic macros resort to comma expression trickery to handle debug * prints. This generates a lot of harmless warnings. These hacks are not * needed normally because DUK_D() and friends will hide the entire debug log * statement from the compiler.
 label: code-design
20124. **comment:** TimeClip() should never be necessary

label: code-design
20125. 0x10-0x1f
20126. E5.1 Section B.2.2, step 7.
20127. Parser part masks.
20128. end switch (tok)
20129. negative -> complex atom
20130. -> [... val root ""]
20131. result array
20132. XXX: fast path for arrays?
20133. -> [... ToObject(this) ToUint32(length)]
20134. Continue checked execution if there are breakpoints or we're stepping. * Also use checked execution if paused flag is active - it shouldn't be * because the debug message loop shouldn't terminate if it was. Step out * is handled by callstack unwind and doesn't need checked execution. * Note that debugger may have detached due to error or explicit request * above, so we must recheck attach status.
20135. 0x00...0x0f
20136. coercion order matters
20137. 'jc'
20138. 1 if property found, 0 otherwise
20139. fixed-format
20140. not strictly necessary because of lookahead ')' above
20141. **comment:** XXX: must be able to represent -len
 label: code-design
20142. DUK_USE_DPRINT_COLORS
20143. expect_eof
20144. borrowed reference to catch variable name (or NULL if none)
20145. **comment:** XXX: these could be implemented as macros calling an internal function * directly. * XXX: same issue as with Duktape.fin: there's no way to delete the property * now (just set it to undefined).
 label: code-design
20146. **comment:** XXX: set magic directly here? (it could share the c_nargs arg)
 label: code-design
20147. lookup should prevent this
20148. * Preliminary activation record and valstack manipulation. * The concrete actions depend on whether we're dealing * with a tail call (reuse an existing activation), a resume, * or a normal call. * * The basic actions, in varying order, are: * * - Check stack size for call handling * - Grow call stack if necessary (non-tail-calls) * - Update current activation (idx_retval) if necessary * (non-tail, non-resume calls) * - Move start of args (idx_args) to valstack bottom * (tail calls) * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
20149. result index for filter()
20150. either nonzero value is ok
20151. bc
20152. * DELVAR * * See E5 Sections: * 11.4.1 The delete operator * 10.2.1.1.5 DeleteBinding (N) [declarative environment record] * 10.2.1.2.5 DeleteBinding (N) [object environment record] * * Variable bindings established inside eval() are deletable (configurable), * other bindings are not, including variables declared in global level. * Registers are always non-deletable, and the deletion of other bindings * is controlled by the configurable flag. * * For strict mode code, the 'delete' operator should fail with a compile * time SyntaxError if applied to identifiers. Hence, no strict mode * run-time deletion of identifiers should ever happen. This function * should never be called from strict mode code!
20153. * Mark refzero_list objects. * * Objects on the refzero_list have no inbound references. They might have * outbound references to objects that we might free, which would invalidate * any references held by the refzero objects. A refzero object might also * be rescued by refcount finalization. Refzero objects are treated as * reachability roots to ensure they (or anything they point to) are not * freed in mark-and-sweep.
20154. Negative zero needs special handling in JX/JC because * it would otherwise serialize to '0', not '-0'.
20155. can't resize below 'top'
20156. round up roughly to next 'grow step'
20157. **comment:** * E5 Section 7.6: * * IdentifierPart: * IdentifierStart * UnicodeCombiningMark * UnicodeDigit * UnicodeConnectorPunctuation * <ZWNJ> [U+200C] * <ZWJ> [U+200D] * * IdentifierPart production has one multi-character production * as part of its IdentifierStart alternative. The '\v' character * of an escape sequence is not matched here, see discussion in * duk_unicode_is_identifier_start(). * * To match non-ASCII characters (codepoints >= 0x80), a very slow * linear range-by-range scan is used. The codepoint is first compared * to the IdentifierStart ranges, and if it doesn't match, then to a * set consisting of code points in IdentifierPart but not in * IdentifierStart. This is done to keep the unicode range data small, * at the expense of speed. * * The ASCII fast path consists of: * * 0x0030 ... 0x0039 ['0' ... '9', UnicodeDigit] * 0x0041 ... 0x005a ['A' ... 'Z', IdentifierStart] * 0x0061 ... 0x007a ['a' ... 'z', IdentifierStart] * 0x0024 ['\$', IdentifierStart] * 0x005f['_', IdentifierStart and * UnicodeConnectorPunctuation] * * UnicodeCombiningMark has no code points <= 0x7f. * * The matching code reuses the "identifier start" tables, and then * consults a separate range set for characters in "identifier part" * but not in "identifier start". These can be extracted with the * "src/extract_chars.py" script. * * UnicodeCombiningMark -> categories Mn, Mc * UnicodeDigit -> categories Nd * UnicodeConnectorPunctuation -> categories Pc
 label: code-design
20158. last array index explicitly initialized, +1
20159. Value would yield 'undefined', so skip key altogether. * Side effects have already happened.
20160. need side effects, not value
20161. * Check function name validity now that we know strictness. * This only applies to function declarations and expressions, * not setter/getter name. * * See: test-dev-strict-mode-boundary.js
20162. Target may be a Proxy or property may be an accessor, so we must * use an actual, Proxy-aware hasprop check here. * * out->holder is NOT set to the actual duk_hobject where the * property is found, but rather the object binding target object.
20163. Note: for srclen=0, src may be NULL
20164. never an empty match, so step 13.c.iii can't be triggered
20165. Outer executor with setjmp/longjmp handling.
20166. if B/C is >= this value, refers to a const
20167. if debugging disabled
20168. if true, the stack already contains the final result
20169. * E5 Section 7.2 specifies six characters specifically as * white space: * * 0009;<control>;Cc;0;S;;;;N;CHARACTER TABULATION;;;; * 000B;
 <control>;Cc;0;S;;;;N;LINE TABULATION;;;; * 000C;<control>;Cc;0;WS;;;;N;FORM FEED (FF);;; * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK
 SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; * FEFF;ZERO WIDTH NO-BREAK SPACE;Cf;0;BN;;;;N;BYTE ORDER MARK;;;; * It
 also specifies any Unicode category 'Zs' characters as white * space. These can be extracted with the "src/extract_chars.py" script. * Current result: * * RAW
 OUTPUT: * ===== * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; *
 1680;OGHAM SPACE MARK;Zs;0;WS;;;;N;;;; * 180E;MONGOLIAN VOWEL SEPARATOR;Zs;0;WS;;;;N;;;; * 2000;EN QUAD;Zs;0;WS;2002;;;;N;;;; *
 2001;EM QUAD;Zs;0;WS;2003;;;;N;;;; * 2002;EN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2003;EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; *
 2004;THREE-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2005;FOUR-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2006;SIX-PER-EM
 SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2007;FIGURE SPACE;Zs;0;WS;<noBreak> 0020;;;;N;;;; * 2008;PUNCTUATION SPACE;Zs;0;WS;<compat>
 0020;;;;N;;;; * 2009;THIN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 200A;HAIR SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 202F;NARROW NO-BREAK
 SPACE;Zs;0;CS;<noBreak> 0020;;;;N;;;; * 205F;MEDIUM MATHEMATICAL SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 3000;IDEOGRAPHIC
 SPACE;Zs;0;WS;<wide> 0020;;;;N;;;; * * RANGES: * ===== * 0x0020 * 0x00a0 * 0x1680 * 0x180e * 0x2000 ... 0x200a * 0x202f * 0x205f * 0x3000 * * A
 manual decoder (below) is probably most compact for this.
20170. ToNumber(bool) is +1.0 or 0.0. Tagged boolean value is always 0 or 1.
20171. when nothing is running, API calls are in non-strict mode
20172. **comment:** String length is computed here to avoid multiple evaluation * of a macro argument in the calling side.

label: code-design

20173. see duk_js_executor.c

20174. Convert heap string index to a token (reserved words)

20175. **comment:** XXX: refactor and share with other code

label: code-design

20176. order must match constants in genbuiltins.py

20177. Buffer is kept as is, with the fixed/dynamic nature of the * buffer only changed if requested. An external buffer * is converted into a non-external dynamic buffer in a * duk_to_dynamic_buffer() call.

20178. * Ecmascript compliant [[Delete]](P, Throw).

20179. -> [... str]

20180. args go here in parens

20181. 0xb0...0xbf

20182. Write in big endian

20183. only need to guarantee 1 more slot, but allocation growth is in chunks

20184. **comment:** alloc function typedefs in duktape.h

label: code-design

20185. i >= 0 would always be true

20186. **comment:** XXX: inline into a prototype walking loop?

label: code-design

20187. Lightfunc handling by ToObject() coercion.

20188. DUK_BUFOBJ_UINT16ARRAY

20189. 'else'

20190. 'String' object, array index exotic behavior

20191. A -> target reg * B -> object reg/const (may be const e.g. in "foo'[1]") * C -> key reg/const

20192. There is no eval() special handling here: eval() is never * automatically converted to a lightfunc.

20193. **comment:** XXX: identify enumeration target with an object index (not top of stack)

label: code-design

20194. Write unsigned 32-bit integer.

20195. The underlying types for offset/length in duk_hbufferobject is * duk_uint_t; make sure argument values fit and that offset + length * does not wrap.

20196. [body formals source template closure]

20197. no refcount changes

20198. [arg1 ... argN-1 body] -> [body arg1 ... argN-1]

20199. First evaluate LHS fully to ensure all side effects are out.

20200. properties object

20201. Call handling and success path. Success path exit cleans * up almost all state.

20202. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer * stack[4] = regexp match OR match string

20203. object is extensible

20204. * The length comparisons are present to handle * strings like "use strict\u0000foo" as required.

20205. lookup name from an open declarative record's registers

20206. relookup (side effects)

20207. **comment:** XXX: make a common DUK_USE_ option, and allow custom fixed seed?

label: code-design

20208. x <- y * z

20209. * Note: assuming new_a_size == 0, and that entry part contains * no conflicting keys, refcounts do not need to be adjusted for * the values, as they remain exactly the same. * * The keys, however, need to be interned, incref'd, and be * reachable for GC. Any intern attempt may trigger a GC and * claim any non-reachable strings, so every key must be reachable * at all times. * * A longjmp must not occur here, as the new_p allocation would * be freed without these keys being decref'd, hence the messy * decref handling if intern fails.

20210. **comment:** * 'props' contains {key,value,flags} entries, optional array entries, and * an optional hash lookup table for non-array entries in a single 'sliced' * allocation. There are several layout options, which differ slightly in * generated code size/speed and alignment/padding; duk_features.h selects * the layout used. * * Layout 1 (DUK_USE_HOBJECT_LAYOUT_1): ** e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * * Layout 2 (DUK_USE_HOBJECT_LAYOUT_2): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_uint8_t) + pad bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * * Layout 3 (DUK_USE_HOBJECT_LAYOUT_3): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * * In layout 1, the 'e_next' count is rounded to 4 or 8 on platforms * requiring 4 or 8 byte alignment. This ensures proper alignment * for the entries, at the cost of memory footprint. However, it's * probably preferable to use another layout on such platforms instead. * * In layout 2, the key and value parts are swapped to avoid padding * the key array on platforms requiring alignment by 8. The flags part * is padded to get alignment for array entries. The 'e_next' count does * not need to be rounded as in layout 1. * * In layout 3, entry values and array values are always aligned properly, * and assuming pointers are at most 8 bytes, so are the entry keys. Hash * indices will be properly aligned (assuming pointers are at least 4 bytes). * Finally, flags don't need additional alignment. This layout provides * compact allocations without padding (even on platforms with alignment * requirements) at the cost of a bit slower lookups. * * Objects with few keys don't have a hash index; keys are looked up linearly, * which is cache efficient because the keys are consecutive. Larger objects * have a hash index part which contains integer indexes to the entries part. * * A single allocation reduces memory allocation overhead but requires more * work when any part needs to be resized. A sliced allocation for entries * makes linear key matching faster on most platforms (more locality) and * skimps on flags size (which would be followed by 3 bytes of padding in * most architectures if entries were placed in a struct). * * 'props' also contains internal properties distinguished with a non-BMP * prefix. Often used properties should be placed early in 'props' whenever * possible to make accessing them as fast as possible.

label: code-design

20211. Catches >0x100000000 and negative values.

20212. * Entry points

20213. * Error built-ins

20214. Packed or unpacked tval

20215. * PC-to-line constants

20216. top-down expression parser

20217. E5 Section 15.4.5.1, steps 4.e.i - 4.e.ii

20218. fall through

20219. x

20220. **comment:** XXX: write protect after flag? -> any chance of handling it here?

label: code-design

20221. Term was ' and is eaten entirely (including dup slashes).

20222. fast path for ASCII

20223. num_stack_res

20224. %lx

20225. * Not found as concrete or virtual

20226. 'Buffer'

20227. [... env target target]

20228. DUK_TOK_LOR
20229. DUK_OP_DECLVAR flags in A; bottom bits are reserved for propdesc flags (DUK_PROPDESC_FLAG_XXX)
20230. Day-of-month is one-based in the API, but zero-based * internally, so fix here. Note that month is zero-based * both in the API and internally.
20231. The coercion order must match the ToPropertyDescriptor() algorithm * so that side effects in coercion happen in the correct order. * (This order also happens to be compatible with duk_def_prop(), * although it doesn't matter in practice.)
20232. triggers garbage digit check below
20233. relevant array index is non-configurable, blocks write
20234. **comment:** the NaN variant we use
 label: code-design
20235. * Compact the function template.
20236. PRIMARY EXPRESSIONS
20237. * indexOf(), lastIndexOf()
20238. Sometimes this assert is not true right now; it will be true after * rounding. See: test-bug-numconv-mantissa-assert.js.
20239. Automatic error throwing, retval check.
20240. Fast path for the case where the register * is a number (e.g. loop counter).
20241. always push some string
20242. spare
20243. Note: new_e_next matches pushed temp key count, and nothing can * fail above between the push and this point.
20244. * Heap flags
20245. Object extra properties. * * There are some difference between function templates and functions. * For example, function templates don't have .length and nargs is * normally used to instantiate the functions.
20246. We intentionally ignore the duk_safe_call() return value and only * check the output type. This way we don't also need to check that * the returned value is indeed a string in the success case.
20247. * Arguments exotic behavior not possible for new properties: all * magically bound properties are initially present in the arguments * object, and if they are deleted, the binding is also removed from * parameter map.
20248. get_value
20249. clear array part flag only after switching
20250. !'
20251. tv1 points to value storage
20252. Map DUK_HBUFFEROBJECT_ELEM_xxx to prototype object built-in index. * Sync with duk_hbufferobject.h.
20253. * Create "fallback" object to be used as the object instance, * unless the constructor returns a replacement value. * Its internal prototype needs to be set based on "prototype" * property of the constructor.
20254. Slightly modified "Bernstein hash" from: * * http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx * * Modifications: string skipping and reverse direction similar to * Lua 5.1.5, and different hash initializer. * * The reverse direction ensures last byte is always included in the * hash which is a good default as changing parts of the string are * more often in the suffix than in the prefix.
20255. ascii fast path: avoid decoding utf-8
20256. just in case
20257. entry_top + 4
20258. m- <- (* m- B)
20259. UTC
20260. no valstack space check
20261. End of input (NUL) goes through slow path and causes SyntaxError.
20262. [... enum_target res trap_result val]
20263. **comment:** XXX: unnecessary '%s' formatting here, but cannot use * 'msg' as a format string directly.
 label: code-design
20264. ptr_curr_pc != NULL only when bytecode executor is active.
20265. Restore entry thread executor curr_pc stack frame pointer.
20266. Storing the entry top is cheaper here to ensure stack is correct at exit, * as there are several paths out.
20267. duk_hobject_set_length_zero(thr, func->h_funcs);
20268. fall thru
20269. see above
20270. * Struct defines
20271. [... name reg/null] -> [...]
20272. mandatory if du.d is a NaN
20273. duk_tval ptr for 'func' on stack (borrowed reference)
20274. For tv1 == tv2, this is a no-op (no explicit check needed).
20275. Use b[] to access the size of the union, which is strictly not * correct. Can't use fixed size unless there's feature detection * for pointer byte size.
20276. local copy to avoid relookups
20277. **comment:** XXX: this has some overlap with object inspection; remove this and make * DumpHeap return lists of heapptrs instead?
 label: code-design
20278. Return to the bytecode executor caller which will unwind stacks. * Return value is already on the stack top: [... retval].
20279. * Hobject Ecmascript [[Class]].
20280. Handle one slow path unit (or finish if we're done).
20281. **comment:** Copy the .buffer property, needed for TypedArray.prototype.subarray(). * * XXX: limit copy only for TypedArray classes specifically?
 label: code-design
20282. enough to cover the whole mantissa
20283. DUK_TOK_BXOR_EQ
20284. **comment:** * Hobject enumeration support. * * Creates an internal enumeration state object to be used e.g. with for-in * enumeration. The state object contains a snapshot of target object keys * and internal control state for enumeration. Enumerator flags allow caller * to e.g. request internal/non-enumerable properties, and to enumerate only * "own" properties. * * Also creates the result value for e.g. Object.keys() based on the same * internal structure. * * This snapshot-based enumeration approach is used to simplify enumeration: * non-snapshot-based approaches are difficult to reconcile with mutating * the enumeration target, running multiple long-lived enumerators at the * same time, garbage collection details, etc. The downside is that the * enumerator object is memory inefficient especially for iterating arrays.
 label: code-design
20285. repl string scan
20286. [key result] -> [result]
20287. '\xffTracedata'
20288. ... and its 'message' from an instance property
20289. AUTHORS.rst
20290. Potential direct eval call detected, flag the CALL * so that a run-time "direct eval" check is made and * special behavior may be triggered. Note that this * does not prevent 'eval' from being register bound.
20291. prefer jump
20292. **comment:** * Arithmetic, binary, and logical helpers. * * Note: there is no opcode for logical AND or logical OR; this is on * purpose, because the evalution order semantics for them make such * opcodes pretty pointless: short circuiting means they are most * comfortably implemented as jumps. However, a logical NOT opcode * is useful. * * Note: careful with duk_tval pointers here: they are potentially * invalidated by any DECREF and almost any API call. It's still * preferable to work without making a copy but that's not always * possible.
 label: code-design
20293. automatic length update

20294. args
20295. 'String'
20296. With ROM-based strings, heap->strs[] and thr->strs[] are omitted * so nothing to initialize for str[].
20297. * Note: prototype chain is followed BEFORE first comparison. This * means that the instanceof lval is never itself compared to the * rval.prototype property. This is apparently intentional, see E5 * Section 15.3.5.3, step 4.a. ** Also note: ** js>(function() {}) instanceof Function * true * js> Function instanceof Function * true ** For the latter, h_proto will be Function.prototype, which is the * built-in Function prototype. Because Function.[[Prototype]] is * also the built-in Function prototype, the result is true.
20298. not a vararg function
20299. Shouldn't happen but check anyway.
20300. E5 Sections 11.8.3, 11.8.5; x <= y --> not (x > y) --> not (y < x)
20301. Used by duk_emit*() calls so that we don't shuffle the loadints that * are needed to handle indirect opcodes.
20302. No strs[] pointer.
20303. **comment:** XXX: assertion that entries >= old_len are already unused
label: code-design
20304. e.g. DUK_OP_PREINCR
20305. eat 'while'
20306. nuke values at idx_rebase to get the first retval (initially * at idx_rcbase) to idx_rebase
20307. input radix
20308. char: use int cast
20309. result
20310. exponent digit
20311. **comment:** This is a cleaner approach and also produces smaller code than * the other alternative. Use duk_require_string() for format * safety (although the source property should always exist).
label: code-design
20312. defprop_flags
20313. s <- 2
20314. spaces (nargs - 1) + newline
20315. * Parse inner function
20316. if accessor without getter, return value is undefined
20317. **comment:** * For non-ASCII strings, we need to scan forwards or backwards * from some starting point. The starting point may be the start * or end of the string, or some cached midpoint in the string * cache. ** For "short" strings we simply scan without checking or updating * the cache. For longer strings we check and update the cache as * necessary, inserting a new cache entry if none exists.
label: code-design
20318. XXX: len >= 0x80000000 won't work below because we need to be * able to represent -len.
20319. who resumed us (if any)
20320. out_clamped==NULL -> RangeError if outside range
20321. relookup after possible realloc
20322. [...] regexp_object escaped_source bytecode]
20323. m+ <- (* m+ B)
20324. bytecode opcode (or extraop) for binary ops
20325. forget temp
20326. * Shared assignment expression handling * * args = (opcode << 8) + rbp * * If 'opcode' is DUK_OP_NONE, plain assignment without arithmetic. * Syntactically valid left-hand-side forms which are not accepted as * left-hand-side values (e.g. as in "f() = 1") must NOT cause a * SyntaxError, but rather a run-time ReferenceError. ** When evaluating X <op>= Y, the LHS (X) is conceptually evaluated * to a temporary first. The RHS is then evaluated. Finally, the * <op> is applied to the initial value of RHS (not the value after * RHS evaluation), and written to X. Doing so concretely generates * inefficient code so we'd like to avoid the temporary when possible. * See: <https://github.com/svaarala/duktape/pull/992>. ** The expression value (final LHS value, written to RHS) is * conceptually copied into a fresh temporary so that it won't * change even if the LHS/RHS values change in outer expressions. * For example, it'd be generally incorrect for the expression value * to be the RHS register binding, unless there's a guarantee that it * won't change during further expression evaluation. Using the * temporary concretely produces inefficient bytecode, so we try to * avoid the extra temporary for some known-to-be-safe cases. * Currently the only safe case we detect is a "top level assignment", * for example "x = y + z;", where the assignment expression value is * ignored. * See: test-dev-assign-expr.js and test-bug-assign-mutate-gh381.js.
20327. TRYCATCH, cannot emit now (not enough info)
20328. Value stack slot limits: these are quite approximate right now, and * because they overlap in control flow, some could be eliminated.
20329. absolute position for digit considered for rounding
20330. -> [...] this timeval_new]
20331. TypeError if wrong; class check, see E5 Section 15.10.6
20332. DUK_HOBJECT_FLAG_EXOTIC_DUKFUNC: omitted here intentionally
20333. * Byte order. Important to self check, because on some exotic platforms * there is no actual detection but rather assumption based on platform * defines.
20334. Both inputs are zero; cases where only one is zero can go * through main algorithm.
20335. same thread
20336. 'Boolean'
20337. * Inside one or more 'with' statements fall back to slow path always. * (See e.g. test-stmt-with.js.)
20338. * Init the heap thread
20339. XXX: need a duk_require_func_or_lfunc_coerce()
20340. avoid referencing, invalidated
20341. Don't support "straddled" source now.
20342. * Object compaction. * * Compaction is assumed to never throw an error.
20343. [arg1 ... argN this loggerLevel loggerName]
20344. duk_tval intentionally skipped
20345. * Since built-ins are not often extended, compact them.
20346. * Compilation
20347. Push a new ArrayBuffer (becomes view .buffer)
20348. idx_start
20349. Note: we rely on the _Varmap having a bunch of nice properties, like: * - being compacted and unmodified during this process * - not containing an array part * - having correct value types
20350. **comment:** XXX: return indication of "terminalness" (e.g. a 'throw' is terminal)
label: code-design
20351. may be NULL if no constants or inner funcs
20352. Equivalent year for DST calculations outside [1970,2038[range, see * E5 Section 15.9.1.8. Equivalent year has the same leap-year-ness and * starts with the same weekday on Jan 1. * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
20353. empty expressions can be detected conveniently with nud/led counts
20354. There are at most 7 args, but we use 8 here so that also * DUK_DATE_IDX_WEEKDAY gets initialized (to zero) to avoid the potential * for any Valgrind gripes later.
20355. [...] val callback thisArg val i obj]
20356. [...] this tracedata sep this str1 ... strN]
20357. important chosen base types
20358. Digit count indicates number of fractions (i.e. an absolute * digit index instead of a relative one). Used together with * DUK_N2S_FLAG_FIXED_FORMAT for toFixed().

20359. must also check refzero_list
20360. MEMBER/NEW/CALL EXPRESSIONS
20361. **comment:** XXX: add flag to indicate whether caller cares about return value; this * affects e.g. handling of assignment expressions. This change needs API * changes elsewhere too.
label: code-design
20362. Fast jumps (which avoid longjmp) jump directly to the jump sites * which are always known even while the iteration/switch statement * is still being parsed. A final peephole pass "straightens out" * the jumps.
20363. [... func]
20364. * Function gets no new environment when called. This is the * case for global code, indirect eval code, and non-strict * direct eval code. There is no direct correspondence to the * E5 specification, as global/eval code is not exposed as a * function.
20365. Wrappers for calling standard math library methods. These may be required * on platforms where one or more of the math built-ins are defined as macros * or inline functions and are thus not suitable to be used as function pointers.
20366. **comment:** Throwing an error this deep makes the error rather vague, but * saves hundreds of bytes of code.
label: code-design
20367. This is not strictly necessary, but avoids compiler warnings; e.g. * gcc won't reliably detect that no uninitialized data is read below.
20368. Trigger at zero or below
20369. Custom coercion for API
20370. **comment:** Note: cannot read more than 24 bits without possibly shifting top bits out. * Fixable, but adds complexity.
label: code-design
20371. Almost all global level Function objects are constructable * but not all: Function.prototype is a non-constructable, * callable Function.
20372. source ends before dest starts
20373. * Statement terminator check, including automatic semicolon * handling. After this step, 'curr_tok' should be the first * token after a possible statement terminator.
20374. * Exposed string-to-number API * * Input: [string] * Output: [number] * * If number parsing fails, a NaN is pushed as the result. If number parsing * fails due to an internal error, an InternalError is thrown.
20375. 64-bit OK because always ≥ 0
20376. reg/const for switch value
20377. **comment:** When formatting an argument to a string, errors may happen from multiple * causes. In general we want to catch obvious errors like a toLogString() * throwing an error, but we don't currently try to catch every possible * error. In particular, internal errors (like out of memory or stack) are * not caught. Also, we expect Error.toString() to not throw an error.
label: code-design
20378. Each round of finalizer execution may spawn new finalizable objects * which is normal behavior for some applications. Allow multiple * rounds of finalization, but use a shrinking limit based on the * first round to detect the case where a runaway finalizer creates * an unbounded amount of new finalizable objects. Finalizer rescue * is not supported: the semantics are unclear because most of the * objects being finalized here are already reachable. The finalizer * is given a boolean to indicate that rescue is not possible. * * See discussion in: <https://github.com/svaarala/duktape/pull/473>
20379. no prototype, updated below
20380. unwind to 'resume' caller
20381. 0xf0-0xff
20382. **comment:** XXX: missing trap result validation for non-configurable target keys * (must be present), for non-extensible target all target keys must be * present and no extra keys can be present. * <http://www.ecma-international.org/ecma-262/6.0/#sec-proxy-object-internal-methods-and-internal-slots-ownpropertykeys>
label: code-design
20383. **comment:** NOTE: lightfuncs are coerced to full functions because * lightfuncs don't fit into a property value slot. This * has some side effects, see test-dev-lightfunc-accessor.js.
label: code-design
20384. IdentityEscape, with dollar added as a valid additional * non-standard escape (see test-regexp-identity-escape-dollar.js). * Careful not to match end-of-buffer (<0) here.
20385. 1st related value (type specific)
20386. The caller is responsible for being sure that bytecode being loaded * is valid and trusted. Invalid bytecode can cause memory unsafe * behavior directly during loading or later during bytecode execution * (instruction validation would be quite complex to implement). * * This signature check is the only sanity check for detecting * accidental invalid inputs. The initial 0xFF byte ensures no * ordinary string will be accepted by accident.
20387. Shared object part
20388. always throw ReferenceError for unresolvable
20389. * Mark roots, hoping that recursion limit is not normally hit. * If recursion limit is hit, run additional reachability rounds * starting from "temproots" until marking is complete. * * Marking happens in two phases: first we mark actual reachability * roots (and run "temproots" to complete the process). Then we * check which objects are unreachable and are finalizable; such * objects are marked as FINALIZABLE and marked as reachability * (and "temproots" is run again to complete the process). * * The heap finalize_list must also be marked as a reachability root. * There may be objects on the list from a previous round if the * previous run had finalizer skip flag.
20390. Shift to sign extend.
20391. '^'
20392. required_desc_flags
20393. **comment:** Stack indices for better readability
label: code-design
20394. * Try registers
20395. no prototype or class yet
20396. shadowed; update value
20397. E5 Section 8.6.2 + custom classes
20398. **comment:** Lookup current thread; use the stable 'entry_thread' for this to * avoid clobber warnings. Any valid, reachable 'thr' value would be * fine for this, so using 'entry_thread' is just to silence warnings.
label: code-design
20399. RELATIONAL EXPRESSION
20400. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property existence check right now.
20401. These are just convenience "wiping" of state. Side effects should * not be an issue here: thr->heap and thr->heap->lj have a stable * pointer. Finalizer runs etc capture even out-of-memory errors so * nothing should throw here.
20402. Zero escape (also allowed in non-strict mode)
20403. roughly 64 bytes
20404. XXX: this doesn't actually work properly for tail calls, so * tail calls are disabled when DUK_USE_NONSTD_FUNC_CALLER_PROPERTY * is in use.
20405. * Dragon4 slow path (binary) digit generation. * An extra digit is generated for rounding.
20406. this is a bit approximate (errors out before max is reached); this is OK
20407. **comment:** * Parse a function-body-like expression (FunctionBody or Program * in E5 grammar) using a two-pass parse. The productions appear * in the following contexts: * * - function expression * - function statement * - function declaration * - getter in object literal * - setter in object literal * - global code * - eval code * - Function constructor body * * This function only parses the statement list of the body; the argument * list and possible function name must be initialized by the caller. * For instance, for Function constructor, the argument names are originally * on the value stack. The parsing of statements ends either at an EOF or * a closing brace; this is controlled by an input flag. * * Note that there are many differences affecting parsing and even code * generation: * * - Global and eval code have an implicit return value generated * by the last statement; function code does not * * - Global code, eval code, and Function constructor body end in * an EOF, other bodies in a closing brace ('}') * * Upon entry, 'curr_tok' is ignored and the function will pull in the * first token on its own. Upon exit, 'curr_tok' is the terminating * token (EOF or closing brace).
label: code-design
20408. When torture not enabled, can just use the same helper because * 'reg' won't get spilled.

20409. Note: actual update happens once write has been completed * without error below. The write should always succeed * from a specification viewpoint, but we may e.g. run out * of memory. It's safer in this order.

20410. Convert buffer to result string.

20411. default prototype (Note: 'thr' must be reachable)

20412. we know these because enum objects are internally created

20413. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
label: code-design

20414. **comment:** Direct eval requires that there's a current * activation and it is an Ecmascript function. * When Eval is executed from e.g. cooperate API * call we'll need to do an indirect eval instead.
label: code-design

20415. 1 = num actual 'return values'

20416. should never happen

20417. '-' verbatim

20418. Regexp tokens

20419. avoid attempt to compact the current object

20420. DUK_TOK_TRUE

20421. idx_bottom and idx_retval are only used for book-keeping of * Ecmascript-initiated calls, to allow returning to an Ecmascript * function properly. They are duk_size_t to match the convention * that value stack sizes are duk_size_t and local frame indices * are duk_idx_t.

20422. **comment:** This loop is optimized for size. For speed, there should be * two separate loops, and we should ensure that memcmp() can be * used without an extra "will searchstring fit" check. Doing * the preconditioning for 'p' and 'p_end' is easy but cpos * must be updated if 'p' is wound back (backward scanning).
label: code-design

20423. min(incl)

20424. may be equal

20425. traceback depth ensures fits into 16 bits

20426. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack. * * The right hand side could be a light function (as they generally * behave like objects). Light functions never have a 'prototype' * property so E5.1 Section 15.3.5.3 step 3 always throws a TypeError. * Using duk_require_hobject() is thus correct (except for error msg).

20427. DUK_TOK_GET

20428. * Various

20429. Module has now evaluated to a wrapped module function. Force its * .name to match module.name (defaults to last component of resolved * ID) so that it is shown in stack traces too. Note that we must not * introduce an actual name binding into the function scope (which is * usually the case with a named function) because it would affect the * scope seen by the module and shadow accesses to globals of the same name. * This is now done by compiling the function as anonymous and then forcing * its .name without setting a "has name binding" flag.

20430. Characters outside BMP cannot be escape()'d. We could * encode them as surrogate pairs (for codepoints inside * valid UTF-8 range, but not extended UTF-8). Because * escape() and unescape() are legacy functions, we don't.

20431. trailer

20432. * Check whether we need to abandon an array part (if it exists)

20433. [... env callee]

20434. **comment:** These is not 100% because format would need to be non-portable "long long". * Also print out as doubles to catch cases where the "long" type is not wide * enough; the limits will then not be printed accurately but the magnitude * will be correct.
label: code-design

20435. an Ecmascript function

20436. **comment:** avoid pressure to add/remove strings, invalidation of call data argument, etc.
label: code-design

20437. end of bytecode

20438. key is 'length', cannot match argument exotic behavior

20439. * First check whether property exists; if not, simple case. This covers * steps 1-4.

20440. * Do-while statement is mostly trivial, but there is special * handling for automatic semicolon handling (triggered by the * DUK__ALLOW_AUTO_SEMI_ALWAYS) flag related to a bug filed at: * * https://bugs.ecmascript.org/show_bug.cgi?id=8 * * See doc/compiler.rst for details.

20441. must be found: was found earlier, and cannot be inherited

20442. Input value should be on stack top and will be coerced and * popped. Refuse to update an Array's 'length' to a value * outside the 32-bit range. Negative zero is accepted as zero.

20443. shift in zeroes

20444. free object and all auxiliary (non-heap) allocs

20445. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Returns NULL for a lightfunc.

20446. bytes in dest

20447. stack[0] = regexp * stack[1] = string

20448. flags

20449. Check for maximum buffer length.

20450. Thread'

20451. **comment:** Pointer to bytecode executor's 'curr_pc' variable. Used to copy * the current PC back into the topmost activation when activation * state is about to change (or "syncing" is otherwise needed). This * is rather awkward but important for performance, see execution.rst.
label: code-design

20452. The Quote(value) operation: quote a string. * * Stack policy: [] -> [].

20453. Output #2: last component name

20454. tc1 = false, tc2 = false

20455. 0

20456. Exponent without a sign or with a +/- sign is accepted * by all call sites (even JSON.parse()).

20457. 0x90-0x9f

20458. **comment:** XXX: Shuffling support could be implemented here so that LDINT+LDINTX * would only shuffle once (instead of twice). The current code works * though, and has a smaller compiler footprint.
label: code-design

20459. * Write to entry part

20460. To use the shared helper need the virtual index.

20461. * Slow delete, but we don't care as we're already in a very slow path. * The delete always succeeds: key has no exotic behavior, property * is configurable, and no resize occurs.

20462. LAYOUT 1

20463. For len == 0, i is initialized to len - 1 which underflows. * The condition (i < len) will then exit the for-loop on the * first round which is correct. Similarly, loop termination * happens by i underflowing.

20464. Raw internal valstack access macros: access is unsafe so call site * must have a guarantee that the index is valid. When that is the case, * using these macro results in faster and smaller code than duk_get_tval(). * Both 'ctx' and 'idx' are evaluated multiple times, but only for asserts.

20465. * Node.js Buffer.isEncoding()

20466. array entries are all plain values

20467. **comment:** Buffer values are encoded in (lowercase) hex to make the * binary data readable. Base64 or similar would be more * compact but less readable, and the point of JX/JC * variants is to be as useful to a programmer as possible.
label: code-design

20468. **comment:** XXX: the helper currently assumes stack top contains new * 'length' value and the whole calling convention is not very * compatible with what we need.
label: code-design

20469. Coerce to number before validating pointers etc so that the * number coercions in duk_hbufferobject_validated_write() are * guaranteed to be side effect free and not invalidate the * pointer checks we do here.

20470. Mark-and-sweep interval is relative to combined count of objects and * strings kept in the heap during the latest mark-and-sweep pass. * Fixed point .8 multiplier and .0 adder. Trigger count (interval) is * decreased by each (re)allocation attempt (regardless of size), and each * refzero processed object. ** 'SKIP' indicates how many (re)allocations to wait until a retry if * GC is skipped because there is no thread do it with yet (happens * only during init phases).

20471. XXX: assert 'c' is an enumerator

20472. two special escapes: "\ and "", other printables as is

20473. for indexOf, ToInteger(undefined) would be 0, i.e. correct, but * handle both indexOf and lastIndexOf specially here.

20474. DUK_USE_AUGMENT_ERROR_THROW

20475. * Exponent notation for non-base-10 numbers isn't specified in EcmaScript * specification, as it never explicitly turns up: non-decimal numbers can * only be formatted with Number.prototype.toString([radix]) and for that, * behavior is not explicitly specified. ** Logical choices include formatting the exponent as decimal (e.g. binary * 100000 as 1e+5) or in current radix (e.g. binary 100000 as 1e+101). * The Dragon4 algorithm (in the original paper) prints the exponent value * in the target radix B. However, for radix values 15 and above, the * exponent separator 'e' is no longer easily parseable. Consider, for * instance, the number "1.faecee+1c".

20476. Template functions are not strictly constructable (they don't * have a "prototype" property for instance), so leave the * DUK_HOBJECT_FLAG_CONSTRUCTABLE flag cleared here.

20477. **comment:** Most practical strings will go here.
label: code-design

20478. non-strict eval: env is caller's env or global env (direct vs. indirect call) * global code: env is is global env

20479. **comment:** XXX: use the exactly same arithmetic function here as in executor
label: code-design

20480. -> [... lval new_rval]

20481. sep (even before first one)

20482. **comment:** XXX: use advancing pointers instead of index macros -> faster and smaller?
label: code-design

20483. 0x7F is special

20484. IEEE requires that zeros compare the same regardless * of their signed, so if both x and y are zeroes, they * are caught above.

20485. Signed integers always map to 4 bytes now.

20486. Resolve Proxy targets until Proxy chain ends. No explicit check for * a Proxy loop: user code cannot create such a loop without tweaking * internal properties directly.

20487. estimate is valid

20488. Note: it's nice if size is 2^N (not 4x4 = 16 bytes on 32 bit)

20489. always true, arg is unsigned

20490. ArrayBuffer: unlike any other argument variant, create * a view into the existing buffer.

20491. DUK_TOK_DO

20492. * Slow path: potentially requires function calls for coercion

20493. 'ArrayBuffer'

20494. Initialize index so that we skip internal control keys.

20495. flags have been already cleared

20496. * ArrayBuffer, DataView, and TypedArray constructors

20497. **comment:** * XXX: for now, indicate that an expensive catch binding * declarative environment is always needed. If we don't * need it, we don't need the const_varname either.
label: code-design

20498. -> [... this timeval_new timeval_new]

20499. DUK_TOK_CONST

20500. **comment:** XXX: unnecessary copying of values? Just set 'top' to * b + c, and let the return handling fix up the stack frame?
label: code-design

20501. Slot C

20502. **comment:** Array is dense and contains only strings, but ASIZE may * be larger than used part and there are UNUSED entries.
label: code-design

20503. At least: ... [err]

20504. wrapped

20505. whole or fraction digit

20506. * Constructor calls

20507. only 'length'

20508. DUK_USE_DEBUGGER_DUMPHEAP

20509. 'toISOString'

20510. 'res' contains expression value

20511. No other escape beginning with a digit in strict mode

20512. **comment:** Use Murmurhash2 directly for short strings, and use "block skipping" * for long strings: hash an initial part and then sample the rest of * the string with reasonably sized chunks. An initial offset for the * sampling is computed based on a hash of the initial part of the string; * this is done to (usually) avoid the case where all long strings have * certain offset ranges which are never sampled. ** Skip should depend on length and bound the total time to roughly * logarithmic. With current values: * * 1M string => 256 * 241 = 61696 bytes (0.06M) of hashing * 1G string => 256 * 16321 = 4178176 bytes (3.98M) of hashing ** XXX: It would be better to compute the skip offset more "smoothly" * instead of having a few boundary values.
label: code-design

20513. [key setter this val key] -> [key retval]

20514. point to start of 'reserved area'

20515. Shared helper for match() steps 3-4, search() steps 3-4.

20516. dummy

20517. Just call the "original" Object.defineProperties() to * finish up.

20518. Although these could be parsed as PatternCharacters unambiguously (here), * E5 Section 15.10.1 grammar explicitly forbids these as PatternCharacters.

20519. line terminator will be handled on next round

20520. * Debug dumping

20521. x in [2**31, 2**32[

20522. Error path.

20523. if a site already exists, nop: max one label site per statement

20524. Flag ORed to err_code to indicate __FILE__ / __LINE__ is not * blamed as source of error for error fileName / lineNumber.

20525. Start in paused state.

20526. **comment:** XXX: much to improve (code size)
label: code-design

20527. [regexp string res_arr]

20528. range conversion with a "skip"

20529. **comment:** Update function min/max line from current token. Needed to improve * function line range information for debugging, so that e.g. opening * curly brace is covered by line range even when no opcodes are emitted * for the line containing the brace.
label: code-design

20530. * Regular expression structs, constants, and bytecode defines.
20531. **comment:** should never be zero, because we (Duktape.Thread.yield) are on the stack
 label: code-design
20532. **comment:** XXX: This will now return false for non-numbers, even though they would * coerce to NaN (as a general rule). In particular, duk_get_number() * returns a NaN for non-numbers, so should this function also return * true for non-numbers?
 label: code-design
20533. **comment:** The handler is looked up with a normal property lookup; it may be an * accessor or the handler object itself may be a proxy object. If the * handler is a proxy, we need to extend the valstack as we make a * recursive proxy check without a function call in between (in fact * there is no limit to the potential recursion here). ** (For sanity, proxy creation rejects another proxy object as either * the handler or the target at the moment so recursive proxy cases * are not realized now.)
 label: code-design
20534. Flag handling currently assumes that flags are consistent. This is OK * because the call sites are now strictly controlled.
20535. DUK_TOK_DEFAULT
20536. [key value] or [key undefined]
20537. Pad significand with "virtual" zero digits so that Dragon4 will * have enough (apparent) precision to work with.
20538. * Set lexer input position and reinitialize lookup window.
20539. nargs
20540. With lightfuncs, act 'func' may be NULL
20541. return ('?:')
20542. DUK_USE_REFZERO_FINALIZER_TORTURE
20543. next to use, highest used is top - 1
20544. **comment:** XXX: actually single step levels would work just fine, clean up
 label: code-design
20545. * DUK_CALL_FLAG_IGNORE_RECLIMIT causes duk_handle_call() to ignore C * recursion depth limit (and won't increase it either). This is * dangerous, but useful because it allows the error handler to run * even if the original error is caused by C recursion depth limit. ** The heap level DUK_HEAP_FLAG_ERRHANDLER_RUNNING is set for the * duration of the error handler and cleared afterwards. This flag * prevents the error handler from running recursively. The flag is * heap level so that the flag properly controls even coroutines * launched by an error handler. Since the flag is heap level, it is * critical to restore it correctly. ** We ignore errors now: a success return and an error value both * replace the original error value. (This would be easy to change.)
20546. **comment:** XXX: remove heap->dbg_exec_counter, use heap->inst_count_interrupt instead?
 label: code-design
20547. DUK_USE_JSON_DECSTRING_FASTPATH
20548. The actual detached_cb call is postponed to message loop so * we don't need any special precautions here (just skip to EOM * on the already closed connection).
20549. arg2 would be clobbered so reassign it to a temp.
20550. standard behavior, step 3.f.i
20551. avoid degenerate cases, so that (len - 1) won't underflow
20552. Note: %06d for positive value, %07d for negative value to include * sign and 6 digits.
20553. 'let'
20554. * Halt execution helper
20555. 1110 xxxx; 3 bytes
20556. ch1 = (r_increment << 8) + byte
20557. variable name reg/const, if variable not register-bound
20558. Cast converts magic to 16-bit signed value
20559. * Remove the object from the refzero list. This cannot be done * before a possible finalizer has been executed; the finalizer * may trigger a mark-and-sweep, and mark-and-sweep must be able * to traverse a complete refzero_list.
20560. Argument variants. When the argument is an ArrayBuffer a view to * the same buffer is created; otherwise a new ArrayBuffer is always * created.
20561. entry part size
20562. * Property not found in prototype chain.
20563. function executes in strict mode
20564. 'data' is reachable through every compiled function which * contains a reference.
20565. Caller must trigger recomputation of active breakpoint list. To * ensure stale values are not used if that doesn't happen, clear the * active breakpoint list here.
20566. * Longjmp types, also double as identifying continuation type for a rethrow (in 'finally')
20567. note: long live range
20568. * Object will be kept; queue object back to heap_allocated (to tail)
20569. no change
20570. DUK_USE_ASSERTIONS
20571. force the property to 'undefined' to create a slot for it
20572. Optimized for speed.
20573. **comment:** prevent multiple in-progress detaches
 label: requirement
20574. Set a high interrupt counter; the original executor * interrupt invocation will rewrite before exiting.
20575. DUK_TOK_THIS
20576. index
20577. * Expression parsing: duk_expr_nud(), duk_expr_led(), duk_expr_lbp(), and helpers. ** - duk_expr_nud(): ("null denotation"): process prev_token as a "start" of an expression (e.g. literal) * - duk_expr_led(): ("left denotation"): process prev_token in the "middle" of an expression (e.g. operator) * - duk_expr_lbp(): ("left-binding power"): return left-binding power of curr_token
20578. 3-letter log level strings
20579. **comment:** * Convert char offset to byte offset ** Avoid using the string cache if possible: for ASCII strings byte and * char offsets are equal and for short strings direct scanning may be * better than using the string cache (which may evict a more important * entry). ** Typing now assumes 32-bit string byte/char offsets (duk_uint_fast32_t). * Better typing might be to use duk_size_t.
 label: code-design
20580. Must prevent finalizers which may have arbitrary side effects.
20581. DUK_EXCEPTION_H_INCLUDED
20582. invalidates h_buf pointer
20583. Special case: original qmin was zero so there is nothing * to repeat. Emit an atom copy but jump over it here.
20584. **comment:** Higher footprint, less churn.
 label: code-design
20585. -> [... holder name val new_elem]
20586. NUL term or -1 (EOF), NUL check would suffice
20587. core property functions
20588. 25%, i.e. less than 25% used -> abandon
20589. Allow empty fraction (e.g. "123.")
20590. 0xff...0xff
20591. don't allow const
20592. this is just beneath bottom
20593. * Duktape.Buffer: constructor
20594. 'Uint8Array'
20595. * Helpers for writing multiple properties
20596. exhaustive

20597. refcounting requires direct heap frees, which in turn requires a dual linked heap
20598. lastIndexOff() needs to be a vararg function because we must distinguish * between an undefined fromIndex and a "not given" fromIndex; indexOf() is * made vararg for symmetry although it doesn't strictly need to be.
20599. [...] error
20600. XXX: share final setting code for value and flags? difficult because * refcount code is different. Share entry allocation? But can't allocate * until array index checked.
20601. Terminating conditions. For fixed width output, we just ignore the * terminating conditions (and pretend that tc1 == tc2 == false). The * the current shortcut for fixed-format output is to generate a few * extra digits and use rounding (with carry) to finish the output.
20602. Return value when returning to this activation (points to caller * reg, not callee reg); index is absolute (only set if activation is * not topmost). * * Note: idx_bottom is always set, while idx_retval is only applicable * for activations below the topmost one. Currently idx_retval for * the topmost activation is considered garbage (and it not initialized * on entry or cleared on return; may contain previous or garbage * values).
20603. Unlike non-obsolete String calls, substr() algorithm in E5.1 * specification will happily coerce undefined and null to strings * ("undefined" and "null").
20604. DecimalEscape, only \0 is allowed, no leading zeroes are allowed
20605. parse new token
20606. 1e9
20607. Negative value checked so that a "time jump" works * reasonably. * * Same interval is now used for status sending and * peeking.
20608. * Misc shared helpers.
20609. don't set 'n' at all, inherited value is used as name
20610. ;'
20611. switch cmd
20612. **comment:** XXX: could add a fast path to process chunks of input codepoints, * but relative benefit would be quite small.
 label: code-design
20613. Copy interrupt counter/init value state to new thread (if any). * It's OK for new_thr to be the same as curr_thr.
20614. User totalLength overrides a computed length, but we'll check * every copy in the copy loop. Note that duk_to_uint() can * technically have arbitrary side effects so we need to recheck * the buffers in the copy loop.
20615. **comment:** XXX: some overlapping code; cleanup
 label: code-design
20616. stack[0] = searchElement * stack[1] = fromIndex * stack[2] = object * stack[3] = length (not needed, but not popped above)
20617. from curr pc
20618. * Maximum size check
20619. isError flag for yield
20620. * callstack_top - 1 --> this function * callstack_top - 2 --> caller (may not exist) * * If called directly from C, callstack_top might be 1. If calling * activation doesn't exist, call must be indirect.
20621. no need to unwind callstack
20622. * Array part
20623. **comment:** * Number should already be in NaN-normalized form, * but let's normalize anyway.
 label: code-design
20624. [array totalLength bufres buf]
20625. When src_size == 0, src_data may be NULL (if source * buffer is dynamic), and dst_data may be NULL (if * target buffer is dynamic). Avoid zero-size memcpy() * with an invalid pointer.
20626. [r1,r2] is the range
20627. * NormalizePropertyDescriptor() related helper. * * Internal helper which validates and normalizes a property descriptor * represented as an EcmaScript object (e.g. argument to defineProperty()). * The output of this conversion is a set of defprop_flags and possibly * some values pushed on the value stack; some subset of: property value, * getter, setter. Caller must manage stack top carefully because the * number of values pushed depends on the input property descriptor. * * The original descriptor object must not be altered in the process.
20628. duk_handle_call / duk_handle_safe_call recursion depth limiting
20629. DUK_HOBJECT_FLAG_NEWENV: handled below
20630. A leading digit is not required in some cases, e.g. accept ".123". * In other cases (JSON.parse()) a leading digit is required. This * is checked for after the loop.
20631. common case, already closed, so skip
20632. ignore millisecond fractions after 3
20633. * unshift()
20634. E5 Section 10.4.3
20635. No built-in functions are constructable except the top * level ones (Number, etc).
20636. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'd, * and we're in an infinite loop.
20637. Raw helper for getting a value from the stack, checking its tag. * The tag cannot be a number because numbers don't have an internal * tag in the packed representation.
20638. Bernstein hash init value is normally 5381
20639. * DECLVAR * * See E5 Sections: * 10.4.3 Entering Function Code * 10.5 Declaration Binding Instantiation * 12.2 Variable Statement * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * * Variable declaration behavior is mainly discussed in Section 10.5, * and is not discussed in the execution semantics (Sections 11-13). * * Conceptually declarations happen when code (global, eval, function) * is entered, before any user code is executed. In practice, register- * bound identifiers are 'declared' automatically (by virtue of being * allocated to registers with the initial value 'undefined'). Other * identifiers are declared in the function prologue with this primitive. * * Since non-register bindings eventually back to an internal object's * properties, the 'prop_flags' argument is used to specify binding * type: * * - Immutable binding: set DUK_PROPDESC_FLAG_WRITABLE to false * - Non-deletable binding: set DUK_PROPDESC_FLAG_CONFIGURABLE to false * - The flag DUK_PROPDESC_FLAG_ENUMERABLE should be set, although it * doesn't really matter for internal objects * * All bindings are non-deletable mutable bindings except: * * - Declarations in eval code (mutable, deletable) * - 'arguments' binding in strict function code (immutable) * - Function name binding of a function expression (immutable) * * Declarations may go to declarative environment records (always * so for functions), but may also go to object environment records * (e.g. global code). The global object environment has special * behavior when re-declaring a function (but not a variable); see * E5.1 specification, Section 10.5, step 5.e. * * Declarations always go to the 'top-most' environment record, i.e. * we never check the record chain. It's not an error even if a * property (even an immutable or non-deletable one) of the same name * already exists. * * If a declared variable already exists, its value needs to be updated * (if possible). Returns 1 if a PUTVAR needs to be done by the caller; * otherwise returns 0.
20640. **comment:** duk_handle_call_xxx: call ignores C recursion limit (for errhandler calls)
 label: code-design
20641. expt 0xffff is infinite/NaN
20642. Augment the error if called as a normal function. __FILE__ and __LINE__ * are not desirable in this case.
20643. %s
20644. e.g. "x" < "xx"
20645. **comment:** XXX: for fastints, could use a variant which assumes a double duk_tval * (and doesn't need to check for fastint again).
 label: code-design
20646. SameValue
20647. How much stack to require on entry to object/array encode
20648. * Parse code after the clause. Possible terminators are * 'case', 'default', and ')'. * * Note that there may be no code at all, not even an empty statement, * between case clauses. This must be handled just like an empty statement * (omitting seemingly pointless JUMPs), to avoid situations like * test-bug-case-fallthrough.js.
20649. ... name ': ' message
20650. to exit
20651. parenthesis count, 0 = top level
20652. * Useful Unicode codepoints * * Integer constants must be signed to avoid unexpected coercions * in comparisons.
20653. input string (may be a user pointer)

20654. * round_idx points to the digit which is considered for rounding; the * digit to its left is the final digit of the rounded value. If round_idx * is zero, rounding will be performed; the result will either be an empty * rounded value or if carry happens a '1' digit is generated.

20655. Careful with carry condition: * - If carry not added: $0x12345678 + 0 + 0xffffffff = 0x12345677$ ($< 0x12345678$) * - If carry added: $0x12345678 + 1 + 0xffffffff = 0x12345678$ ($= 0x12345678$)

20656. shared helper

20657. * Fast buffer writer with spare management.

20658. success/error path both do this

20659. DUK_TOK_MUL

20660. attempt to change from accessor to data property

20661. allowed ascii whitespace

20662. [... source? filename?]

20663. continue jump

20664. -> [... key val]

20665. Prepare value stack for a method call through an object property. * May currently throw an error e.g. when getting the property.

20666. no need to create environment record now; leave as NULL

20667. * Thread state check and book-keeping.

20668. "/=" and not in regexp mode

20669. for lastIndexOf, result may be -1 (mark immediate termination)

20670. fatal_func should be noreturn, but noreturn declarations on function * pointers has a very spotty support apparently so it's not currently * done.

20671. requested identifier

20672. Function expression. Note that any statement beginning with 'function' * is handled by the statement parser as a function declaration, or a * non-standard function expression/statement (or a SyntaxError). We only * handle actual function expressions (occurring inside an expression) here. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().

20673. * JSON.stringify() fast path * * Otherwise supports full JSON, JX, and JC features, but bails out on any * possible side effect which might change the value being serialized. The * fast path can take advantage of the fact that the value being serialized * is unchanged so that we can walk directly through property tables etc.

20674. **comment:** XXX: helper
label: code-design

20675. no return value -> don't replace created value

20676. 'Arguments' object and has arguments exotic behavior (non-strict callee)

20677. [requested_id require require.id resolved_id last_comp Duktape.modLoaded Duktape.modLoaded[id]]

20678. Note: left shift, should mask

20679. 31 to 36

20680. two level break

20681. don't allow an empty match at the end of the string

20682. 'thr' is now reachable

20683. Parse argument list. Arguments are written to temps starting from * "next temp". Returns number of arguments parsed. Expects left paren * to be already eaten, and eats the right paren before returning.

20684. Allow 'Infinity'

20685. Opcode slot C is used in a non-standard way, so shuffling * is not allowed.

20686. * Helpers for managing property storage size

20687. if-digit-else-ctrl

20688. Safe write calls which will ensure space first.

20689. having this as a separate function provided a size benefit

20690. code emission

20691. **comment:** * obj->props is intentionally left as NULL, and duk_hobject_props.c must deal * with this properly. This is intentional: empty objects consume a minimum * amount of memory. Further, an initial allocation might fail and cause * 'obj' to "leak" (require a mark-and-sweep) since it is not reachable yet.
label: code-design

20692. Term was '.', backtrack resolved name by one component. * q[-1] = previous slash (or beyond start of buffer) * q[-2] = last char of previous component (or beyond start of buffer)

20693. **comment:** This could also be thrown internally (set the error, goto check_longjmp), * but it's better for internal errors to bubble outwards so that we won't * infinite loop in this catchpoint.
label: code-design

20694. * Debugging related API calls

20695. -> [... func this arg1 ... argN _Args length]

20696. 'Float64Array'

20697. 1 1 1 <32 bits> * Encode in two parts to avoid bitencode 24-bit limitation

20698. -> [this]

20699. **comment:** Note: array entries are always writable, so the writability check * above is pointless for them. The check could be avoided with some * refactoring but is probably not worth it.
label: code-design

20700. retval indicates delete failed

20701. formatters always get one-based month/day-of-month

20702. continue matching, set neg_tzoffset flag

20703. bound function chain has already been resolved

20704. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property get right now.

20705. standard JSON; array gaps

20706. Note: array part values are [[Writable]], [[Enumerable]], * and [[Configurable]] which matches the required attributes * here.

20707. 0=before period/exp, * 1=after period, before exp * 2=after exp, allow '+' or '-' * 3=after exp and exp sign

20708. just check 'env'

20709. **comment:** XXX: primitive to make array from valstack slice
label: code-design

20710. **comment:** * Dragon4 setup. * * Convert double from IEEE representation for conversion; * normal finite values have an implicit leading 1-bit. The * slow path algorithm doesn't handle zero, so zero is special * cased here but still creates a valid nc_ctx, and goes * through normal formatting in case special formatting has * been requested (e.g. forced exponential format: 0 -> "0e+0").
label: code-design

20711. consistency requires this

20712. DUK_TOK_CASE

20713. Nothing to initialize, str[] is in ROM.

20714. currently 6 characters of lookup are actually needed (duk_lexer.c)

20715. index/length check guarantees

20716. left.x1 -> res.x1

20717. * Case conversion tables generated using src/extract_caseconv.py.

20718. is global code

20719. **comment:** * Macros to set a duk_tval and update refcount of the target (decref the * old value and incref the new value if necessary). This is both performance * and footprint critical; any changes made should be measured for size/speed.
label: code-design

20720. We don't duk_require_stack() here now, but rely on the caller having * enough space.

20721. * Regexp executor. ** Safety: the EcmaScript executor should prevent user from reading and * replacing regexp bytecode. Even so, the executor must validate all * memory accesses etc. When an invalid access is detected (e.g. a 'save' * opcode to invalid, unallocated index) it should fail with an internal * error but not cause a segmentation fault. ** Notes: * - Backtrack counts are limited to unsigned 32 bits but should * technically be duk_size_t for strings longer than 4G chars. * This also requires a regexp bytecode change.

20722. Note: not a valid stack index if num_stack_args == 0

20723. * Convenience (independent of representation)

20724. no refcounting

20725. duk_handle_ecma_call_setup: setup for a resume()

20726. escape any non-ASCII characters

20727. if 0, 'str' used, if > 0, 'strlist' used

20728. These limits are based on bytecode limits. Max temps is limited * by duk_hcompiledfunction nargs/nregs fields being 16 bits.

20729. * pop(), push()

20730. 'DecEnv'

20731. number of elements guaranteed to be user accessible * (in addition to call arguments) on Duktape/C function entry.

20732. NOTE: The Array special behaviors are NOT invoked by duk_xdef_prop_index() * (which differs from the official algorithm). If no error is thrown, this * doesn't matter as the length is updated at the end. However, if an error * is thrown, the length will be unset. That shouldn't matter because the * caller won't get a reference to the intermediate value.

20733. Torture option to shake out finalizer side effect issues: * make a bogus function call for every finalizable object, * essentially simulating the case where everything has a * finalizer.

20734. [... key]

20735. **comment:** This seems faster than emitting bytes one at a time and * then potentially rewinding.
label: code-design

20736. * Writability check

20737. may be changed by call

20738. DUK_USE_NONSTD_FUNC_CALLER_PROPERTY

20739. DUK_USE_JSON_STRINGIFY_FASTPATH

20740. Update all labels with matching label_id.

20741. guaranteed by duk_to_string()

20742. already declared, must update binding value

20743. DUK_USE_TRACEBACKS

20744. **comment:** Formatting function pointers is tricky: there is no standard pointer for * function pointers and the size of a function pointer may depend on the * specific pointer type. This helper formats a function pointer based on * its memory layout to get something useful on most platforms.
label: code-design

20745. ensures callstack_top - 1 >= 0

20746. Count free operations toward triggering a GC but never actually trigger * a GC from a free. Otherwise code which frees internal structures would * need to put in NULLs at every turn to ensure the object is always in * consistent state for a mark-and-sweep.

20747. detached

20748. 'constructor'

20749. disabled for now

20750. idx_replacer

20751. type to represent a straight register reference, with <0 indicating none

20752. continue jump not patched, an INVALID opcode remains there

20753. -> [... enum_target res trap_result]

20754. current (next) array index

20755. fmin() with args -0 and +0 is not guaranteed to return * -0 as EcmaScript requires.

20756. 'const'

20757. all codepoints up to U+10FFFF

20758. **comment:** This is performance critical because it's needed for every DECREF. * Take advantage of the fact that the first heap allocated tag is 8, * so that bit 3 is set for all heap allocated tags (and never set for * non-heap-allocated tags).
label: code-design

20759. if no NaN handling flag, may still be NaN here, but not Inf

20760. thr argument only used for thr->heap, so specific thread doesn't matter

20761. Preshifted << 4. Must use 16-bit entry to allow negative value signaling.

20762. [0x00000000, 0xffffffff]

20763. fall through to error

20764. [... key_obj key key]

20765. Specification stripPrefix maps to DUK_S2N_FLAG_ALLOW_AUTO_HEX_INT. ** Don't autodetect octals (from leading zeroes), require user code to * provide an explicit radix 8 for parsing octal. See write-up from Mozilla: * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt#ECMAScript_5_Removes_Octal_Interpretation

20766. Process messages until we're no longer paused or we peek * and see there's nothing to read right now.

20767. XXX: callstack unwind may now throw an error when closing * scopes; this is a sandboxing issue, described in: * <https://github.com/svaarala/duktape/issues/476>

20768. since no recursive error handler calls

20769. **comment:** XXX: other properties of function instances; 'arguments', 'caller'.
label: code-design

20770. numeric label_id (-1 reserved as marker)

20771. irrelevant when out->value == NULL

20772. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed.

20773. **comment:** * Helper to sort array index keys. The keys are in the enumeration object * entry part, starting from DUK_ENUM_START_INDEX, and the entry part is dense. ** We use insertion sort because it is simple (leading to compact code,) * works nicely in-place, and minimizes operations if data is already sorted * or nearly sorted (which is a very common case here). It also minimizes * the use of element comparisons in general. This is nice because element * comparisons here involve re-parsing the string keys into numbers each * time, which is naturally very expensive. ** Note that the entry part values are all "true", e.g. * * "1" -> true, "3" -> true, "2" -> true * * so it suffices to only work in the key part without exchanging any keys, * simplifying the sort. * *
http://en.wikipedia.org/wiki/Insertion_sort * * (Compiles to about 160 bytes now as a stand-alone function.)
label: code-design

20774. The function object is now reachable and refcounts are fine, * so we can pop off all the temporaries.

20775. Care must be taken to avoid pointer wrapping in the index * validation. For instance, on a 32-bit platform with 8-byte * duk_tval the index 0x20000000UL would wrap the memory space * once.

20776. Convert a duk_tval fastint (caller checks) to a 32-bit index.

20777. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2016 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

20778. Insert bytes in the middle of the buffer from a slice already * in the buffer. Source offset is interpreted "before" the operation.

20779. DUK_USE_STRTAB_CHAIN
20780. Original function instance/template had NAMEBINDING. * Must create a lexical environment on loading to allow * recursive functions like 'function foo() { foo(); }'.
20781. **comment:** the message should be a compile time constant without formatting (less risk); * we don't care about assertion text size because they're not used in production * builds.
label: code-design
20782. assume value is a number
20783. **comment:** combines steps 3, 6; step 7 is not needed
label: requirement
20784. * Debugger support
20785. **comment:** XXX: spare handling, slow now
label: code-design
20786. normal and constructor calls have identical semantics
20787. * A token is interpreted as any possible production of InputElementDiv * and InputElementRegExp, see E5 Section 7 in its entirety. Note that * the E5 "Token" production does not cover all actual tokens of the * language (which is explicitly stated in the specification, Section 7.5). * Null and boolean literals are defined as part of both ReservedWord * (E5 Section 7.6.1) and Literal (E5 Section 7.8) productions. Here, * null and boolean values have literal tokens, and are not reserved * words. * * Decimal literal negative/positive sign is -not- part of DUK_TOK_NUMBER. * The number tokens always have a non-negative value. The unary minus * operator in "-1.0" is optimized during compilation to yield a single * negative constant. * * Token numbering is free except that reserved words are required to be * in a continuous range and in a particular order. See genstrings.py.
20788. Same coercion behavior as for Number.
20789. the outer loop will recheck and exit
20790. **comment:** These are macros for now, but could be separate functions to reduce code * footprint (check call site count before refactoring).
label: code-design
20791. x == -Infinity
20792. [... source? func_template]
20793. other call
20794. lastIndex already set up for next match
20795. * Macros to access the 'props' allocation.
20796. **comment:** * Struct size/alignment if platform requires it * * There are some compiler specific struct padding pragmas etc in use, this * selftest ensures they're correctly detected and used.
label: code-design
20797. [... source? filename?] (depends on flags)
20798. ToUint16() coercion is mandatory in the E5.1 specification, but * this non-compliant behavior makes more sense because we support * non-BMP codepoints. Don't use CESU-8 because that'd create * surrogate pairs.
20799. match string
20800. [targetBuffer targetStart sourceStart sourceEnd]
20801. x = sign(x) * floor(abs(x)), i.e. truncate towards zero, keep sign
20802. **comment:** XXX: here again finalizer thread is the heap_thread which needs * to be coordinated with finalizer thread fixes.
label: code-design
20803. at least one opcode emitted
20804. Slice offsets are element (not byte) offsets, which only matters * for TypedArray views, Node.js Buffer and ArrayBuffer have shift * zero so byte and element offsets are the same. Negative indices * are counted from end of slice, crossed indices are allowed (and * result in zero length result), and final values are clamped * against the current slice. There's intentionally no check * against the underlying buffer here.
20805. side effects, in theory (referenced by global env)
20806. * Object.seal() and Object.freeze() (E5 Sections 15.2.3.8 and 15.2.3.9) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: virtual (non-concrete) properties which are non-configurable but * writable would pose some problems, but such properties do not currently * exist (all virtual properties are non-configurable and non-writable). * If they did exist, the non-configurability does NOT prevent them from * becoming non-writable. However, this change should be recorded somehow * so that it would turn up (e.g. when getting the property descriptor), * requiring some additional flags in the object.
20807. additional flags which are passed in the same flags argument as property * flags but are not stored in object properties.
20808. **comment:** XXX: Is this INCREF necessary? 'func' is always a borrowed * reference reachable through the value stack? If changed, stack * unwind code also needs to be fixed to match.
label: code-design
20809. **comment:** XXX: shared decoding of 'b' and 'c'?
label: code-design
20810. **comment:** The fast path for array property put is not fully compliant: * If one places conflicting number-indexed properties into * Array.prototype (for example, a non-writable Array.prototype[7]) * the fast path will incorrectly ignore them. * * This fast path could be made compliant by falling through * to the slow path if the previous value was UNUSED. This would * also remove the need to check for extensibility. Right now a * non-extensible array is slower than an extensible one as far * as writes are concerned. * * The fast path behavior is documented in more detail here: * tests/ecmascript/test-misc-array-fast-write.js
label: code-design
20811. **comment:** XXX: because we unwind stacks above, thr->heap->curr_thread is at * risk of pointing to an already freed thread. This was indeed the * case in test-bug-multithread-valgrind.c, until duk_handle_call() * was fixed to restore thr->heap->curr_thread before rethrowing an * uncaught error.
label: code-design
20812. Allow naked fraction (e.g. ".123")
20813. * Rate limit check for sending status update or peeking into * the debug transport. Both can be expensive operations that * we don't want to do on every opcode. * * Making sure the interval remains reasonable on a wide variety * of targets and bytecode is difficult without a timestamp, so * we use a Date-provided timestamp for the rate limit check. * But since it's also expensive to get a timestamp, a bytecode * counter is used to rate limit getting timestamps.
20814. >>> struct.unpack('>d', '4000112233445566'.decode('hex')) * (2.008366013071895)
20815. Other longjmp types are handled by executor before propagating * the error here.
20816. * E5 Section 11.8.6 describes the main algorithm, which uses * [[HasInstance]]. [[HasInstance]] is defined for only * function objects: * * - Normal functions: * E5 Section 15.3.5.3 * - Functions established with Function.prototype.bind(): * E5 Section 15.3.4.5.3 * * For other objects, a TypeError is thrown. * * Limited Proxy support: don't support 'getPrototypeOf' trap but * continue lookup in Proxy target if the value is a Proxy.
20817. * Ecmascript/native function call or lightfunc call
20818. Halt execution and enter a debugger message loop until execution is resumed * by the client. PC for the current activation may be temporarily decremented * so that the "current" instruction will be shown by the client. This helper * is callable from anywhere, also outside bytecode executor.
20819. computed live
20820. **comment:** Load a function from bytecode. The function object returned here must * match what is created by duk_js_push_closure() with respect to its flags, * properties, etc. * * NOTE: there are intentionally no input buffer length / bound checks. * Adding them would be easy but wouldn't ensure memory safety as untrusted * or broken bytecode is unsafe during execution unless the opcodes themselves * are validated (which is quite complex, especially for indirect opcodes).
label: code-design
20821. * For an external string, the NUL-terminated string data is stored * externally. The user must guarantee that data behind this pointer * doesn't change while it's used.
20822. Note: masking of 'x' is not necessary because of * range check and shifting -> no bits overlapping * the marker should be set.
20823. **comment:** * Object.defineProperty() related helper (E5 Section 15.2.3.6) * * Inlines all [[DefineOwnProperty]] exotic behaviors. * * Note: Ecmascript compliant [[DefineOwnProperty]](P, Desc, Throw) is not * implemented directly, but Object.defineProperty() serves its purpose. * We don't need the [[DefineOwnProperty]] internally and we don't have a * property descriptor with 'missing values' so it's easier to avoid it * entirely. * * Note: this is only called for actual objects, not primitive values. * This must support virtual properties for full objects (e.g. Strings) * but not for plain values (e.g. strings). Lightfuncs, even though * primitive in a sense, are treated like objects and accepted as target * values.

label: code-design

20824. callstack limits
20825. A...P
20826. Module table: * - module.exports: initial exports table (may be replaced by user) * - module.id is non-writable and non-configurable, as the CommonJS * spec suggests this if possible * - module.filename: not set, defaults to resolved ID if not explicitly * set by modSearch() (note capitalization, not .fileName, matches Node.js) * - module.name: not set, defaults to last component of resolved ID if * not explicitly set by modSearch()
20827. String constructor needs to distinguish between an argument not given at all * vs. given as 'undefined'. We're a vararg function to handle this properly.
20828. Explicit length is only needed if it differs from 'nargs'.
20829. Force restart caused by a function return; must recheck * debugger breakpoints before checking line transitions, * see GH-303. Restart and then handle interrupt_counter * zero again.
20830. Decode 'bits' bits from the input stream (bits must be 1...24). * When reading past bitstream end, zeroes are shifted in. The result * is signed to match duk_bd_decode_flagged.
20831. (?:
20832. Must be restored here to handle e.g. yields properly.
20833. The base ID of the current require() function, resolution base
20834. non-Reference deletion is always 'true', even in strict mode
20835. jump to next loop, using reg_v34_iter as iterated value
20836. Scan error: this shouldn't normally happen; it could happen if * string is not valid UTF-8 data, and clen/blen are not consistent * with the scanning algorithm.
20837. max possible
20838. punctuators (unlike the spec, also includes "/" and "/=")
20839. DUK_BUFOBJ_FLOAT32ARRAY
20840. [... ptr]
20841. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property delete right now.
20842. 12: getDate
20843. * Helper for valstack space ** Caller of DUK_ASSERT_VALSTACK_SPACE() estimates the number of free stack entries * required for its own use, and any child calls which are not (a) Duktape API calls * or (b) Duktape calls which involve extending the valstack (e.g. getter call).
20844. 'modLoaded'
20845. * Assertions before
20846. **comment:** * Buffer writer (dynamic buffer only) ** Helper for writing to a dynamic buffer with a concept of a "spare" area * to reduce resizes. You can ensure there is enough space beforehand and * then write for a while without further checks, relying on a stable data * pointer. Spare handling is automatic so call sites only indicate how * much data they need right now. ** There are several ways to write using bufwriter. The best approach * depends mainly on how much performance matters over code footprint. * The key issues are (1) ensuring there is space and (2) keeping the 'p' pointers consistent. Fast code should ensure space for multiple writes * with one ensure call. Fastest inner loop code can temporarily borrow * the 'p' pointer but must write it back eventually. * Be careful to ensure all macro arguments (other than static pointers like * 'thr' and 'bw_ctx') are evaluated exactly once, using temporaries if * necessary (if that's not possible, there should be a note near the macro). * Buffer write arguments often contain arithmetic etc so this is * particularly important here.
label: code-design
20847. Ordering should not matter (E5 Section 11.8.5, step 3.a) but * preserve it just in case.
20848. for zero size, don't push anything on valstack
20849. Here we'd have the option of decoding unpadded base64 * (e.g. "xxxxyy" instead of "xxxxyy==". Currently not * accepted.
20850. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
20851. * Conceptually, we look for the identifier binding by starting from * 'env' and following to chain of environment records (represented * by the prototype chain). ** If 'env' is NULL, the current activation does not yet have an * allocated declarative environment record; this should be treated * exactly as if the environment record existed but had no bindings * other than register bindings. ** Note: we assume that with the DUK_HOBJECT_FLAG_NEWENV cleared * the environment will always be initialized immediately; hence * a NULL 'env' should only happen with the flag set. This is the * case for: (1) function calls, and (2) strict, direct eval calls.
20852. heap level "stash" object (e.g., various reachability roots)
20853. * For global or eval code this is straightforward. For functions * created with the Function constructor we only get the source for * the body and must manufacture the "function ..." part. ** For instance, for constructed functions (v8): * * > a = new Function("foo", "bar", "print(foo)"); * [Function] * > a.toString() * 'function anonymous(foo,bar) {\nprint(foo)\n}' * Similarly for e.g. getters (v8): * * > x = { get a(foo,bar) { print(foo); } } * { a: [Getter] } * > Object.getOwnPropertyDescriptor(x, 'a').get.toString() * 'function a(foo,bar) { print(foo); }'
20854. c recursion check
20855. XXX: optimize by adding the token numbers directly into the * always interned duk_hstring objects (there should be enough * flag bits free for that)?
20856. Continue parsing after padding, allows concatenated, * padded base64.
20857. Check that 'this' is a duk_hbufferobject and return a pointer to it * (NULL if not).
20858. idx_step is +1 for indexOf, -1 for lastIndexOf
20859. is a setter/getter
20860. env created above to stack top
20861. cannot be e.g. arguments exotic, since exotic 'traits' are mutually exclusive
20862. DUK_TOK_LNOT
20863. XXX: sufficient to check 'strict', assert for 'is function'
20864. check final character validity
20865. * Init lexer context
20866. * Inref and decref functions. ** Decref may trigger immediate refzero handling, which may free and finalize * an arbitrary number of objects. *
20867. Generate pc2line data for an instruction sequence, leaving a buffer on stack top.
20868. a dummy undefined value is pushed to make valstack * behavior uniform for caller
20869. idx_step is +1 for reduce, -1 for reduceRight
20870. PatternCharacter, all excluded characters are matched by cases above
20871. Note: -nargs alone would fail for nargs == 0, this is OK
20872. curr and desc are accessors
20873. * Throw the error in the resumed thread's context; the * error value is pushed onto the resumee valstack. ** Note: the callstack of the target may empty in this case * too (i.e. the target thread has never been resumed). The * value stack will contain the initial function in that case, * which we simply ignore.
20874. For string-to-number, pretend we never have the lowest mantissa as there * is no natural "precision" for inputs. Having lowest_mantissa == 0, we'll * fall into the base cases for both e >= 0 and e < 0.
20875. Number built-in accepts a plain number or a Number object (whose * internal value is operated on). Other types cause TypeError.
20876. Encode to extended UTF-8; 'out' must have space for at least * DUK_UNICODE_MAX_XUTF8_LENGTH bytes. Allows encoding of any * 32-bit (unsigned) codepoint.
20877. func.prototype.constructor = func
20878. Accept any pointer-like value; for 'object' dvalue, read * and ignore the class number.
20879. **comment:** Clone the properties of the ROM-based global object to create a * fully RAM-based global object. Uses more memory than the inherit * model but more compliant.
label: code-design
20880. prev value can be garbage, no decref
20881. **comment:** * For/for-in statement is complicated to parse because * determining the statement type (three-part for vs. a * for-in) requires potential backtracking. ** See the helper for the messy stuff.
label: code-design
20882. Stepped? Step out is handled by callstack unwind.
20883. Copy values, the copy method depends on the arguments. ** Copy mode decision may depend on the validity of the underlying * buffer of the source argument; there must be no harmful side effects * from there to here for copy_mode to still be valid.

20884. fills window
20885. Caller must check character offset to be inside the string.
20886. valstack will be unbalanced, which is OK
20887. conservative
20888. unconditional
20889. DUK_USE_FASTINT && DUK_USE_PACKED_TVAL
20890. * NEXTENUM checks whether the enumerator still has unenumerated * keys. If so, the next key is loaded to the target register * and the next instruction is skipped. Otherwise the next instruction * will be executed, jumping out of the enumeration loop.
20891. **comment:** Providing access to e.g. act->lex_env would be dangerous: these * internal structures must never be accessible to the application. * Duktape relies on them having consistent data, and this consistency * is only asserted for, not checked for.
label: code-design
20892. Duktape.modSearch
20893. Set .buffer to the argument ArrayBuffer.
20894. * Parse a function-like expression: * - function expression * - function declaration * - function statement (non-standard) * - setter/getter * * Adds the function to comp_ctx->curr_func function table and returns the * function number. * * On entry, curr_token points to: * - the token after 'function' for function expression/declaration/statement * - the token after 'set' or 'get' for setter/getter
20895. * Match loop. * * Try matching at different offsets until match found or input exhausted.
20896. flags: ""
20897. -> [res_obj]
20898. nbytes * <-----> * [... | p | x | x | q] * => [... | q | p | x | x]
20899. eat '{' on entry
20900. This native function is also used for Date.prototype.getTime() * as their behavior is identical.
20901. resume write to target
20902. if new_size < L * old_size, resize without abandon check; L = 3-bit fixed point, e.g. 9 -> 9/8 = 112.5%
20903. Note: we could try to stuff a partial hash (e.g. 16 bits) into the * shared heap header. Good hashing needs more hash bits though.
20904. Else functionality is identical for function call and constructor * call.
20905. inclusive
20906. Check that 'this' is a duk_hbufferobject and return a pointer to it.
20907. utf-8 continuation bytes have the form 10xx xxxx
20908. This is based on V8 EquivalentYear() algorithm (see src/genequivyear.py): * <http://code.google.com/p/v8/source/browse/trunk/src/date.h#146>
20909. 'buf' contains the string to write, 'sz_buf' contains the length * (which may be zero).
20910. '
20911. hobject management functions
20912. Convert day from one-based to zero-based (internal). This may * cause the day part to be negative, which is OK.
20913. 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [26 bits]
20914. There is no need to check the first character specially here * (i.e. reject digits): the caller only accepts valid initial * characters and won't call us if the first character is a digit. * This also ensures that the plain string won't be empty.
20915. '\xffBytecode'
20916. Catch a double-to-int64 cast issue encountered in practice.
20917. * DumpHeap command
20918. **comment:** * Lexer for source files, ToNumber() string conversions, RegExp expressions, * and JSON. * * Provides a stream of Ecmascript tokens from an UTF-8/CESU-8 buffer. The * caller can also rewind the token stream into a certain position which is * needed by the compiler part for multi-pass scanning. Tokens are * represented as duk_token structures, and contain line number information. * Token types are identified with DUK_TOK_* defines. * * Characters are decoded into a fixed size lookup window consisting of * decoded Unicode code points, with window positions past the end of the * input filled with an invalid codepoint (-1). The tokenizer can thus * perform multiple character lookups efficiently and with few sanity * checks (such as access outside the end of the input), which keeps the * tokenization code small at the cost of performance. * * Character data in tokens, such as identifier names and string literals, * is encoded into CESU-8 format on-the-fly while parsing the token in * question. The string data is made reachable to garbage collection by * placing the token-related values in value stack entries allocated for * this purpose by the caller. The characters exist in Unicode code point * form only in the fixed size lookup window, which keeps character data * expansion (of especially ASCII data) low. * * Token parsing supports the full range of Unicode characters as described * in the E5 specification. Parsing has been optimized for ASCII characters * because ordinary Ecmascript code consists almost entirely of ASCII * characters. Matching of complex Unicode codepoint sets (such as in the * IdentifierStart and IdentifierPart productions) is optimized for size, * and is done using a linear scan of a bit-packed list of ranges. This is * very slow, but should never be entered unless the source code actually * contains Unicode characters. * * Ecmascript tokenization is partially context sensitive. First, * additional future reserved words are recognized in strict mode (see E5 * Section 7.6.1.2). Second, a forward slash character ('/') can be * recognized either as starting a RegExp literal or as a division operator, * depending on context. The caller must provide necessary context flags * when requesting a new token. * * Future work: * * * Make line number tracking optional, as it consumes space. * * * Add a feature flag for disabling UTF-8 decoding of input, as most * source code is ASCII. Because of Unicode escapes written in ASCII, * this does not allow Unicode support to be removed from e.g. * duk_unicode_is_identifier_start() nor does it allow removal of CESU-8 * encoding of e.g. string literals. * * * Add a feature flag for disabling Unicode compliance of e.g. identifier * names. This allows for a build more than a kilobyte smaller, because * Unicode ranges needed by duk_unicode_is_identifier_start() and * duk_unicode_is_identifier_part() can be dropped. String literals * should still be allowed to contain escaped Unicode, so this still does * not allow removal of CESU-8 encoding of e.g. string literals. * * * Character lookup tables for codepoints above BMP could be stripped. * * * Strictly speaking, E5 specification requires that source code consists * of 16-bit code units, and if not, must be conceptually converted to * that format first. The current lexer processes Unicode code points * and allows characters outside the BMP. These should be converted to * surrogate pairs while reading the source characters into the window, * not after tokens have been formed (as is done now). However, the fix * is not trivial because two characters are decoded from one codepoint. * * * Optimize for speed as well as size. Large if-else ladders are (at * least potentially) slow.
label: code-design
20919. [... buf loop]
20920. **comment:** * Heap object representation. * * Heap objects are used for Ecmascript objects, arrays, and functions, * but also for internal control like declarative and object environment * records. Compiled functions, native functions, and threads are also * objects but with an extended C struct. * * Objects provide the required Ecmascript semantics and exotic behaviors * especially for property access. * * Properties are stored in three conceptual parts: * * 1. A linear 'entry part' contains ordered key-value-attributes triples * and is the main method of string properties. * * 2. An optional linear 'array part' is used for array objects to store a * (dense) range of [0,N[array indexed entries with default attributes * (writable, enumerable, configurable). If the array part would become * sparse or non-default attributes are required, the array part is * abandoned and moved to the 'entry part'. * * 3. An optional 'hash part' is used to optimize lookups of the entry * part; it is used only for objects with sufficiently many properties * and can be abandoned without loss of information. * * These three conceptual parts are stored in a single memory allocated area. * This minimizes memory allocation overhead but also means that all three * parts are resized together, and makes property access a bit complicated.
label: code-design
20921. x <- y + z
20922. get array index related to string (or return DUK_HSTRING_NO_ARRAY_INDEX); * avoids helper call if string has no array index value.
20923. [... re_obj input bc]
20924. Keep the error as the result (coercing it might fail below, * but we don't catch that now).
20925. * Note: although arguments object variable mappings are only established * for non-strict functions (and a call to a non-strict function created * the arguments object in question), an inner strict function may be doing * the actual property write. Hence the throw_flag applied here comes from * the property write call.
20926. push before advancing to keep reachable
20927. * duk_hthread allocation and freeing.
20928. [... pattern flags escaped_source bytecode]
20929. * Helper for updating callee 'caller' property.
20930. eat 'do'
20931. saves a few instructions to have this wrapper (see comment on duk_heap_mem_alloc)

20932. [source template result]
20933. 25 user flags
20934. also stash it before constructor, * in case we need it (as the fallback value)
20935. omit print
20936. Unary plus is used to force a fastint check, so must include * downgrade check.
20937. DUK_FLD_16BIT
20938. **comment:** should never happen, but be robust
 label: code-design
20939. _Formals: omitted if function is guaranteed not to need a (non-strict) arguments object
20940. **comment:** XXX: comes out as signed now
 label: code-design
20941. flags
20942. **comment:** If message is undefined, the own property 'message' is not set at * all to save property space. An empty message is inherited anyway.
 label: code-design
20943. writable but not deletable
20944. Decode UTF-8 codepoint from a sequence of hex escapes. The * first byte of the sequence has been decoded to 't'. ** Note that UTF-8 validation must be strict according to the * specification: E5.1 Section 15.1.3, decode algorithm step * 4.d.vii.8. URIError from non-shortest encodings is also * specifically noted in the spec.
20945. [... formals]
20946. **comment:** avoid tag 0xffff0, no risk of confusion with negative infinity
 label: code-design
20947. See: tests/ecmascript/test-spec-bound-constructor.js
20948. guaranteed to succeed
20949. * Define new property ** Note: this may fail if the holder is not extensible.
20950. used elements (includes DELETED)
20951. **comment:** XXX: since the enumerator may be a memory expensive object, * perhaps clear it explicitly here? If so, break jump must * go through this clearing operation.
 label: code-design
20952. -> [... key val replacer holder key]
20953. as is
20954. Step 1 is not necessary because duk_call_method() will take * care of it.
20955. * Code emission helpers ** Some emission helpers understand the range of target and source reg/const * values and automatically emit shuffling code if necessary. This is the * case when the slot in question (A, B, C) is used in the standard way and * for opcodes the emission helpers explicitly understand (like DUK_OP_CALL). ** The standard way is that: * - slot A is a target register * - slot B is a source register/constant * - slot C is a source register/constant * * If a slot is used in a non-standard way the caller must indicate this * somehow. If a slot is used as a target instead of a source (or vice * versa), this can be indicated with a flag to trigger proper shuffling * (e.g. DUK_EMIT_FLAG_B_IS_TARGET). If the value in the slot is not * register/const related at all, the caller must ensure that the raw value * fits into the corresponding slot so as to not trigger shuffling. The * caller must set a "no shuffle" flag to ensure compilation fails if * shuffling were to be triggered because of an internal error. ** For slots B and C the raw slot size is 9 bits but one bit is reserved for * the reg/const indicator. To use the full 9-bit range for a raw value, * shuffling must be disabled with the DUK_EMIT_FLAG_NO_SHUFFLE_{B,C} flag. * Shuffling is only done for A, B, and C slots, not the larger BC or ABC slots. ** There is call handling specific understanding in the A-B-C emitter to * convert call setup and call instructions into indirect ones if necessary.
20956. **comment:** XXX: make this an internal helper
 label: code-design
20957. assumed to bottom relative
20958. intentionally empty
20959. Compute time value from (double) parts. The parts can be either UTC * or local time; if local, they need to be (conceptually) converted into * UTC time. The parts may represent valid or invalid time, and may be * wildly out of range (but may cancel each other and still come out in * the valid Date range).
20960. aaaaabbbcccccc cccccccccc
20961. [[SetPrototypeOf]] standard behavior, E6 9.1.2
20962. **comment:** XXX: could unwind catchstack here, so that call handling * didn't need to do that?
 label: code-design
20963. 27: setUTCSeconds
20964. Lightfunc virtual properties are non-configurable, so * reject if match any of them.
20965. **comment:** init is unnecessary but suppresses "may be used uninitialized" warnings
 label: code-design
20966. no need to unwind catchstack
20967. 'Proxy' object
20968. 'errCreate'
20969. coerce lval with ToString()
20970. see algorithm
20971. * reverse()
20972. index is above internal buffer length -> property is fully normal
20973. ensure never re-entered until rescue cycle complete
20974. **comment:** order: most frequent to least frequent
 label: code-design
20975. **comment:** not very useful, used for debugging
 label: code-design
20976. catch stack depth
20977. [... res_obj]
20978. * Lexer defines.
20979. A token value. Can be memcpy()'d, but note that slot1/slot2 values are on the valstack. * Some fields (like num, str1, str2) are only valid for specific token types and may have * stale values otherwise.
20980. DUK_BUFOBJ_NODEJS_BUFFER
20981. XXX: if 'len' is low, may want to ensure array part is kept: * the caller is likely to want a dense array.
20982. XXX: version specific array format instead?
20983. * Process incoming debug requests ** Individual request handlers can push temporaries on the value stack and * rely on duk_debug_process_message() to restore the value stack top * automatically.
20984. roughly 2 kB
20985. Must avoid duk_pop() in exit path
20986. basic types from duk_features.h
20987. XXX: byte offset
20988. * instanceof
20989. ecmascript compiler limits
20990. token type
20991. -> [... key val replacer]
20992. call_flags
20993. * Comparisons (x >= y, x > y, x <= y, x < y) ** E5 Section 11.8.5: implement 'x < y' and then use negate and eval_left_first * flags to get the rest.
20994. Negative indices are always within allocated stack but * must not go below zero index.

20995. Executor interrupt default interval when nothing else requires a * smaller value. The default interval must be small enough to allow * for reasonable execution timeout checking but large enough to keep * impact on execution performance low.

20996. * Object internal value * * Returned value is guaranteed to be reachable / incref'd, caller does not need * to incref OR decref. No proxies or accessors are invoked, no prototype walk.

20997. Exponent

20998. general temp register

20999. 0xe0...0xef

21000. Maximum write size: XUTF8 path writes max DUK_UNICODE_MAX_XUTF8_LENGTH, * percent escape path writes max two times CESU-8 encoded BMP length.

21001. [arg1 ... argN this loggerLevel loggerName buffer]

21002. duplicate zeroing, expect for (possible) NULL inits

21003. rc_varname and reg_varbind are ignored here

21004. get as double to handle huge numbers correctly

21005. enum_index

21006. XXX: clarify on when and where DUK_CONST_MARKER is allowed

21007. * Error helpers

21008. * Data following the header depends on the DUK_HBUFFER_FLAG_DYNAMIC * flag. * * If the flag is clear (the buffer is a fixed size one), the buffer * data follows the header directly, consisting of 'size' bytes. * * If the flag is set, the actual buffer is allocated separately, and * a few control fields follow the header. Specifically: * * - a "void *" pointing to the current allocation * - a duk_size_t indicating the full allocated size (always >= 'size') * * If DUK_HBUFFER_FLAG_EXTERNAL is set, the buffer has been allocated * by user code, so that Duktape won't be able to resize it and won't * free it. This allows buffers to point to e.g. an externally * allocated structure such as a frame buffer. * * Unlike strings, no terminator byte (NUL) is guaranteed after the * data. This would be convenient, but would pad aligned user buffers * unnecessarily upwards in size. For instance, if user code requested * a 64-byte dynamic buffer, 65 bytes would actually be allocated which * would then potentially round upwards to perhaps 68 or 72 bytes.

21009. 'new'

21010. evaluate to plain value, no forced register (temp/bound reg both ok)

21011. Fast path is triggered for no exponent and also for balanced exponent * and fraction parts, e.g. for "1.23e2" == "123". Remember to respect * zero sign.

21012. DUK_USE_INTERRUPT_COUNTER

21013. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property put right now (putprop protects * against it internally).

21014. **comment:** * Parser control values for tokens. The token table is ordered by the * DUK_TOK_XXX defines. * * The binding powers are for lbp() use (i.e. for use in led() context). * Binding powers are positive for typing convenience, and bits at the * top should be reserved for flags. Binding power step must be higher * than 1 so that binding power "lbp - 1" can be used for right associative * operators. Currently a step of 2 is used (which frees one more bit for * flags).

label: code-design

21015. * Array part * * Note: ordering between array and entry part must match 'abandon array' * behavior in duk_hobject_props.c: key order after an array is abandoned * must be the same.

21016. **comment:** XXX: could be eliminated with valstack adjust

label: code-design

21017. [key val]

21018. DUK_TOK_CATCH

21019. Note: we rely on duk_safe_call() to fix up the stack for the caller, * so we don't need to pop stuff here. There is no return value; * caller determines rescued status based on object refcount.

21020. * Case conversion helper, with context/local sensitivity. * For proper case conversion, one needs to know the character * and the preceding and following characters, as well as * locale/language.

21021. [... arg1 ... argN envobj argobj]

21022. flags:nonstrict

21023. pass2 allocation handles this

21024. * Call the wrapped module function. * * Use a protected call so that we can update Duktape.modLoaded[resolved_id] * even if the module throws an error.

21025. advance manually

21026. **comment:** XXX: awkward

label: code-design

21027. target out-of-bounds (but positive)

21028. free some memory

21029. constrained by string length

21030. stable; precalculated for faster lookups

21031. * Thread defines

21032. same handling for identifiers and strings

21033. -> [... lval rval rval.prototype]

21034. 'implements'

21035. '='

21036. XXX: Not sure what the best return value would be in the API. * Return a boolean for now. Note that rc == 0 is success (true).

21037. Current convention is to use duk_size_t for value stack sizes and global indices, * and duk_idx_t for local frame indices.

21038. **comment:** XXX: where to release temp regs in intermediate expressions? * e.g. 1+2+3 -> don't inflate temp register count when parsing this. * that particular expression temp regs can be forced here.

label: code-design

21039. negative top-relative indices not allowed now

21040. instanceof

21041. avoid attempt to compact any objects

21042. expression parsing helpers

21043. return ToObject(this)

21044. one token

21045. flags field: LLLLLLFT, L = label (24 bits), F = flags (4 bits), T = type (4 bits)

21046. not protected, respect reclimit, is a constructor call

21047. DUK_TOK_BNOT

21048. E5 Section 8.6.1

21049. indicates a deleted string; any fixed non-NULL, non-hstring pointer works

21050. val is unsigned so >= 0

21051. **comment:** NOTE: we try to minimize code size by avoiding unnecessary pops, * so the stack looks a bit cluttered in this function. DUK_ASSERT_TOP() * assertions are used to ensure stack configuration is correct at each * step.

label: code-design

21052. must have start and end

21053. Check for breakpoints only on line transition. * Breakpoint is triggered when we enter the target * line from a different line, and the previous line * was within the same function. * * This condition is tricky: the condition used to be * that transition to -or across- the breakpoint line * triggered the breakpoint. This seems intuitively * better because it handles breakpoints on lines with * no emitted opcodes; but this leads to the issue * described in: <https://github.com/svaaraala/duktape/issues/263>.

21054. env and act may be NULL

21055. * Process yield * * After longjmp(), processing continues in bytecode executor longjmp * handler, which will e.g. update thr->resumer to NULL.

21056. -> [... enum key enum_target key]

21057. **comment:** XXX: store 'bcode' pointer to activation for faster lookup?

label: code-design

21058. * Unicode tables
21059. [... obj]
21060. Note: space must cater for both JX and JC.
21061. Needs to be inserted; scan backwards, since we optimize * for the case where elements are nearly in order.
21062. Normal non-bound function.
21063. Eating a left curly; regexp mode is allowed by left curly * based on duk_token_lbp[] automatically.
21064. A validated read() is always a number, so it's write coercion * is always side effect free and won't invalidate pointers etc.
21065. current and previous token for parsing
21066. probe sequence
21067. message is empty -> return name
21068. NaN timevalue: we need to coerce the arguments, but * the resulting internal timestamp needs to remain NaN. * This works but is not pretty: parts and dparts will * be partially uninitialized, but we only write to them.
21069. **comment:** It might seem that replacing 'thr->heap' with just 'heap' below * might be a good idea, but it increases code size slightly * (probably due to unnecessary spilling) at least on x64.
 label: code-design
21070. Currently nothing to free; 'data' is a heap object
21071. 'prototype'
21072. 'jx'
21073. default prototype (Note: 'obj' must be reachable)
21074. **comment:** XXX: currently NULL allocations are not supported; remove if later allowed
 label: code-design
21075. Unwind the topmost callstack entry before reusing it
21076. internal define property: skip write silently if exists
21077. since new_alloc_size > 0
21078. borrowed, no refcount
21079. allow e.g. '0x0009' and '00077'
21080. DUK_BUFOBJ_DATAVIEW
21081. * Compact an object. Minimizes allocation size for objects which are * not likely to be extended. This is useful for internal and non- * extensible objects, but can also be called for non-extensible objects. * May abandon the array part if it is computed to be too sparse. * * This call is relatively expensive, as it needs to scan both the * entries and the array part. * * The call may fail due to allocation error.
21082. find elements to swap
21083. currently, always the case
21084. * Detected label
21085. **comment:** XXX: an array can have length higher than 32 bits; this is not handled * correctly now.
 label: code-design
21086. * Helper macros
21087. **comment:** XXX: Simplify this algorithm, should be possible to come up with * a shorter and faster algorithm by inspecting IEEE representation * directly.
 label: code-design
21088. * CheckObjectCoercible() (E5 Section 9.10) * * Note: no API equivalent now.
21089. constants are strings or numbers now
21090. '\n'
21091. Must behave like a no-op with NULL and any pointer returned from * malloc/realloc with zero size.
21092. if (is_regexp)
21093. [thisArg arg1 ... argN func boundFunc]
21094. -Infinity
21095. Undefine local defines
21096. * Top-level include file to be used for all (internal) source files. * * Source files should not include individual header files, as they * have not been designed to be individually included.
21097. * Init built-in strings
21098. * Two's complement arithmetic.
21099. [... constructor arg1 ... argN]
21100. no need to decref previous value
21101. 'finally'
21102. Zero size compare not an issue with DUK_MEMCMP.
21103. [...] closure template val]
21104. YIELDED: Ecmascript activation + Duktape.Thread.yield() activation
21105. may be NULL (but only if size is 0)
21106. write to entry part
21107. * Ecmascript bytecode executor. * * Resume execution for the current thread from its current activation. * Returns when execution would return from the entry level activation, * leaving a single return value on top of the stack. Function calls * and thread resumptions are handled internally. If an error occurs, * a longjmp() with type DUK_LJ_TYPE_THROW is called on the entry level * setjmp() jmpbuf. * * Ecmascript function calls and coroutine resumptions are handled * internally (by the outer executor function) without recursive C calls. * Other function calls are handled using duk_handle_call(), increasing * C recursion depth. * * Abrupt completions (= long control transfers) are handled either * directly by reconfiguring relevant stacks and restarting execution, * or via a longjmp. Longjmp-free handling is preferable for performance * (especially Emscripten performance), and is used for: break, continue, * and return. * * For more detailed notes, see doc/execution.rst. * * Also see doc/code-issues.rst for discussion of setjmp(), longjmp(), * and volatile.
21108. * Debug connection message write helpers
21109. We know _Formals is dense and all entries will be in the * array part. GC and finalizers shouldn't affect _Formals * so side effects should be fine.
21110. don't want an intermediate exposed state with func == NULL
21111. **comment:** * (Re)try handling the longjmp. * * A longjmp handler may convert the longjmp to a different type and * "virtually" rethrow by goto'ing to 'check_longjmp'. Before the goto, * the following must be updated: * - the heap 'lj' state * - 'thr' must reflect the "throwing" thread
 label: code-design
21112. XXX=
21113. separators: space, space, colon
21114. **comment:** Format of magic, bits: * ...:1 field type; 0=uint8, 1=uint16, 2=uint32, 3=float, 4=double, 5=unused, 6=unused, 7=unused * 3: endianness: 0=little, 1=big * 4: signed: 1=yes, 0=no * 5: typedarray: 1=yes, 0=no
 label: code-design
21115. **comment:** Note: reuse 'curr' as a temp propdesc
 label: code-design
21116. Slow path
21117. Lightweight function: never bound, so terminate.
21118. unconditionally; is_global==0
21119. Node.js Buffer: return offset + #bytes written (i.e. next * write offset).
21120. string limits
21121. **comment:** function makes one or more slow path accesses
 label: code-design
21122. DUK_USE_AUGMENT_ERROR_CREATE
21123. No need for a NULL/zero-size check because new_size > 0
21124. Assume value stack sizes (in elements) fits into duk_idx_t.

21125. **comment:** Relookup in case duk_js_tointeger() ends up e.g. coercing an object.
label: code-design

21126. '>'
21127. * Debug connection skip primitives
21128. valstack index of 'func' and retval (relative to entry valstack_bottom)
21129. XXX: array internal?
21130. Assertion compatible inside a comma expression, evaluates to void. * Currently not compatible with DUK_USE_PANIC_HANDLER() which may have * a statement block.
21131. avoid key quotes when key is an ASCII Identifier
21132. embed: nothing
21133. **comment:** bit mask which indicates that a regconst is a constant instead of a register
label: code-design

21134. * Memory allocation handling.
21135. current assertion is quite strong: decref's and set to unused
21136. Coerce an duk_ivalue to a 'plain' value by generating the necessary * arithmetic operations, property access, or variable access bytecode. * The duk_ivalue argument ('x') is converted into a plain value as a * side effect.
21137. element type
21138. parsing in "scanning" phase (first pass)
21139. regexp res_obj is at offset 4
21140. Flags for duk_js_compare_helper().
21141. **comment:** * Eval * * Eval needs to handle both a "direct eval" and an "indirect eval". * Direct eval handling needs access to the caller's activation so that its * lexical environment can be accessed. A direct eval is only possible from * Ecmascript code; an indirect eval call is possible also from C code. * When an indirect eval call is made from C code, there may not be a * calling activation at all which needs careful handling.
label: code-design

21142. no fractions
21143. 'Function'
21144. 29: setUTCMinutes
21145. * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written.
21146. accept comma, expect new value
21147. -> [... reviver holder name val]
21148. [... pattern flags escaped_source buffer]
21149. reg
21150. empty match -> bump and continue
21151. regexp execution limits
21152. * Single source autogenerated distributable for Duktape 1.5.2. * * Git commit cad34ae155acb0846545ca6bf2d29f9463b22bbb (v1.5.2). * Git branch HEAD. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.
21153. SCANBUILD: "Dereference of null pointer", normal
21154. catch or with binding is currently active
21155. Note: unbalanced stack on purpose
21156. Get a borrowed duk_tval pointer to the current 'this' binding. Caller must * make sure there's an active callstack entry. Note that the returned pointer * is unstable with regards to side effects.
21157. expt values [0x001,0x7fe] = normal
21158. Receiver: Proxy object
21159. **comment:** * Dump/load helpers, xxx_raw() helpers do no buffer checks
label: code-design

21160. **comment:** * Push readable string summarizing duk_tval. The operation is side effect * free and will only throw from internal errors (e.g. out of memory). * This is used by e.g. property access code to summarize a key/base safely, * and is not intended to be fast (but small and safe).
label: code-design

21161. * Type error thrower, E5 Section 13.2.3.
21162. **comment:** Corner case: see test-numconv-parse-mant-carry.js. We could * just bump the exponent and update bitstart, but it's more robust * to recompute (but avoid rounding twice).
label: code-design

21163. The JA(value) operation: encode array. * * Stack policy: [array] -> [array].
21164. XXX: init or assert catch depth etc -- all values
21165. happens when hash part dropped
21166. 1x heap size
21167. no need for decref/incref because value is a number
21168. **comment:** XXX: this may now fail, and is not handled correctly
label: code-design

21169. [key undefined] -> [key]
21170. * join(), toLocaleString() * * Note: checking valstack is necessary, but only in the per-element loop. * * Note: the trivial approach of pushing all the elements on the value stack * and then calling duk_join() fails when the array contains a large number * of elements. This problem can't be offloaded to duk_join() because the * elements to join must be handled here and have special handling. Current * approach is to do intermediate joins with very large number of elements. * There is no fancy handling; the prefix gets re-joined multiple times.

21171. stack top contains 'false'
21172. LAYOUT 2
21173. If 16-bit hash is in use, stuff it into duk_heaphdr_string flags.
21174. * Other cases, use C recursion. * * If a tail call was requested we ignore it and execute a normal call. * Since Duktape 0.11.0 the compiler emits a RETURN opcode even after * a tail call to avoid test-bug-tailcall-thread-yield-resume.js. * * Direct eval call: (1) call target (before following bound function * chain) is the built-in eval() function, and (2) call was made with * the identifier 'eval'.
21175. (

21176. NOTE! Caller must ensure that any side effects from the * coercions below are safe. If that cannot be guaranteed * (which is normally the case), caller must coerce the * argument using duk_to_number() before any pointer * validations; the result of duk_to_number() always coerces * without side effects here.

21177. **comment:** Compared to duk_handle_call(): * - protected call: never * - ignore recursion limit: never
label: code-design

21178. Note: we ask for one return value from duk_safe_call to get this * error debugging here.
21179. -> [sep ToObject(this) len str]
21180. covered by comparison
21181. Setting "no shuffle A" is covered by the assert, but it's needed * with DUK_USE_SHUFFLE_TORTURE.
21182. **comment:** XXX: .code = err_code disabled, not sure if useful
label: code-design

21183. For now, restrict result array into 32-bit length range.
21184. coerce in-place
21185. DUK_ERR_ASSERTION_ERROR: no macros needed
21186. directive prologue status at entry
21187. * ToNumber() (E5 Section 9.3) * * Value to convert must be on stack top, and is popped before exit. * * See: <http://www.cs.indiana.edu/~burger/FP-Printing-PLDI96.pdf> * <http://www.cs.indiana.edu/~burger/fp/index.html> * * Notes on the conversion: * * - There are specific requirements on the accuracy of the conversion * through a "Mathematical Value" (MV), so this conversion is not * trivial. * * - Quick rejects (e.g. based on first char) are difficult because * the

grammar allows leading and trailing white space. ** - Quick reject based on string length is difficult even after * accounting for white space; there may be arbitrarily many * decimal digits. ** - Standard grammar allows decimal values ("123"), hex values * ("0x123") and infinities * * - Unlike source code literals, ToNumber() coerces empty strings * and strings with only whitespace to zero (not NaN).

21188. [... num]

21189. arrays MUST have a 'length' property

21190. Using an explicit 'ASCII' flag has larger footprint (one call site * only) but is quite useful for the case when there's no explicit * 'clen' in duk_hstring.

21191. **comment:** XXX: try to optimize to 8 (would now be possible, <200 used)

label: code-design

21192. * Arguments handling helpers (argument map mainly). ** An arguments object has exotic behavior for some numeric indices. * Accesses may translate to identifier operations which may have * arbitrary side effects (potentially invalidating any duk_tval * pointers).

21193. Undefined/null are considered equal (e.g. "null == undefined" -> true).

21194. * String value of 'blen+1' bytes follows (+1 for NUL termination * convenience for C API). No alignment needs to be guaranteed * for strings, but fields above should guarantee alignment-by-4 * (but not alignment-by-8).

21195. **comment:** XXX: Could be improved by coercing to a readable duk_tval (especially string escaping)

label: code-design

21196. **comment:** Inner executor, performance critical.

label: code-design

21197. significant from precision perspective

21198. [... Logger clog logfunc clog]

21199. No assertions for offset or length; in particular, \ * it's OK for length to be longer than underlying \ * buffer. Just ensure they don't wrap when added. \

21200. fits into duk_small_int_t

21201. Caller doesn't need to check exotic proxy behavior (but does so for * some fast paths).

21202. is eval code

21203. * Iterate the bitstream (line diffs) until PC is reached

21204. varmap is already in comp_ctx->curr_func.varmap_idx

21205. * Convert binary digits into an IEEE double. Need to handle * denormals and rounding correctly.

21206. [key setter this val] -> [key retval]

21207. DUK_TOK_LAND

21208. **comment:** * E5 Section 15.3.4.2 places few requirements on the output of * this function: ** - The result is an implementation dependent representation * of the function; in particular ** - The result must follow the syntax of a FunctionDeclaration. * In particular, the function must have a name (even in the * case of an anonymous function or a function with an empty * name). ** - Note in particular that the output does NOT need to compile * into anything useful.

label: code-design

21209. Note that multiple catchstack entries may refer to the same * callstack entry.

21210. (-Number.MAX_VALUE).toString(2).length == 1025, + spare

21211. DUK_HNATIVEFUNCTION_H_INCLUDED

21212. check pointer at end

21213. **comment:** XXX: any way to avoid decoding magic bit; there are quite * many function properties and relatively few with magic values.

label: code-design

21214. chain jumps for 'fall-through' * after a case matches.

21215. duk_unicode_ids_noa[]

21216. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).

label: code-design

21217. reset longjmp

21218. non-strict equality for buffers compares contents

21219. all codepoints up to U+FFFF

21220. recursion tracking happens here only

21221. roughly 128 bytes

21222. default clause matches next statement list (if any)

21223. * E5 Sections 11.8.7, 8.12.6. ** Basically just a property existence check using [[HasProperty]].

21224. new value

21225. MULTIPLICATIVE EXPRESSION

21226. **comment:** * typeof ** E5 Section 11.4.3. ** Very straightforward. The only question is what to return for our * non-standard tag / object types. ** There is an unfortunate string constant define naming problem with * typeof return values for e.g. "Object" and "object"; careful with * the built-in string defines. The LC_XXX defines are used for the * lowercase variants now.

label: code-design

21227. -> [... func this]

21228. **comment:** XXX: should this happen in the callee's activation or after unwinding?

label: code-design

21229. **comment:** * Four possible outcomes: * 1. A 'finally' in the same function catches the 'return'. * It may continue to propagate when 'finally' is finished, * or it may be neutralized by 'finally' (both handled by * ENDFIN). ** 2. The return happens at the entry level of the bytecode * executor, so return from the executor (in C stack). ** 3. There is a calling (Ecmascript) activation in the call * stack => return to it, in the same executor instance. ** 4. There is no calling activation, and the thread is * terminated. There is always a resumer in this case, * which gets the return value similarly to a 'yield' * (except that the current thread can no longer be * resumed).

label: code-design

21230. caller checks

21231. t has high mantissa

21232. * Flags ** Fixed buffer: 0 * Dynamic buffer: DUK_HBUFFER_FLAG_DYNAMIC * External buffer: DUK_HBUFFER_FLAG_DYNAMIC | DUK_HBUFFER_FLAG_EXTERNAL

21233. DUK_TOK_LCURLY

21234. **comment:** * Identifier access and function closure handling. ** Provides the primitives for slow path identifier accesses: GETGLOBAL, * GETGLOBAL, GETGLOBAL, etc. The fast path, direct register accesses, should * be used for most identifier accesses. Consequently, these slow path * primitives should be optimized for maximum compactness. ** Ecmascript environment records (declarative and object) are represented * as internal objects with control keys. Environment records have a * parent record ("outer environment reference") which is represented by * the implicit prototype for technical reasons (in other words, it is a * convenient field). The prototype chain is not followed in the ordinary * sense for variable lookups. ** See identifier-handling.rst for more details on the identifier algorithms * and the internal representation. See function-objects.rst for details on * what function templates and instances are expected to look like. ** Care must be taken to avoid duk_tval pointer invalidation caused by * e.g. value stack or object resizing. ** TODO: properties for function instances could be initialized much more * efficiently by creating a property allocation for a certain size and * filling in keys and values directly (and INCREFing both with "bulk incref" * primitives. ** XXX: duk_hobject_getprop() and duk_hobject_putprop() calls are a bit * awkward (especially because they follow the prototype chain); rework * if "raw" own property helpers are added.

label: code-design

21235. If 'day' is NaN, returns NaN.

21236. * New length not lower than old length => no changes needed * (not even array allocation).

21237. * API oriented helpers

21238. [enum_target res key true]

21239. **comment:** XXX: we could save space by using _Target OR _This. If _Target, assume * this binding is undefined. If _This, assumes this binding is _This, and * target is also _This. One property would then be enough.

label: code-design

21240. ADDITIVE EXPRESSION

21241. macros

21242. default case does not exist, or no statements present * after default case: finish case evaluation

21243. use temp_next for tracking register allocations

21244. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer

21245. DUK_REGEXP_H_INCLUDED

21246. **comment:** XXX: add fastint support?

label: requirement

21247. FOUND

21248. * Additional control information is placed into the object itself * as internal properties to avoid unnecessary fields for the * majority of functions. The compiler tries to omit internal * control fields when possible. * * Function templates: * * { * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * } * * Function instances: * * { * length: 2, * prototype: { constructor: <func> }, * caller: <thrower>, * arguments: <thrower>, * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * _Varenv: <variable environment of closure>, * _Lexenv: <lexical environment of closure (if differs from _Varenv)> * } * * More detailed description of these properties can be found * in the documentation.

21249. initialize for debug prints, needed if sce==NULL

21250. start of valstack allocation

21251. Allocate a new duk_hbuffer of a certain type and return a pointer to it * (NULL on error). Write buffer data pointer to 'out_bufdata' (only if * allocation successful).

21252. Lookup 'key' from arguments internal 'map', delete mapping if found. * Used in E5 Section 10.6 algorithm for [[Delete]]. Note that the * variable/argument itself (where the map points) is not deleted.

21253. -> [... enum_target res trap handler target]

21254. derived types

21255. Awkward inclusion condition: drop out of compilation if not needed by any * call site: object hash part or probing stringtable.

21256. next instruction to execute (points to 'func' bytecode, stable pointer), NULL for native calls

21257. [offset littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)

21258. Must be a "pointer object", i.e. class "Pointer"

21259. ES6 proxy

21260. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array * stack[4] = regexp res_obj (if is_regexp)

21261. value of dbg_exec_counter when we last did a Date-based check

21262. idx_retval unsigned

21263. no negative sign for zero

21264. **comment:** Two value cycle, see e.g. test-bi-date-tzoffset-basic-fi.js. * In these cases, favor a higher tzoffset to get a consistent * result which is independent of iteration count. Not sure if * this is a generically correct solution.

label: code-design

21265. 'thr' is the current thread, as no-one resumes except us and we * switch 'thr' in that case.

21266. lastIndex must be ignored for non-global regexps, but get the * value for (theoretical) side effects. No side effects can * really occur, because lastIndex is a normal property and is * always non-configurable for RegExp instances.

21267. clamp anything above nargs

21268. **comment:** XXX: this could be more compact by accessing the internal properties * directly as own properties (they cannot be inherited, and are not * externally visible).

label: code-design

21269. Get/set the current user visible size, without accounting for a dynamic * buffer's "spare" (= usable size).

21270. looping should never happen

21271. Validate byte read/write for virtual 'offset', i.e. check that the * offset, taking into account h->offset, is within the underlying * buffer size. This is a safety check which is needed to ensure * that even a misconfigured duk_hbufferobject never causes memory * unsafe behavior (e.g. if an underlying dynamic buffer changes * after being setup). Caller must ensure 'buf' != NULL.

21272. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.

21273. digit position is absolute, not relative

21274. c

21275. **comment:** Same test with volatiles

label: test

21276. because callstack_top > 0

21277. There's overlap: the desired end result is that * conceptually a copy is made to avoid "trampling" * of source data by destination writes. We make * an actual temporary copy to handle this case.

21278. **comment:** XXX: just use toString() for now; permitted although not recommended. * nargs==1, so radix is passed to toString().

label: code-design

21279. %d; only 16 bits are guaranteed

21280. 'this' value: * E5 Section 6.b.i * * The only (standard) case where the 'this' binding is non-null is when * (1) the variable is found in an object environment record, and * (2) that object environment record is a 'with' block. *

21281. This is important to ensure dynamic buffer data pointer is not * NULL (which is possible if buffer size is zero), which in turn * causes portability issues with e.g. memmove() and memcpy().

21282. Since we already started writing the reply, just emit nothing.

21283. * For top-level objects, 'length' property has the following * default attributes: non-writable, non-enumerable, non-configurable * (E5 Section 15). * * However, 'length' property for Array.prototype has attributes * expected of an Array instance which are different: writable, * non-enumerable, non-configurable (E5 Section 15.4.5.2). * * This is currently determined implicitly based on class; there are * no attribute flags in the init data.

21284. -> [... errhandler undefined(= this) errval]

21285. Reconfigure value stack for return to an Ecmascript function at 'act_idx'.

21286. * First pass. * * Gather variable/function declarations needed for second pass. * Code generated is dummy and discarded.

21287. valid pc range is [0, length[

21288. Current tracedata contains 2 entries per callstack entry.

21289. [... errval]

21290. **comment:** XXX: there is a small risk here: because the ISO 8601 parser is * very loose, it may end up parsing some datetime values which * would be better parsed with a platform specific parser.

label: code-design

21291. * Return (t - LocalTime(t)) in minutes: * * t - LocalTime(t) = t - (t + LocalTZA + DaylightSavingTA(t)) * = -(LocalTZA + DaylightSavingTA(t)) * * where DaylightSavingTA() is checked for time 't'. * * Note that the sign of the result is opposite to common usage, * e.g. for EE(S)T which normally is +2h or +3h from UTC, this * function returns -120 or -180. *

21292. **comment:** XXX: no need for indirect call

label: code-design

21293. XXX: the implementation now assumes "chained" bound functions, * whereas "collapsed" bound functions (where there is ever only * one bound function which directly points to a non-bound, final * function) would require a "collapsing" implementation which * merges argument lists etc here.

21294. The current implementation of localeCompare() is simply a codepoint * by codepoint comparison, implemented with a simple string compare * because UTF-8 should preserve codepoint ordering (assuming valid * shortest UTF-8 encoding). * * The specification requires that the return value must be related * to the sort order: e.g. negative means that 'this' comes before * 'that' in sort order. We assume an ascending sort order.

21295. valstack

21296. not protected, respect reclimit, not constructor
21297. Push the resulting view object and attach the ArrayBuffer.
21298. **comment:** XXX: this should be optimized to be a raw query and avoid valstack * operations if possible.
 label: code-design
21299. unbalanced stack
21300. value from hook
21301. * If object has been marked finalizable, move it to the * "to be finalized" work list. It will be collected on * the next mark-and-sweep if it is still unreachable * after running the finalizer.
21302. shared pop
21303. * Struct defs
21304. nbytes zero size case * <-----> * [... | p | x | x | q] [... | p==q] * => [... | x | x | q] [...]
21305. **comment:** not really necessary
 label: code-design
21306. [... target] -> [... target keys]
21307. Leading zero.
21308. ''
21309. Make underlying buffer compact to match DUK_BW_GET_SIZE().
21310. first heap allocated, match bit boundary
21311. **comment:** XXX: more specific error classes?
 label: code-design
21312. 25: setUTCMilliseconds
21313. * Intermediate value helpers
21314. Limits
21315. Encoded as surrogate pair, each encoding to 3 bytes for * 6 bytes total. Codepoints above U+10FFFF encode as 6 bytes * too, see duk_unicode_encode_cesu8().
21316. * Setup value stack: clamp to 'nargs', fill up to 'nregs' * * Value stack may either grow or shrink, depending on the * number of func registers and the number of actual arguments. * If nregs >= 0, func wants args clamped to 'nargs'; else it * wants all args (= 'num_stack_args').
21317. [sep ToObject(this) len sep result]
21318. * Variant 2 or 4
21319. Lightfunc, always success.
21320. step 11.c is relevant only if non-strict (checked in 11.c.ii)
21321. XX==
21322. **comment:** A plain buffer coerces to a Duktape.Buffer because it's the * object counterpart of the plain buffer value. But it might * still make more sense to produce an ArrayBuffer here?
 label: code-design
21323. Force exponential format. Used for toExponential().
21324. Cannot wrap: each object is at least 8 bytes so count is * at most 1/8 of that.
21325. wipe the capture range made by the atom (if any)
21326. Anything else is not constructable.
21327. attrs in E5 Section 15.3.5.1
21328. * Use the index in the header to find the right starting point
21329. Tolerate act_caller->func == NULL which happens in * some finalization cases; treat like unknown caller.
21330. **comment:** XXX: remove DUK_CALL_FLAG_IGNORE_RECLIMIT flag: there's now the * reclimit bump?
 label: code-design
21331. maximum length of standard format tag that we support
21332. buffer size is >= 1
21333. ignore encoding for now
21334. enable manually for dumping
21335. (?=
21336. require to be safe
21337. Parser part count.
21338. for side effects, result ignored
21339. Now we can check offset validity.
21340. Milliseconds between status notify and transport peeks.
21341. Matching separator index is used in the control table
21342. rehash even if old and new sizes are the same to get rid of * DELETED entries.
21343. * Two pass approach to processing the property descriptors. * On first pass validate and normalize all descriptors before * any changes are made to the target object. On second pass * make the actual modifications to the target object. * * Right now we'll just use the same normalize/validate helper * on both passes, ignoring its outputs on the first pass.
21344. * Stringify implementation.
21345. ToNumber() for a double is a no-op.
21346. default: char escape (two chars)
21347. * Hobject allocation. * * Provides primitive allocation functions for all object types (plain object, * compiled function, native function, thread). The object return is not yet * in "heap allocated" list and has a refcount of zero, so caller must careful.
21348. Check actual underlying buffers for validity and that they * cover the copy. No side effects are allowed after the check * so that the validity status doesn't change.
21349. * Special parser for character classes; calls callback for every * range parsed and returns the number of ranges present.
21350. negate result
21351. * Lightfunc
21352. **comment:** * Refzero handling is skipped entirely if (1) mark-and-sweep is * running or (2) execution is paused in the debugger. The objects * are left in the heap, and will be freed by mark-and-sweep or * eventual heap destruction. * * This is necessary during mark-and-sweep because refcounts are also * updated during the sweep phase (otherwise objects referenced by a * swept object would have incorrect refcounts) which then calls here. * This could be avoided by using separate decref macros in * mark-and-sweep; however, mark-and-sweep also calls finalizers which * would use the ordinary decref macros anyway and still call this * function. * * This check must be enabled also when mark-and-sweep support has been * disabled: the flag is also used in heap destruction when running * finalizers for remaining objects, and the flag prevents objects from * being moved around in heap linked lists.
 label: code-design
21353. XXX: len >= 0x80000000 won't work below because a signed type * is needed by qsort.
21354. Special handling for year sign.
21355. [arg1 ... argN obj length new_length]
21356. The algorithm in E5.1 Section 15.9.1.12 normalizes month, but * does not normalize the day-of-month (nor check whether or not * it is finite) because it's not necessary for finding the day * number which matches the (year,month) pair. * * We assume that duk_day_from_year() is exact here. * * Without an explicit infinity / NaN check in the beginning, * day_num would be a bogus integer here. * * It's possible for 'year' to be out of integer range here. * If so, we need to return NaN without integer overflow. * This fixes test-bug-setyear-overflow.js.
21357. * Node.js Buffer.prototype.slice([start], [end]) * ArrayBuffer.prototype.slice(begin, [end]) * TypedArray.prototype.slice(begin, [end]) * * The API calls are almost identical; negative indices are counted from end * of buffer, and final indices are clamped (allowing crossed indices). Main * differences: * * - Copy/view behavior; Node.js .slice() and TypedArray .subarray() create * views, ArrayBuffer .slice() creates a copy * * - Resulting object has a different class and prototype depending on the * call (or 'this' argument) * * - TypedArray .subarray() arguments are element indices, not byte offsets
21358. **comment:** When alloc_size == 0 the second allocation may not * actually exist.
 label: code-design
21359. statements go here (if any) on next loop

21360. 'yield'
21361. eat trailing comma
21362. pruned
21363. * Declarative environment record. * * Identifiers can never be stored in ancestors and are * always plain values, so we can use an internal helper * and access the value directly with an duk_tval ptr. * * A closed environment is only indicated by it missing * the "book-keeping" properties required for accessing * register-bound variables.
21364. Note: there is no requirement that: 'thr->callstack_preventcount == 1' * like for yield.
21365. lookup results is ignored
21366. **comment:** * ToPrimitive() (E5 Section 9.1) * * ==> implemented in the API.
 label: requirement
21367. resume caller must be an ecmascript func
21368. e.g. a = -4, b = 5 --> -4 - 5 + 1 / 5 --> -8 / 5 --> -1 * a = -5, b = 5 --> -5 - 5 + 1 / 5 --> -9 / 5 --> -1 * a = -6, b = 5 --> -6 - 5 + 1 / 5 --> -10 / 5 --> -2
21369. errors out if out of memory
21370. stack contains value (if requested), 'out_desc' is set
21371. no issues with empty memcmp()
21372. [str] -> [substr]
21373. func is NULL for lightfunc
21374. property attributes for identifier (relevant if value != NULL)
21375. Default function to format objects. Tries to use toLogString() but falls * back to toString(). Any errors are propagated out without catching.
21376. don't allow continue
21377. respect reclimit, not constructor
21378. * Notation: double underscore used for internal properties which are not * stored in the property allocation (e.g. '__valstack').
21379. 0x60-0x6f
21380. DUK_BUFOBJ_INT32ARRAY
21381. Caller has already eaten the first char so backtrack one byte.
21382. controls for minimum entry part growth
21383. temp accumulation buffer
21384. Unwind.
21385. * Object built-ins
21386. callstack index
21387. **comment:** XXX: The ES5/5.1/6 specifications require that the key in 'key in obj' * must be string coerced before the internal HasProperty() algorithm is * invoked.
 A fast path skipping coercion could be safely implemented for * numbers (as number-to-string coercion has no side effects). For ES6 * proxy behavior, the trap 'key' argument must be in a string coerced * form (which is a shame).
 label: code-design
21388. take last entry
21389. curr is data, desc is accessor
21390. **comment:** XXX: The incref macro takes a thread pointer but doesn't * use it right now.
 label: code-design
21391. An object may be in heap_allocated list with a zero * refcount also if it is a temporary object created by * a finalizer; because finalization now runs inside * mark-and-sweep, such objects will not be queued to * refzero_list and will thus appear here with refcount * zero.
21392. Accept 32-bit decimal integers, no leading zeroes, signs, etc. * Leading zeroes are not accepted (zero index "0" is an exception * handled above).
21393. **comment:** Set: currently a finalizer is disabled by setting it to * undefined; this does not remove the property at the moment. * The value could be type checked to be either a function * or something else; if something else, the property could * be deleted.
 label: code-design
21394. * Bitstream decoder
21395. statement id allocation (running counter)
21396. 'public'
21397. single match always
21398. Note: 'curr' refers to 'length' propdesc
21399. Use double parts, they tolerate unnormalized time. * * Note: DUK_DATE_IDX_WEEKDAY is initialized with a bogus value (DUK__PI_TZHOUR) * on purpose. It won't be actually used by duk_bi_date_get_timeval_from_dparts(), * but will make the value initialized just in case, and avoid any * potential for Valgrind issues.
21400. Steps 8-10 have been merged to avoid a "partial" variable.
21401. Used during heap destruction: don't actually run finalizers * because we're heading into forced finalization. Instead, * queue finalizable objects back to the heap_allocated list.
21402. duk_safe_call discipline
21403. **comment:** * Unicode tables containing ranges of Unicode characters in a * packed format. These tables are used to match non-ASCII * characters of complex productions by resorting to a linear * range-by-range comparison. This is very slow, but is expected * to be very rare in practical Ecmascript source code, and thus * compactness is most important. * * The tables are matched using uni_range_match() and the format * is described in src/extract_chars.py.
 label: code-design
21404. **comment:** XXX: encoding is ignored now.
 label: code-design
21405. use 'undefined' for value automatically
21406. [... obj ...]
21407. **comment:** Function pointers do not always cast correctly to void * * (depends on memory and segmentation model for instance), * so they coerce to NULL.
 label: code-design
21408. Buffer length is bounded to 0xffff automatically, avoid compile warning.
21409. Integer division which floors also negative values correctly.
21410. consequence of above
21411. * Self tests to ensure execution environment is sane. Intended to catch * compiler/platform problems which cannot be detected at compile time.
21412. DUK_TAG_NUMBER would logically go here, but it has multiple 'tags'
21413. backtrack (safe)
21414. DUK_FLD_32BIT
21415. * Parse a RegExp token. The grammar is described in E5 Section 15.10. * Terminal constructions (such as quantifiers) are parsed directly here. * * 0xffffffffU is used as a marker for "infinity" in quantifiers. Further, * DUK__MAX_RE_QUANT_DIGITS limits the maximum number of digits that * will be accepted for a quantifier.
21416. **comment:** Note: reuse 'curr'
 label: code-design
21417. * Convert and push final string.
21418. evaluate to final form (e.g. coerce GETPROP to code), throw away temp
21419. * Selftest code
21420. use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to the existing one)
21421. duk_tval ptr for 'func' on stack (borrowed reference) or tv_func_copy
21422. * Encoding and decoding basic formats: hex, base64. * * These are in-place operations which may allow an optimized implementation. * * Base-64: <https://tools.ietf.org/html/fc4648#section-4>
21423. Run mark-and-sweep a few times just in case (unreachable object * finalizers run already here). The last round must rescue objects * from the previous round without running any more finalizers. This * ensures rescued objects get their FINALIZED flag cleared so that * their finalizer is called once more in forced finalization to * satisfy finalizer guarantees. However, we don't want to run any * more finalizer because that'd required one more loop, and so on.

21424. required to keep recursion depth correct
21425. [... buf loop (proplist) (gap) holder ""]
21426. Get default hash part size for a certain entry part size.
21427. Maximum number of digits generated.
21428. [... func buf] -> [... buf]
21429. **comment:** XXX: could share code with duk_js_ops.c, duk_js_compare_helper
 label: code-design
21430. Math.pow(+0,y) should be Infinity when y<0. NetBSD pow() * returns -Infinity instead when y is <0 and finite. The * if-clause also catches y == -Infinity (which works even * without the fix).
21431. Free strings in the stringtable and any allocations needed * by the stringtable itself.
21432. e.g. DUK_OP_PREINCV
21433. If neutered must return 0; offset is zeroed during * neutering.
21434. array of declarations: [name1, val1, name2, val2, ...] * valN = (typeN) | (fnum << 8), where fnum is inner func number (0 for vars) * record function and variable declarations in pass 1
21435. breakpoints: [0,breakpoint_count[gc reachable
21436. **comment:** XXX: inefficient loop
 label: code-design
21437. * Insert a jump offset at 'offset' to complete an instruction * (the jump offset is always the last component of an instruction). * The 'skip' argument must be computed relative to 'offset', * -without- taking into account the skip field being inserted. * * ... A B C ins X Y Z ... (ins may be a JUMP, SPLIT1/SPLIT2, etc) * => ... A B C ins SKIP X Y Z * * Computing the final (adjusted) skip value, which is relative to the * first byte of the next instruction, is a bit tricky because of the * variable length UTF-8 encoding. See doc/regexp.rst for discussion.
21438. * NYF <int: 5> <int: fatal> <str: msg> <str: filename> <int: linenumber> EOM
21439. Note: not necessary to check p against re_ctx->input_end: * the memory access is checked by duk__inp_get_cp(), while * valid compiled regexps cannot write a saved[] entry * which points to outside the string.
21440. Strings and ROM objects are never placed on the heap allocated list.
21441. [... val root ""] -> [... val val']
21442. **comment:** * JSON built-ins. * * See doc/json.rst. * * Codepoints are handled as duk_uint_fast32_t to ensure that the full * unsigned 32-bit range is supported. This matters to e.g. JX. * * Input parsing doesn't do an explicit end-of-input check at all. This is * safe: input string data is always NUL-terminated (0x00) and valid JSON * inputs never contain plain NUL characters, so that as long as syntax checks * are correct, we'll never read past the NUL. This approach reduces code size * and improves parsing performance, but it's critical that syntax checks are * indeed correct!
 label: code-design
21443. stack type fits into 16 bits
21444. 'set'
21445. low to high
21446. required, NULL implies detached
21447. right associative
21448. covers +Infinity
21449. no size check is necessary
21450. r <- (* f 2) * s <- (* (expt b (- e)) 2) == b^(-e) * 2 [if b==2 -> b^(1-e)] * m+ <- 1 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round
21451. **comment:** XXX: best behavior for real world compatibility?
 label: code-design
21452. 34: setMonth
21453. 20: getSeconds
21454. refcount macros not defined without refcounting, caller must #ifdef now
21455. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is never NULL.
21456. DUK_MEMCMP() is guaranteed to return zero (equal) for zero length * inputs so no zero length check is needed.
21457. **comment:** XXX: shared parsing?
 label: code-design
21458. Note: reject negative zero.
21459. Delete entry in Duktape.modLoaded[] and rethrow.
21460. already closed
21461. '\xffFormals'
21462. 1
21463. XXX: Multiple tv_func lookups are now avoided by making a local * copy of tv_func. Another approach would be to compute an offset * for tv_func from valstack bottom and recomputing the tv_func * pointer quickly as valstack + offset instead of calling duk_get_tval().
21464. DUK_TOK_NEQ
21465. **comment:** XXX: would be nice to omit this jump when the jump is not * reachable, at least in the obvious cases (such as the case * ending with a 'break'. * * Perhaps duk_parse_stmt() could provide some info on whether * the statement is a "dead end"? * * If implemented, just set pc_prevstmt to -1 when not needed.
 label: code-design
21466. no shrink
21467. XXX: duk_to_int() ensures we'll get 8 lowest bits as * as input is within duk_int_t range (capped outside it).
21468. * Ecmascript bytecode
21469. Current require() function
21470. [... errobj]
21471. [... arr]
21472. -> [... escaped_source]
21473. func
21474. ignore fclose() error
21475. DUK_TOK_RETURN
21476. * <https://github.com/svaarala/duktape/issues/127#issuecomment-77863473>
21477. **comment:** Note: not initializing all bytes is normally not an issue: Duktape won't * read or use the uninitialized bytes so valgrind won't issue warnings. * In some special cases a harmless valgrind warning may be issued though. * For example, the DumpHeap debugger command writes out a compiled function's * 'data' area as is, including any uninitialized bytes, which causes a * valgrind warning.
 label: code-design
21478. XXX: to util
21479. Byte length would overflow.
21480. register declarations in first pass
21481. implicit this value always undefined for * declarative environment records.
21482. match_caps == 0 without regexps
21483. Value coercion (in stack): ToInteger(), E5 Section 9.4 * API return value coercion: custom
21484. thr->heap->jl_jmpbuf_ptr is checked by duk_err_longjmp() so we don't * need to check that here. If the value is NULL, a panic occurs because * we can't return.
21485. * Object finalizer
21486. **comment:** never shrinks; auto-adds DUK_VALSTACK_INTERNAL_EXTRA, which is generous
 label: code-design
21487. -> [... proplist enum_obj key]
21488. elems in source and dest
21489. E5 Section 15.4.5.1, step 4

21490. **comment:** * Handle special cases (NaN, infinity, zero).
label: code-design

21491. DUK_USE_ROM_OBJECTS

21492. state for current expression being parsed

21493. **comment:** maintain highest 'used' temporary, needed to figure out nregs of function
label: code-design

21494. DUK_IVAL_XXX

21495. **comment:** Serialize uncovered backing buffer as a null; doesn't * really matter as long we're memory safe.
label: code-design

21496. * Error, fatal, and panic handling.

21497. A number can be loaded either through a constant, using * LDINT, or using LDINT+LDINTX. LDINT is always a size win, * LDINT+LDINTX is not if the constant is used multiple times. * Currently always prefer LDINT+LDINTX over a double constant.

21498. guaranteed by string limits

21499. Slot A

21500. terminates expression; e.g. post-increment/-decrement

21501. ignore arguments, return undefined (E5 Section 15.3.4)

21502. Thread state.

21503. g...v

21504. [...] or [...] errobj (M * undefined)] where M = num_stack_rets - 1

21505. omit

21506. always >= 0

21507. Fixed buffer, no zeroing because we'll fill all the data.

21508. Ecma-to-ecma call possible, may or may not be a tail call. * Avoid C recursion by being clever.

21509. terminal type: no depth check

21510. track cpos while scanning

21511. DUK_TOK_SEQ

21512. curr_pc synced by duk_handle_ecma_call_setup()

21513. **comment:** XXX: could accept numbers larger than 32 bits, e.g. up to 53 bits?
label: code-design

21514. accessor flag not encoded explicitly

21515. For now only needed by the debugger.

21516. * Bytecode instruction representation during compilation * * Contains the actual instruction and (optionally) debug info.

21517. remove value

21518. * Forward declarations

21519. mm <- mp

21520. **comment:** LDINTX is not necessarily in FASTINT range, so * no fast path for now. * * XXX: perhaps restrict LDINTX to fastint range, wider * range very rarely needed.
label: code-design

21521. the entries [new_e_next, new_e_size_adjusted[are left uninitialized on purpose (ok, not gc reachable)

21522. rethrow

21523. * Accessor macros for function specific data areas

21524. Argument validation and func/args offset.

21525. day number for Jan 1 since epoch

21526. DUK_HOBJECT_FLAG_NATIVEFUNCTION varies

21527. compact

21528. **comment:** XXX: this generates quite large code - perhaps select the error * class based on the code and then just use the error 'name'?
label: code-design

21529. require a short (8-bit) reg/const which fits into bytecode B/C slot

21530. **comment:** Function.prototype.bind() should never let this happen, * ugly error message is enough.
label: code-design

21531. "123." is allowed in some formats

21532. Apply ToNumber() to specified index; if ToInteger(val) in [0,99], add * 1900 and replace value at idx_val.

21533. NB: must accept reserved words as property name

21534. no array part

21535. * Misc

21536. error message doesn't matter, ignored anyway

21537. Resolve 'res' directly into the LHS binding, and use * that as the expression value if safe. If not safe, * resolve to a temp/const and copy to LHS.

21538. * Found existing own (non-inherited) plain property. * Do an access control check and update in place.

21539. Currently nothing to free

21540. **comment:** Convenience copies from heap/vm for faster access.
label: code-design

21541. Reply with tvals pushed by request callback

21542. If out of catchstack, cat = thr->catchstack - 1; * new_cat_top will be 0 in that case.

21543. **comment:** XXX: Could just lookup . toJSON() and continue in fast path, * as it would almost never be defined.
label: code-design

21544. **comment:** * Raw write/read macros for big endian, unaligned basic values. * Caller ensures there's enough space. The macros update the pointer * argument automatically on resizes. The idiom seems a bit odd, but * leads to compact code.
label: code-design

21545. **comment:** * "arguments" and "caller" must be mapped to throwers for strict * mode and bound functions (E5 Section 15.3.5). * * XXX: This is expensive to have for every strict function instance. * Try to implement as virtual properties or on-demand created properties.
label: code-design

21546. 'index'

21547. index is above internal string length -> property is fully normal

21548. LF line terminator; CR LF and Unicode lineterms are handled in slow path

21549. * Misc helpers

21550. expt== -1023 -> bitstart=0 (leading 1); * expt== -1024 -> bitstart=-1 (one left of leading 1), etc

21551. bound chain resolved

21552. base

21553. Line format: <time> <entryLev> <loggerName>: <msg>

21554. NEXTENUM jump slot: executed when enum finished

21555. **comment:** prototype should be last, for readability
label: code-design

21556. **comment:** XXX: direct implementation
label: requirement

21557. Error codes.

21558. multi-character sets not allowed as part of ranges, see * E5 Section 15.10.2.15, abstract operation CharacterRange.

21559. * Property lookup

21560. Constants for duk_hashstring().

21561. roughly 1 kB
21562. advance by one character (code point) and one char_offset
21563. no need to coerce
21564. remove key and value
21565. A regexp token value.
21566. large context; around 2kB now
21567. lexer character window helpers
21568. at index 1
21569. trivial cases
21570. duplicate/invalid key checks; returns 1 if syntax error
21571. Here we could remove references to built-ins, but it may not be * worth the effort because built-ins are quite likely to be shared * with another (unterminated) thread, and terminated threads are also * usually garbage collected quite quickly. Also, doing DECREFs * could trigger finalization, which would run on the current thread * and have access to only some of the built-ins. Garbage collection * deals with this correctly already.
21572. Name will be filled from function expression, not by caller. * This case is used by Function constructor and duk_compile() * API with the DUK_COMPILE_FUNCTION option.
21573. original target at entry_top - 1
21574. not emitted
21575. **comment:** XXX: duk_ssize_t would be useful here
label: code-design
21576. comp_ctx->lex.input and comp_ctx->lex.input_length filled by caller
21577. **comment:** XXX: pre-checks (such as no duplicate keys)
label: code-design
21578. To handle deeper indents efficiently, make use of copies we've * already emitted. In effect we can emit a sequence of 1, 2, 4, * 8, etc copies, and then finish the last run. Byte counters * avoid multiply with gap_len on every loop.
21579. **comment:** XXX: TRYCATCH handling should be reworked to avoid creating * an explicit scope unless it is actually needed (e.g. function * instances or eval is executed inside the catch block). This * rework is not trivial because the compiler doesn't have an * intermediate representation. When the rework is done, the * opcode format can also be made more straightforward.
label: code-design
21580. Because new_size != 0, if condition doesn't need to be * (p != NULL || new_size == 0).
21581. * Init lexer
21582. %'
21583. * This test fails on an exotic ARM target; double-to-uint * cast is incorrectly clamped to -signed- int highest value. * *
<https://github.com/svaarala/duktape/issues/336>
21584. DUK_TOK_NUMBER
21585. IEEE exp without bias
21586. 31 bits
21587. call handling
21588. not exposed
21589. No need to check for size of bp_active list, * it's always larger than maximum number of * breakpoints.
21590. **comment:** * Restart execution by reloading thread state. * * Note that 'thr' and any thread configuration may have changed, * so all local variables are suspect and we need to reinitialize. * * The number of local variables should be kept to a minimum: if * the variables are spilled, they will need to be loaded from * memory anyway. * * Any 'goto restart_execution;' code path in opcode dispatch must * ensure 'curr_pc' is synced back to act->curr_pc before the goto * takes place. * * The interpreter must be very careful with memory pointers, as * many pointers are not guaranteed to be 'stable' and may be * reallocated and relocated on-the-fly quite easily (e.g. by a * memory allocation or a property access). * * The following are assumed to have stable pointers: * - the current thread * - the current function * - the bytecode, constant table, inner function table of the * current function (as they are a part of the function allocation) * * The following are assumed to have semi-stable pointers: * - the current activation entry: stable as long as callstack * is not changed (reallocated by growing or shrinking), or * by any garbage collection invocation (through finalizers) * - Note in particular that ANY DECREF can invalidate the * activation pointer, so for the most part a fresh lookup * is required * * The following are not assumed to have stable pointers at all: * - the value stack (registers) of the current thread * - the catch stack of the current thread * * See execution.rst for discussion.
label: code-design
21591. For internal use: get array part value
21592. 10xx xxxx -> invalid
21593. Compiler is responsible for selecting property flags (configurability, * writability, etc).
21594. _Varmap is dense
21595. DUK_TOK_BAND
21596. **comment:** 0x00 ... 0x7f: as is * 0x80: escape generically * 0x81: slow path * 0xa0 ... 0xff: backslash + one char
label: code-design
21597. * Exposed matcher function which provides the semantics of RegExp.prototype.exec(). * * RegExp.prototype.test() has the same semantics as exec() but does not return the * result object (which contains the matching string and capture groups). Currently * there is no separate test() helper, so a temporary result object is created and * discarded if test() is needed. This is intentional, to save code space. * * Input stack: [... re_obj input] * Output stack: [... result]
21598. * Buffers have no internal references. However, a dynamic * buffer has a separate allocation for the buffer. This is * freed by duk_heap_free_heapdr_raw().
21599. indirect eval
21600. Numbers half-way between integers must be rounded towards +Infinity, * e.g. -3.5 must be rounded to -3 (not -4). When rounded to zero, zero * sign must be set appropriately. E5.1 Section 15.8.2.15. * * Note that ANSI C round() is "round to nearest integer, away from zero", * which is incorrect for negative values. Here we make do with floor().
21601. * Some assertions (E5 Section 13.2).
21602. Shared lenient buffer length clamping helper. No negative indices, no * element/byte shifting.
21603. **comment:** XXX: some code might benefit from DUK_SETTEMP_IFTEMP(ctx,x)
label: code-design
21604. preincrement and predecrement
21605. Value would normally be omitted, replace with 'null'.
21606. e.g. require('/foo'), empty terms not allowed
21607. string is internal
21608. * Parse regexp Disjunction. Most of regexp compilation happens here. * * Handles Disjunction, Alternative, and Term productions directly without * recursion. The only constructs requiring recursion are positive/negative * lookaheads, capturing parentheses, and non-capturing parentheses. * * The function determines whether the entire disjunction is a 'simple atom' * (see doc/regexp.rst discussion on 'simple quantifiers') and if so, * returns the atom character length which is needed by the caller to keep * track of its own atom character length. A disjunction with more than one * alternative is never considered a simple atom (although in some cases * that might be the case). * * Return value: simple atom character length or < 0 if not a simple atom. * Appends the bytecode for the disjunction matcher to the end of the temp * buffer. * * Regexp top level structure is: * * Disjunction = Term* * | Term* | Disjunction * * Term = Assertion * | Atom * | Atom Quantifier * * An empty Term sequence is a valid disjunction alternative (e.g. /|||c||/). * * Notes: * * * Tracking of the 'simple-ness' of the current atom vs. the entire * disjunction are separate matters. For instance, the disjunction * may be complex, but individual atoms may be simple. Furthermore, * simple quantifiers are used whenever possible, even if the * disjunction as a whole is complex. * * * The estimate of whether an atom is simple is conservative now, * and it would be possible to expand it. For instance, captures * cause the disjunction to be marked complex, even though captures * -can- be handled by simple quantifiers with some minor modifications. * * * Disjunction 'tainting' as 'complex' is handled at the end of the * main for loop collectively for atoms. Assertions, quantifiers, * and '| tokens need to taint the result manually if necessary. * Assertions cannot add to result char length, only atoms (and * quantifiers) can; currently quantifiers will taint the result * as complex though.
21609. Get a pointer to the current buffer contents (matching current allocation * size). May be NULL for zero size dynamic/external buffer.
21610. success: leave varname in stack

21611. safe
21612. [... closure template len_value]
21613. Note: negative pc values are ignored when patching jumps, so no explicit checks needed
21614. [... func this (crud) retval]
21615. DUK_USE_DATE_PRS_STRPTIME
21616. **comment:** Global case is more complex.
 label: code-design
21617. max, excl. varargs marker
21618. should be a safe way to compute this
21619. not an error
21620. prev_token slot2
21621. side effects, perform in-place
21622. **comment:** XXX: turkish / azeri
 label: code-design
21623. current variable environment (may be NULL if delayed)
21624. May happen in some out-of-memory corner cases.
21625. * Store lexer position for a later rewind
21626. -> [... this timeval]
21627. Avoid using RegExp.prototype methods, as they're writable and * configurable and may have been changed.
21628. Index validation is strict, which differs from duk_equals(). * The strict behavior mimics how instanceof itself works, e.g. * it is a TypeError if rval is not a - callable- object. It would * be somewhat inconsistent if rval would be allowed to be * non-existent without a TypeError.
21629. prev_token slot1
21630. Note1
21631. **comment:** When DUK_USE_HEAPPTR16 (and DUK_USE_REFCOUNT16) is in use, the * struct won't align nicely to 4 bytes. This 16-bit extra field * is added to make the alignment clean; the field can be used by * heap objects when 16-bit packing is used. This field is now * conditional to DUK_USE_HEAPPTR16 only, but it is intended to be * used with DUK_USE_REFCOUNT16 and DUK_USE_DOUBLE_LINKED_HEAP; * this only matter to low memory environments anyway.
 label: code-design
21632. DUK_TOK_IMPORT
21633. * Native call.
21634. input offset tracking
21635. -> [... closure template newobj closure]
21636. fixed arg count: value
21637. chain jumps for case * evaluation and checking
21638. Called for handling both a longjmp() with type DUK_LJ_TYPE_YIELD and * when a RETURN opcode terminates a thread and yields to the resumer.
21639. function call
21640. full 3-byte -> 4-char conversions
21641. regexp compilation limits
21642. **comment:** Current PC, accessed by other functions through thr->ptr_to_curr_pc. * Critical for performance. It would be safest to make this volatile, * but that eliminates performance benefits; aliasing guarantees * should be enough though.
 label: code-design
21643. 0x00: slow path * other: as is
21644. **comment:** XXX: this could be optimized; there is only one call site now though
 label: code-design
21645. reject 'in' token (used for for-in)
21646. Data area, fixed allocation, stable data ptrs.
21647. list of conversion specifiers that terminate a format tag; * this is unfortunately guesswork.
21648. Preliminaries for __proto__ and setPrototypeOf (E6 19.1.2.18 steps 1-4); * magic: 0=setter call, 1=Object.setPrototypeOf
21649. DUK_LEXER_H_INCLUDED
21650. type bit mask: types which certainly produce 'undefined'
21651. XXX: for threads, compact value stack, call stack, catch stack?
21652. **comment:** Lightfuncs are currently supported by coercing to a temporary * Function object; changes will be allowed (the coerced value is * extensible) but will be lost.
 label: code-design
21653. second, parse flags
21654. * Not found: write to global object (non-strict) or ReferenceError * (strict); see E5 Section 8.7.2, step 3.
21655. strict mode restrictions (E5 Section 12.2.1)
21656. **comment:** Forget the previous allocation, setting size to 0 and alloc to * NULL. Caller is responsible for freeing the previous allocation. * Getting the allocation and clearing it is done in the same API * call to avoid any chance of a realloc.
 label: code-design
21657. [regexp]
21658. \xffFinalizer'
21659. **comment:** The entries can either be register numbers or 'null' values. * Thus, no need to DECREF them and get side effects. DECREF'ing * the keys (strings) can cause memory to be freed but no side * effects as strings don't have finalizers. This is why we can * rely on the object properties not changing from underneath us.
 label: code-design
21660. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.
21661. For all combinations: +0 < +0, +0 < -0, -0 < +0, -0 < -0, * steps e, f, and g.
21662. output is never longer than input during resolution
21663. **comment:** XXX: make fast paths optional for size minimization?
 label: code-design
21664. **comment:** XXX: handling for array part missing now; this doesn't affect * compliance but causes array entry writes using defineProperty() * to always abandon array part.
 label: code-design
21665. Fast path the binary case
21666. * Unicode helpers
21667. **comment:** XXX: zero assumption
 label: code-design
21668. * Assertions after
21669. r <- r * s <- (* s B) * m+ <- m+ * m- <- m- * k <- (+ k 1)
21670. 'Math'
21671. 1e8: protects against deeply nested inner functions
21672. insert ranges instruction, range count patched in later
21673. * Validate and convert argument property descriptor (an Ecmascript * object) into a set of defprop_flags and possibly property value, * getter, and/or setter values on the value stack. * * Lightfunc set/get values are coerced to full Functions.
21674. tc1 = false, tc2 = true
21675. after this, return paths should 'goto finished' for decrement

21676. obj_index
21677. zero-width non-joiner
21678. activation has active breakpoint(s)
21679. Constants for built-in string data depacking.
21680. Need to set curr_token.t because lexing regexp mode depends on current * token type. Zero value causes "allow regexp" mode.
21681. start (incl) and end (excl) of trimmed part
21682. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.
21683. interpret e.g. '09' as '0', not NaN
21684. * splice()
21685. no argument given -> leave components untouched
21686. **comment:** This fast path is pretty marginal in practice. * XXX: candidate for removal.
 label: code-design
21687. character represents itself
21688. eat 'return'
21689. **comment:** If thr != NULL, the thr may still be in the middle of * initialization. * XXX: Improve the thread viability test.
 label: code-design
21690. [value offset end]
21691. Relies on NULL encoding to zero.
21692. **comment:** XXX: the current implementation works but is quite clunky; it compiles * to almost 1,4kB of x86 code so it needs to be simplified (better approach, * shared helpers, etc). Some ideas for refactoring: * * - a primitive to convert a string into a regexp matcher (reduces matching * code at the cost of making matching much slower) * - use replace() as a basic helper for match() and split(), which are both * much simpler * - API call to get_prop and to_boolean
 label: code-design
21693. * Compilation and evaluation
21694. Note: Identifier rejects reserved words
21695. -> [obj trap_result]
21696. * Unicode letter is now taken to be the categories: * * Lu, Ll, Lt, Lm, Lo * * (Not sure if this is exactly correct.) * * The ASCII fast path consists of: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z']
21697. caller ensures
21698. * Not found (in registers or record objects). Declare * to current variable environment.
21699. **comment:** 'this' binding is not needed here
 label: requirement
21700. no need to check callable; duk_call() will do that
21701. mark finalizer work list as reachability roots
21702. Initial '{' has been checked and eaten by caller.
21703. '\xffMap'
21704. The directive prologue flag is cleared by default so that it is * unset for any recursive statement parsing. It is only "revived" * if a directive is detected. (We could also make directives only * allowed if 'allow_source_elem' was true.)
21705. 'NaN'
21706. unsigned 31-bit range
21707. DUK_USE_EXPLICIT_NULL_INIT
21708. valgrind whine without this
21709. **comment:** * Helper for calling a user error handler. * * 'thr' must be the currently active thread; the error handler is called * in its context. The valstack of 'thr' must have the error value on * top, and will be replaced by another error value based on the return * value of the error handler. * * The helper calls duk_handle_call() recursively in protected mode. * Before that call happens, no longjumps should happen; as a consequence, * we must assume that the valstack contains enough temporary space for * arguments and such. * * While the error handler runs, any errors thrown will not trigger a * recursive error handler call (this is implemented using a heap level * flag which will "follow" through any coroutines resumed inside the * error handler). If the error handler is not callable or throws an * error, the resulting error replaces the original error (for Duktape * internal errors, duk_error_throw.c further substitutes this error with * a DoubleError which is not ideal). This would be easy to change and * even signal to the caller. * * The user error handler is stored in 'Duktape.errCreate' or * 'Duktape.errThrow' depending on whether we're augmenting the error at * creation or throw time. There are several alternatives to this approach, * see doc/error-objects.rst for discussion. * * Note: since further longjmp()s may occur while calling the error handler * (for many reasons, e.g. a labeled 'break' inside the handler), the * caller can make no assumptions on the thr->heap->lj state after the * call (this affects especially duk_error_throw.c). This is not an issue * as long as the caller writes to the lj state only after the error handler * finishes.
 label: code-design
21710. * Exposed calls
21711. key encountered as a setter
21712. Don't need to sync curr_pc here; duk_new() will do that * when it augments the created error.
21713. one i_step over
21714. Shared prefix for all buffer types.
21715. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. * * Delayed creation (on demand) is handled in duk_js_var.c.
21716. Reject a proxy object as the target because it would need * special handler in property lookups. (ES6 has no such restriction)
21717. must restore reliably before returning
21718. * Basic context initialization. * * Some init values are read from the bytecode header * whose format is (UTF-8 codepoints): * * uint flags * uint nsaved (even, 2n+2 where n = num captures)
21719. return 'res_arr' or 'null'
21720. **comment:** This would be nice, but parsing is faster without resetting the * value slots. The only side effect is that references to temporary * string values may linger until lexing is finished; they're then * freed normally.
 label: code-design
21721. break jump
21722. "123"
21723. Note: the only yield-preventing call is Duktape.Thread.yield(), hence check for 1, not 0
21724. range check not necessary because all 4-bit values are mapped
21725. floor(1.15 * (1 << 10))
21726. expect_eof
21727. -- p_insert -- p_curr * v v * | ... | insert | ... | curr
21728. * Stack constants
21729. call: this(fmt(arg)
21730. Combined step 11 (empty string special case) and 14-15.
21731. w/o refcounting
21732. Example: d=3.5, t=0.5 -> ret = (3 + 1) & 0xfe = 4 & 0xfe = 4 * Example: d=4.5, t=0.5 -> ret = (4 + 1) & 0xfe = 5 & 0xfe = 4
21733. Write fully.
21734. Assume IEEE round-to-even, so that shorter encoding can be used * when round-to-even would produce correct result. By removing * this check (and having low_ok == high_ok == 0) the results would * still be accurate but in some cases longer than necessary.
21735. object established using Function.prototype.bind()
21736. * Heap object refcount finalization. * * When an object is about to be freed, all other objects it refers to must * be decref'd. Refcount finalization does NOT free the object or its inner * allocations (mark-and-sweep shares these helpers), it just manipulates * the refcounts. * * Note that any of the decref's may cause a refcount to drop to zero, BUT * it will not be processed inline; instead, because refzero is already * running, the objects will just be queued to refzero list and processed * later. This eliminates C recursion.

21737. output radix
21738. resume check from proxy target
21739. native function properties
21740. 1e8
21741. Lightfunc flags packing and unpacking.
21742. Helpers exposed for internal use
21743. * Hash function duk_util_hashbytes(). ** Currently, 32-bit MurmurHash2. ** Don't rely on specific hash values; hash function may be endianness * dependent, for instance.
21744. The final object may be a normal function or a lightfunc. * We need to re-lookup tv_func because it may have changed * (also value stack may have been resized). Loop again to * do that; we're guaranteed not to come here again.
21745. **comment:** For objects JSON.stringify() only looks for own, enumerable * properties which is nice for the fast path here. ** For arrays JSON.stringify() uses [[Get]] so it will actually * inherit properties during serialization! This fast path * supports gappy arrays as long as there's no actual inherited * property (which might be a getter etc). ** Since recursion only happens for objects, we can have both * recursion and loop checks here. We use a simple, depth-limited * loop check in the fast path because the object-based tracking * is very slow (when tested, it accounted for 50% of fast path * execution time for input data with a lot of small objects!).
label: code-design
21746. assertion omitted
21747. -> [... obj finalizer]
21748. does not fit into an uchar
21749. need to normalize, may even cancel to 0
21750. Callee/caller are throwers and are not deletable etc. They * could be implemented as virtual properties, but currently * there is no support for virtual properties which are accessors * (only plain virtual properties). This would not be difficult * to change in duk_hobject_props, but we can make the throwers * normal, concrete properties just as easily. ** Note that the specification requires that the *same* thrower * built-in object is used here! See E5 Section 10.6 main * algorithm, step 14, and Section 13.2.3 which describes the * thrower. See test case test-arguments-throwers.js.
21751. used for array, stack, and entry indices
21752. [... name reg_bind]
21753. Select copy mode. Must take into account element * compatibility and validity of the underlying source * buffer.
21754. invalid value which never matches
21755. Arrays never have other exotic behaviors.
21756. 0xxx xxxx [7 bits]
21757. [... env target]
21758. Setup builts from ROM objects. All heaps/threads will share * the same readonly objects.
21759. **comment:** alloc size in elements
label: code-design
21760. * Init remaining result fields * * 'nregs' controls how large a register frame is allocated. ** 'nargs' controls how many formal arguments are written to registers: * r0, ... r(nargs-1). The remaining registers are initialized to * undefined.
21761. note: caller 'sp' is intentionally not updated here
21762. Underlying buffer (refcounted), may be NULL.
21763. actual chars dropped (not just NUL term)
21764. free-form
21765. formals for 'func' (may be NULL if func is a C function)
21766. **comment:** bytes of indent available for copying
label: code-design
21767. * Valstack manipulation for results.
21768. [obj key desc value get set curr_value varname]
21769. * Add '_Tracedata to an error on the stack top.
21770. force executor restart to recheck breakpoints; used to handle function returns (see GH-303)
21771. Lexer context. Same context is used for Ecmascript and Regexp parsing.
21772. **comment:** XXX: spilling
label: code-design
21773. Note: rely on index ordering
21774. * Get enumerated keys in an Ecmascript array. Matches Object.keys() behavior * described in E5 Section 15.2.3.14.
21775. NB: duk_lexer_getpoint() is a macro only
21776. dangerous: must only lower (temp_max not updated)
21777. x <- x + y, use t as temp
21778. * Init argument related properties
21779. add BC*2^16
21780. initial stringtable size, must be prime and higher than DUK_UTIL_MIN_HASH_PRIME
21781. char format: use int
21782. Clamping only necessary for 32-bit ints.
21783. maximum size check is handled by callee
21784. -> [...] val root val]
21785. Note: start_offset/end_offset can still be < 0 here.
21786. cat_idx catcher is kept, even for finally
21787. "null", "true", and "false" are always reserved words. * Note that "get" and "set" are not!
21788. AssignmentExpression
21789. DUK_TOK_MOD
21790. **comment:** XXX: this is pointless here because pass 1 is throw-away
label: code-design
21791. Possibly truncated; there is no explicit truncation * marker so this is the best we can do.
21792. 'fatal'
21793. * Plain, boring reachable object.
21794. value of re_ctx->captures at start of atom
21795. * Find an existing key from entry part either by linear scan or by * using the hash index (if it exists). ** Sets entry index (and possibly the hash index) to output variables, * which allows the caller to update the entry and hash entries in-place. * If entry is not found, both values are set to -1. If entry is found * but there is no hash part, h_idx is set to -1.
21796. pc_label
21797. 'configurable'
21798. * Object prototype
21799. 'this' binding exists
21800. XXX: direct manipulation, or duk_replace_tval()
21801. fill window
21802. **comment:** * Helper for handling a "bound function" chain when a call is being made. ** Follows the bound function chain until a non-bound function is found. * Prepends the bound arguments to the value stack (at idx_func + 2), * updating 'num_stack_args' in the process. The 'this' binding is also * updated if necessary (at idx_func + 1). Note that for constructor calls * the 'this' binding is never updated by [[BoundThis]]. ** XXX: bound function chains could be collapsed at bound function creation * time so that each bound function would point directly to a non-bound * function. This would make call time handling much easier.
label: code-design

21803. * Constructor arguments are currently somewhat compatible with * (keep it that way if possible): * * http://nodejs.org/api/buffer.html * * Note that the ToBuffer() coercion (duk_to_buffer()) does NOT match * the constructor behavior.

21804. * Emit compiled regexp header: flags, ncaptures * (insertion order inverted on purpose)

21805. * Logging support

21806. [key result]

21807. unchanged by Duktape.Thread.resume()

21808. arguments object would be accessible; note that shadowing * bindings are arguments or function declarations, neither * of which are deletable, so this is safe.

21809. Return 1: got EOM

21810. include removed: duk_internal.h

21811. NOTE: length may be zero

21812. SCANBUILD: NULL pointer dereference, doesn't actually trigger, * asserted above.

21813. Alignment guarantee

21814. #DUK_USE_PREFER_SIZE

21815. Note1: the specification doesn't require matching a time form with * just hours ("HH"), but we accept it here, e.g. "2012-01-02T12Z". * * Note2: the specification doesn't require matching a timezone offset * with just hours ("HH"), but accept it here, e.g. "2012-01-02T03:04:05+02"

21816. * Mark the heap.

21817. silence scan-build warning

21818. varname is popped by above code

21819. Estimating the result size beforehand would be costly, so * start with a reasonable size and extend as needed.

21820. Decode failed.

21821. * Init stringcache

21822. [... constructor arg1 ... argN final_cons]

21823. **comment:** XXX: lithuanian, explicit dot rules
label: code-design

21824. XXX: similar coercion issue as in DUK_TOK_PERIOD

21825. **comment:** Wipe capture range and save old values for backtracking. * * XXX: this typically happens with a relatively small idx_count. * It might be useful to handle cases where the count is small * (say <= 8) by saving the values in stack instead. This would * reduce memory churn and improve performance, at the cost of a * slightly higher code footprint.
label: code-design

21826. source out-of-bounds (but positive)

21827. 'with' binding has no catch clause, so can't be here unless a normal try-catch

21828. second incref for the entry reference

21829. **comment:** XXX: declvar takes an duk_tval pointer, which is awkward and * should be reworked.
label: code-design

21830. Get.

21831. DUK_TOK_PRIVATE

21832. randomized pivot selection

21833. * Check whether the property already exists in the prototype chain. * Note that the actual write goes into the original base object * (except if an accessor property captures the write).

21834. These are not needed when only Duktape.Buffer is supported.

21835. duk_hobject specific fields.

21836. [val] -> []

21837. Map DUK_HBUFFEROBJECT_ELEM_xxx to duk_hobject class number. * Sync with duk_hbufferobject.h and duk_hobject.h.

21838. * Shared exit path

21839. force_exponential

21840. As an initial implementation, write flush after every EOM (and the * version identifier). A better implementation would flush only when * Duktape is finished processing messages so that a flush only happens * after all outbound messages are finished on that occasion.

21841. DUK_INVALID_INDEX won't be accepted as a valid index.

21842. * Heap thread object representation. * * duk_hthread is also the 'context' (duk_context) for exposed APIs * which mostly operate on the topmost frame of the value stack.

21843. DUK_USE_BUFFEROBJECT_SUPPORT

21844. As an initial implementation, read flush after exiting the message * loop. If transport is broken, this is a no-op (with debug logs).

21845. * Figure out how generated digits match up with the mantissa, * and then perform rounding. If mantissa overflows, need to * recompute the exponent (it is bumped and may overflow to * infinity). * * For normal numbers the leading '1' is hidden and ignored, * and the last bit is used for rounding: * * rounding pt * <-----52----->| * 1 x x x x ... x x x x |y ==> x x x x ... x x x x * * For denormals, the leading '1' is included in the number, * and the rounding point is different: * * rounding pt * <-52 or less--->| * 1 x x x x ... x x|x y ==> 0 0 ... 1 x x ... x x * * The largest denormals will have a mantissa beginning with * a '1' (the explicit leading bit); smaller denormals will * have leading zero bits. * * If the exponent would become too high, the result becomes * Infinity. If the exponent is so small that the entire * mantissa becomes zero, the result becomes zero. * * Note: the Dragon4 'k' is off-by-one with respect to the IEEE * exponent. For instance, k==0 indicates that the leading '1' * digit is at the first binary fraction position (0.1xxx...); * the corresponding IEEE exponent would be -1.

21846. string hash

21847. Relookup in case coerce_func() has side effects, e.g. ends up coercing an object

21848. 0 = no update

21849. regexp opcodes

21850. 'function'

21851. recursion limit

21852. utf-8 validation ensures these

21853. count, not including sep

21854. string data is external (duk_hstring_external)

21855. currently paused: talk with debug client until step/resume

21856. duk_concat() coerces arguments with ToString() in correct order

21857. 2 to 11

21858. guaranteed to finish

21859. const flag for B

21860. one i_step added at top of loop

21861. 'res' may be NULL if new allocation size is 0.

21862. **comment:** Shared entry code for many Array built-ins. Note that length is left * on stack (it could be popped, but that's not necessary).
label: code-design

21863. thread

21864. assume PC is at most 32 bits and non-negative

21865. 1 0 <2 bits>

21866. **comment:** * Shared handling for logical AND and logical OR. * * args = (truthval << 8) + rbp * * Truthval determines when to skip right-hand-side. * For logical AND truthval=1, for logical OR truthval=0. * * See doc/compiler.rst for discussion on compiling logical * AND and OR expressions. The approach here is very simplistic, * generating extra jumps and multiple evaluations of truth values, * but generates code on-the-fly with only local back-patching. * * Both logical AND and OR are syntactically left-associated. * However, logical ANDs are compiled as right associative * expressions, i.e. "A && B && C" as "A && (B && C)", to allow * skip jumps to skip over the entire tail. Similarly for logical OR.
label: code-design

21867. NULL obj->p is OK

21868. * In strict mode E5 protects 'eval' and 'arguments' from being * assigned to (or even declared anywhere). Attempt to do so * should result in a compile time SyntaxError. See the internal * design documentation for details. * * Thus, we should never come here, run-time, for strict code, * and name 'eval' or 'arguments'.
21869. Assume that thr->valstack_bottom has been set-up before getting here.
21870. len: 9
21871. * E5 Section 11.4.9
21872. deal with weak references first
21873. duk_push_sprintf constants
21874. negative: dst info not available
21875. topmost element is the result array already
21876. Fast path, handle units with just actual encoding characters.
21877. out of spec, must be configurable
21878. * Debug logging after adjustment.
21879. * Pushers
21880. **comment:** * Note: each API operation potentially resizes the callstack, * so be careful to re-lookup after every operation. Currently * these is no issue because we don't store a temporary 'act' * pointer at all. (This would be a non-issue if we operated * directly on the array part.)
label: code-design
21881. implicit_return_value
21882. XXX: direct valstack write
21883. [... source? filename]
21884. No resize has occurred so temp_desc->e_idx is still OK
21885. lookup for 0x000a above assumes shortest encoding now
21886. XXX: E5.1 Section 11.1.4 coerces the final length through * ToUint32() which is odd but happens now as a side effect of * 'arr_idx' type.
21887. Read tvals from the message and push them onto the valstack, * then call the request callback to process the request.
21888. true, despite side effect resizes
21889. the stack is unbalanced here on purpose; we only rely on the * initial two values: [name this].
21890. [start end str]
21891. source is eval code (not global)
21892. switch initial byte
21893. Note: pc is unsigned and cannot be negative
21894. * Casting
21895. **comment:** * User declarations, e.g. prototypes for user functions used by Duktape * macros. Concretely, if DUK_USE_PANIC_HANDLER is used and the macro * value calls a user function, it needs to be declared for Duktape * compilation to avoid warnings.
label: code-design
21896. Longjmp handling has restored jmpbuf_ptr.
21897. * Round a number upwards to a prime (not usually the nearest one). * * Uses a table of successive 32-bit primes whose ratio is roughly * constant. This keeps the relative upwards 'rounding error' bounded * and the data size small. A simple 'predict-correct' compression is * used to compress primes to one byte per prime. See genhashsizes.py * for details. * * The minimum prime returned here must be coordinated with the possible * probe sequence steps in duk_hobject and duk_heap stringtable.
21898. -> [... varname val this]
21899. processed one or more messages
21900. [... v1(filename) v2(line+flags)]
21901. must have been, since in array part
21902. slower but more compact variant
21903. TypedArray (or other non-ArrayBuffer duk_hbufferobject). * Conceptually same behavior as for an Array-like argument, * with a few fast paths.
21904. **comment:** parameter passing, not thread safe
label: requirement
21905. **comment:** Two temporaries are preallocated here for variants 3 and 4 which need * registers which are never clobbered by expressions in the loop * (concretely: for the enumerator object and the next enumerated value). * Variants 1 and 2 "release" these temps.
label: code-design
21906. Note: decimal number may start with a period, but must be followed by a digit
21907. main reachability roots
21908. DUK_HBUFFER_H_INCLUDED
21909. Inputs: explicit arguments (nargs), +1 for key, +2 for obj_index/nargs passing. * If the value stack does not contain enough args, an error is thrown; this matches * behavior of the other protected call API functions.
21910. [value offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
21911. placed in duk_heapdr_string
21912. * Main mark-and-sweep function. * * 'flags' represents the features requested by the caller. The current * heap->mark_and_sweep_base_flags is ORed automatically into the flags; * the base flags mask typically prevents certain mark-and-sweep operations * to avoid trouble.
21913. 'private'
21914. toJSON() can also be a lightfunc
21915. Special behavior for 'caller' property of (non-bound) function objects * and non-strict Arguments objects: if 'caller' -value- (!) is a strict * mode function, throw a TypeError (E5 Sections 15.3.5.4, 10.6). * Quite interestingly, a non-strict function with no formal arguments * will get an arguments object -without- special 'caller' behavior! * * The E5.1 spec is a bit ambiguous if this special behavior applies when * a bound function is the base value (not the 'caller' value): Section * 15.3.4.5 (describing bind()) states that [[Get]] for bound functions * matches that of Section 15.3.5.4 ([[Get]] for Function instances). * However, Section 13.3.5.4 has "NOTE: Function objects created using * Function.prototype.bind use the default [[Get]] internal method." * The current implementation assumes this means that bound functions * should not have the special [[Get]] behavior. * * The E5.1 spec is also a bit unclear if the TypeError throwing is * applied if the 'caller' value is a strict bound function. The * current implementation will throw even for both strict non-bound * and strict bound functions. * * See test-dev-strict-func-as-caller-prop-value.js for quite extensive * tests. * * This exotic behavior is disabled when the non-standard 'caller' property * is enabled, as it conflicts with the free use of 'caller'.
21916. move pivot out of the way
21917. final configuration
21918. -1 == not set, -2 == pending (next statement list)
21919. **comment:** temporary, must be signed and 32-bit to hold Unicode code points
label: code-design
21920. NB: 'length' property is automatically updated by the array setup loop
21921. index for next new key ([0,e_next[are gc reachable)
21922. idx is unsigned, < 0 check is not necessary
21923. indirect opcode follows direct
21924. * toFixed(), toExponential(), toPrecision()
21925. * Helpers for UTF-8 handling * * For bytecode readers the duk_uint32_t and duk_int32_t types are correct * because they're used for more than just codepoints.
21926. follow prototype chain
21927. [value]
21928. XXX: incref by count (here 2 times)
21929. equals
21930. true even with reattach
21931. Fatal error handling, called e.g. when a longjmp() is needed but * lj.jmpbuf_ptr is NULL. fatal_func must never return; it's not * declared as "noreturn" because doing that for typedefs is a bit * challenging portability-wise.
21932. **comment:** XXX: very messy now, but works; clean up, remove unused variables (nomimally * used so compiler doesn't complain).

label: code-design

21933. * Note: type is treated as a field separate from flags, so some masking is * involved in the macros below.
21934. Compute final offset in seconds, positive if local time ahead of * UTC (returned value is UTC-to-local offset). * * difftime() returns a double, so coercion to int generates quite * a lot of code. Direct subtraction is not portable, however. * XXX: allow direct subtraction on known platforms.
21935. **comment:** XXX: this is a very narrow check, and doesn't cover * zeroes, subnormals, infinities, which compare normally.
label: code-design
21936. Don't allow a zero divisor. Fast path check by * "verifying" with multiplication. Also avoid zero * dividend to avoid negative zero issues (0 / -1 = -0 * for instance).
21937. numeric value of a hex digit (also covers octal and decimal digits)
21938. DUK_TOK_MUL_EQ
21939. DUK_USE_DEBUG
21940. **comment:** XXX: optimize to use direct reads, i.e. avoid * value stack operations.
label: code-design
21941. 0x00: finish (non-white) * 0x01: continue
21942. %lu
21943. [... arg1 ... argN]
21944. executor interrupt running (used to avoid nested interrupts)
21945. [... func this]
21946. Constants: variable size encoding.
21947. * Variable declarations. * * Unlike function declarations, variable declaration values don't get * assigned on entry. If a binding of the same name already exists, just * ignore it silently.
21948. **comment:** XXX: duk_small_uint_t would be enough for this loop
label: code-design
21949. no res->strs[]
21950. * Debug dumping of duk_heap.
21951. Detach a debugger if attached (can be called multiple times) * safely.
21952. if 1, doing a string-to-number; else doing a number-to-string
21953. **comment:** Faster alternative: avoid making a temporary copy of tvptr_dst and use * fast incref/decref macros.
label: code-design
21954. * duk_hbuffer allocation and freeing.
21955. stash to bottom of value stack to keep new_env reachable for duration of eval
21956. step 4.a
21957. Handle a RETURN opcode. Avoid using longjmp() for return handling because * it has a measurable performance impact in ordinary environments and an extreme * impact in Emscripten (GH-342). Return value is on value stack top.
21958. exports
21959. Note: must behave like a no-op with NULL and any pointer * returned from malloc/realloc with zero size.
21960. **comment:** XXX: use duk_is_valid_index() instead?
label: code-design
21961. eat closing bracket
21962. * Object.isSealed() and Object.isFrozen() (E5 Sections 15.2.3.11, 15.2.3.13) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: all virtual (non-concrete) properties are currently non-configurable * and non-writable (and there are no accessor virtual properties), so they don't * need to be considered here now.
21963. * The switch statement is pretty messy to compile. * See the helper for details.
21964. re_ctx->captures at start and end of atom parsing. * Since 'captures' indicates highest capture number emitted * so far in a DUK_REOP_SAVE, the captures numbers saved by * the atom are:]start_captures,end_captures].
21965. Loop check using a hybrid approach: a fixed-size visited[] array * with overflow in a loop check object.
21966. first value: comma must not precede the value
21967. actually used, non-NULL keys
21968. r <- (2 * b) * f
21969. [... re_obj input bc saved_buf res_obj]
21970. * Create wrapper object and serialize
21971. t1 <- t1 - s
21972. * Label handling * * Labels are initially added with flags prohibiting both break and continue. * When the statement type is finally uncovered (after potentially multiple * labels), all the labels are updated to allow/prohibit break and continue.
21973. 112.5%, i.e. new size less than 12.5% higher -> fast resize
21974. * URI handling
21975. * ASCII character constants * * C character literals like 'x' have a platform specific value and do * not match ASCII (UTF-8) values on e.g. EBCDIC platforms. So, use * these (admittedly awkward) constants instead. These constants must * also have signed values to avoid unexpected coercions in comparisons. * * <http://en.wikipedia.org/wiki/ASCII>
21976. 32-bit value
21977. string intern table (weak refs)
21978. **comment:** * Shared error message strings * * To minimize code footprint, try to share error messages inside Duktape * code. Modern compilers will do this automatically anyway, this is mostly * for older compilers.
label: code-design
21979. Convert indices to byte offsets.
21980. T'
21981. Fast path: source is a TypedArray (or any bufferobject).
21982. Note: object cannot be a finalizable unreachable object, as * they have been marked temporarily reachable for this round, * and are handled above.
21983. 'var'
21984. [args(n) [crud] formals arguments map mappednames]
21985. **comment:** Note: 'this' is not necessarily an Array object. The push() * algorithm is supposed to work for other kinds of objects too, * so the algorithm has e.g. an explicit update for the 'length' * property which is normally "magical" in arrays.
label: code-design
21986. **comment:** * XXX: array indices are mostly typed as duk_uint32_t here; duk_uarridx_t * might be more appropriate.
label: code-design
21987. 'pointer'
21988. Return value would be pointless: because throw_flag==1, we always * throw if the identifier doesn't resolve.
21989. PC points to next instruction, find offending PC, * PC == 0 for native code.
21990. DUK_UNICODE_H_INCLUDED
21991. **comment:** dead code, but ensures portability (see Linux man page notes)
label: code-design
21992. DUK__ALLOW_AUTO_SEMI_ALWAYS workaround
21993. end of valstack allocation (exclusive)
21994. previous value is assumed to be garbage, so don't touch it
21995. * Scan number and setup for Dragon4. * * The fast path case is detected during setup: an integer which * can be converted without rounding, no net exponent. The fast * path could be implemented as a separate scan, but may not really * be worth it: the multiplications for building 'f' are not * expensive when 'f' is small. * * The significand ('f') must contain enough bits of (apparent) * accuracy, so that Dragon4 will generate enough binary output digits. * For decimal numbers, this means generating a 20-digit significand, * which should yield enough practical accuracy to parse IEEE doubles. * In fact, the Ecmascript specification explicitly

allows an * implementation to treat digits beyond 20 as zeroes (and even * to round the 20th digit upwards). For non-decimal numbers, the * appropriate number of digits has been precomputed for comparable * accuracy. ** Digit counts: * [dig_lzero] * | * .+---[dig_prec]---. * | || | * 0000123.45678901234567890123456 * | | | | * `--+-'`----[dig_frac]-----`-+-`* | | * [dig_whole] [dig_expt] * * dig_frac and dig_expt are -1 if not present * dig_lzero is only computed for whole number part * * Parsing state * * Parsing whole part dig_frac < 0 AND dig_expt < 0 * Parsing fraction part dig_frac >= 0 AND dig_expt < 0 * Parsing exponent part dig_expt >= 0 (dig_frac may be < 0 or >= 0) * * Note: in case we hit an implementation limit (like exponent range), * we should throw an error, NOT return NaN or Infinity. Even with * very large exponent (or significand) values the final result may be * finite, so NaN/Infinity would be incorrect.

21996. **comment:** suppress warning, not used

label: code-design

21997. Buffer/string -> compare contents.

21998. * Update cache entry (allocating if necessary), and move the * cache entry to the first place (in an "LRU" policy).

21999. finished jump

22000. string is ASCII, clen == blen

22001. DUK_USE_HOBJECT_HASH_PART || DUK_USE_STRTAB_PROBE

22002. * Since character data is only generated by decoding the source or by * the compiler itself, we rely on the input codepoints being correct * and avoid a check here.

* * Character data can also come here through decoding of Unicode * escapes ("udead\ubeef") so all 16-bit unsigned values can be * present, even when the source file itself is strict UTF-8.

22003. **comment:** XXX: more emergency behavior, e.g. find smaller hash sizes etc

label: code-design

22004. * Copy array elements to new array part.

22005. For strings, special form for short lengths.

22006. Setup function properties.

22007. **comment:** * In a fast check we assume old_size equals old_used (i.e., existing * array is fully dense). ** Slow check if: * * (new_size - old_size) / old_size > limit * new_size - old_size > limit * old_size * new_size > (1 + limit) * old_size || limit' is 3 bits fixed point * new_size > (1 + (limit' / 8)) * old_size || * 8 * 8 * new_size > (8 + limit') * old_size || : 8 * new_size > (8 + limit') * (old_size / 8) * new_size > limit' * (old_size / 8) || limit" = 9 -> max 25% increase * arr_idx + 1 > limit" * (old_size / 8) ** This check doesn't work well for small values, so old_size is rounded * up for the check (and the '+ 1' of arr_idx can be ignored in practice): * * arr_idx > limit" * ((old_size + 7) / 8)

label: code-design

22008. **comment:** * A few notes on the algorithm: ** - Terms are not allowed to begin with a period unless the term * is either '.' or '..'. This simplifies implementation (and * is within CommonJS modules specification). ** - There are few output bound checks here. This is on purpose: * the resolution input is length checked and the output is never * longer than the input. The resolved output is written directly * over the input because it's never longer than the input at any * point in the algorithm. ** - Non-ASCII characters are processed as individual bytes and * need no special treatment. However, U+0000 terminates the * algorithm; this is not an issue because U+0000 is not a * desirable term character anyway.

label: code-design

22009. * Utilities

22010. * Helpers ** The fast path checks are done within a macro to ensure "inlining" * while the slow path actions use a helper (which won't typically be * inlined in size optimized builds).

22011. **comment:** If the platform doesn't support the entire Ecmascript range, we need * to return 0 so that the caller can fall back to the default formatter. ** For now, assume that if time_t is 8 bytes or more, the whole Ecmascript * range is supported. For smaller time_t values (4 bytes in practice), * assumes that the signed 32-bit range is supported. ** XXX: detect this more correctly per platform. The size of time_t is * probably not an accurate guarantee of strftime() supporting or not * supporting a large time range (the full Ecmascript range).

label: code-design

22012. **comment:** XXX: now that pc2line is used by the debugger quite heavily in * checked execution, this should be optimized to avoid value stack * and perhaps also implement some form of pc2line caching (see * future work in debugger.rst).

label: code-design

22013. * Case clause. ** Note: cannot use reg_case as a temp register (for SEQ target) * because it may be a constant.

22014. **comment:** XXX: not sure what the correct semantic details are here, * e.g. handling of missing values (gaps), handling of non-array * trap results, etc. ** For keys, we simply skip non-string keys which seems to be * consistent with how e.g. Object.keys() will process proxy trap * results (ES6, Section 19.1.2.14).

label: code-design

22015. **comment:** XXX: 'copy properties' API call?

label: code-design

22016. * Determine whether to use the constructor return value as the created * object instance or not.

22017. Stack top contains plain value

22018. make current token the previous; need to fiddle with valstack "backing store"

22019. '{_func:true}'

22020. prediction: portable variant using doubles if 64-bit values not available

22021. [... arr val]

22022. Casting convenience.

22023. The stack has a variable shape here, so force it to the * desired one explicitly.

22024. **comment:** XXX: does not work if thr->catchstack is NULL

label: code-design

22025. Object.getOwnPropertyNames

22026. 'undefined'

22027. **comment:** XXX: with 'caller' property the callstack would need * to be unwound to update the 'caller' properties of * functions in the callstack.

label: code-design

22028. Overestimate required size; debug code so not critical to be tight.

22029. **comment:** no special formatting

label: code-design

22030. proplist is very rare

22031. * Handling DUK_OP_ADD this way is more compact (experimentally) * than a separate case with separate argument decoding.

22032. currently processing messages or breakpoints: don't enter message processing recursively (e.g. no breakpoints when processing debugger eval)

22033. DUK_USE_JC

22034. function refers to 'arguments' identifier

22035. don't want an intermediate exposed state with invalid pc

22036. Use loop to minimize code size of relookup after bound function case

22037. 'Undefined'

22038. **comment:** * Variable already declared, ignore re-declaration. * The only exception is the updated behavior of E5.1 for * global function declarations, E5.1 Section 10.5, step 5.e. * This behavior does not apply to global variable declarations.

label: code-design

22039. **comment:** Avoid fake finalization for the duk_refcount_fake_finalizer function * itself, otherwise we're in infinite recursion.

label: code-design

22040. **comment:** Borrow is detected based on wrapping which relies on exact 32-bit * types.

label: code-design

22041. if abandon_array, new_a_size must be 0

22042. Parse enumeration target and initialize enumerator. For 'null' and 'undefined', * INITENUM will creates a 'null' enumerator which works like an empty enumerator * (E5 Section 12.6.4, step 3). Note that INITENUM requires the value to be in a * register (constant not allowed).

22043. pointer is aligned, guaranteed for fixed buffer

22044. See MPUTOBJ comments above.

22045. Pause on the next opcode executed. This is always safe to do even * inside the debugger message loop: the interrupt counter will be reset * to its proper value when the message loop exits.

22046. Ensure there is internal valstack spare before we exit; this may * throw an alloc error. The same guaranteed size must be available * as before the call. This is not optimal now: we store the valstack * allocated size during entry; this value may be higher than the * minimal guarantee for an application.

22047. \xffPc2line'

22048. **comment:** XXX: better coercion
label: code-design

22049. guaranteed to finish, as hash is never full

22050. shared flags for a subset of types

22051. nothing to incref

22052. The short string/integer initial bytes starting from 0x60 don't have * defines now.

22053. **comment:** XXX: use a helper for prototype traversal; no loop check here
label: code-design

22054. FunctionDeclaration: not strictly a statement but handled as such. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().

22055. unpack args

22056. hash lookup

22057. * Cleanup: restore original function, restore valstack state.

22058. don't touch property attributes or hash part

22059. Note: duk_uint8_t type yields larger code

22060. * The error object has been augmented with a traceback and other * info from its creation point -- usually another thread. The * error handler is called here right before throwing, but it also * runs in the resumer's thread. It might be nice to get a traceback * from the resumee but this is not the case now.

22061. Push function object, init flags etc. This must match * duk_js_push_closure() quite carefully.

22062. return arg as-is

22063. NULL accepted

22064. Lightfunc, not blamed now.

22065. 0 = no throw

22066. **comment:** XXX: macros
label: code-design

22067. Interrupt counter for triggering a slow path check for execution * timeout, debugger interaction such as breakpoints, etc. The value * is valid for the current running thread, and both the init and * counter values are copied whenever a thread switch occurs. It's * important for the counter to be conveniently accessible for the * bytecode executor inner loop for performance reasons.

22068. start value for current countdown

22069. maximum recursion depth for loop detection stacks

22070. us

22071. [... holder name val]

22072. Allow automatic detection of hex base ("0x" or "0X" prefix), * overrides radix argument and forces integer mode.

22073. * Init arguments properties, map, etc.

22074. **comment:** NUL terminator handling doesn't matter here
label: code-design

22075. **comment:** XXX: error handling is incomplete. It would be cleanest if * there was a setjmp catchpoint, so that all init code could * freely throw errors. If that were the case, the return code * passing here could be removed.
label: code-design

22076. keep in on valstack, use borrowed ref below

22077. [... Logger clog logfunc clog(=this) msg]

22078. -> [... func this arg1 ... argN _Args]

22079. * Found * * Arguments object has exotic post-processing, see E5 Section 10.6, * description of [[GetOwnProperty]] variant for arguments.

22080. compiler's responsibility

22081. [... func this arg1 ... argN] (not tail call) * [this | arg1 ... argN] (tail call) * * idx_args updated to match

22082. func is NULL for lightfuncs

22083. shared exit path now

22084. Finish the wrapped module source. Force module.filename as the * function .filename so it gets set for functions defined within a * module. This also ensures loggers created within the module get * the module ID (or overridden filename) as their default logger name. * (Note capitalization: .filename matches Node.js while .fileName is * used elsewhere in Duktape.)

22085. **comment:** This limitation would be fixable but adds unnecessary complexity.
label: code-design

22086. -> [O toISOString O]

22087. * Finish

22088. -> [timeval this timeval]

22089. cached: valstack_end - valstack (in entries, not bytes)

22090. **comment:** XXX: awkward, especially the bunch of separate output values -> output struct?
label: code-design

22091. According to E5.1 Section 15.4.4.4 nonexistent trailing * elements do not affect 'length' of the result. Test262 * and other engines disagree, so update idx_last here too.

22092. * Bitstream encoder.

22093. * Once the whole refzero cascade has been freed, check for * a voluntary mark-and-sweep.

22094. Resolve start/end offset as element indices first; arguments * at idx_start/idx_end are element offsets. Working with element * indices first also avoids potential for wrapping.

22095. IdentifierStart production with ASCII and non-BMP excluded

22096. temp object for tracking / detecting duplicate keys

22097. negative

22098. * First create all built-in bare objects on the empty valstack. * * Built-ins in the index range [0,DUK_NUM_BUILTINS-1] have value * stack indices matching their eventual thr->builtins[] index. * * Built-ins in the index range [DUK_NUM_BUILTINS,DUK_NUM_ALL_BUILTINS] * will exist on the value stack during init but won't be placed * into thr->builtins[]. These are objects referenced in some way * from thr->builtins[] roots but which don't need to be indexed by * Duktape through thr->builtins[] (e.g. user custom objects).

22099. DUK_BUFOBJ_INT8ARRAY

22100. DUK_USE_ROM_STRINGS

22101. get pointer offsets for tweaking below

22102. Note: not normalized, but duk_push_number() will normalize

22103. NULL if not an object

22104. * Exposed number-to-string API * * Input: [number] * Output: [string]

22105. XXX: could add fast path for u8 compatible views

22106. Note2

22107. 'resume'

22108. should never happen for a strict callee

22109. **comment:** XXX: This is VERY inefficient now, and should be e.g. a * binary search or perfect hash, to be fixed.
label: code-design

22110. [... env]

22111. Lookup an identifier name in the current varmap, indicating whether the * identifier is register-bound and if not, allocating a constant for the * identifier name. Returns 1 if register-bound, 0 otherwise. Caller can * also check (`out_reg_varbind >= 0`) to check whether or not identifier is * register bound. The caller must NOT use `out_rc_varname` at all unless * return code is 0 or `out_reg_varbind` is < 0; this is because `out_rc_varname` * is unsigned and doesn't have a "unused" / none value.

22112. 'export'

22113. DUK_ERR_URI_ERROR: no macros needed

22114. array case is handled comprehensively above

22115. * Bytecode executor call. * * Execute bytecode, handling any recursive function calls and * thread resumptions. Returns when execution would return from * the entry level activation. When the executor returns, a * single return value is left on the stack top. * * The only possible longjmp() is an error (DUK_LJ_TYPE_THROW), * other types are handled internally by the executor.

22116. **comment:** XXX: refactor into an internal helper, pretty awkward
label: code-design

22117. Allocate a new valstack. * * Note: cannot use a plain DUK_REALLOC() because a mark-and-sweep may * invalidate the original thr->valstack base pointer inside the realloc * process. See doc/memory-management.rst

22118. Note that this is the same operation for strict and loose equality: * - E5 Section 11.9.3, step 1.c (loose) * - E5 Section 11.9.6, step 4 (strict)

22119. force_no_namebind

22120. DUK_TOK_NEW

22121. * Breakpoint and step state checks

22122. **comment:** XXX: this behavior is quite useless now; it would be nice to be able * to create pointer values from e.g. numbers or strings. Numbers are * problematic on 64-bit platforms though. Hex encoded strings?
label: code-design

22123. global object doesn't have array part

22124. **comment:** Then evaluate RHS fully (its value becomes the expression value too). * Technically we'd need the side effect safety check here too, but because * we always throw using INVLHS the result doesn't matter.
label: code-design

22125. **comment:** XXX: specific getter
label: code-design

22126. [... escape_source bytecode]

22127. First we need to insert a jump in the middle of previously * emitted code to get the control flow right. No jumps can * cross the position where the jump is inserted. See doc/compiler.rst * for discussion on the intricacies of control flow and side effects * for variants 3 and 4.

22128. Cannot simulate individual finalizers because finalize_list only * contains objects with actual finalizers. But simulate side effects * from finalization by doing a bogus function call and resizing the * stacks.

22129. lastIndex is initialized to zero by new RegExp()

22130. **comment:** Note: we could return the hash index here too, but it's not * needed right now.
label: code-design

22131. >>>

22132. [... closure template formals]

22133. inherit

22134. filename being compiled (ends up in functions' '_filename' property)

22135. filter out flags from exprop rbp_flags here to save space

22136. **comment:** * Debugging macros, DUK_DPRINT() and its variants in particular. * * DUK_DPRINT() allows formatted debug prints, and supports standard * and Duktape specific formatters. See duk_debug_vsnprintf.c for details. * * DUK_D(x), DUK_DD(x), and DUK_DDD(x) are used together with log macros * for technical reasons. They are concretely used to hide 'x' from the * compiler when the corresponding log level is disabled. This allows * clean builds on non-C99 compilers, at the cost of more verbose code. * Examples: * * DUK_D(DUK_DPRINT("foo")); * DUK_DD(DUK_DPRINT("foo")); * DUK_DDD(DUK_DPRINT("foo")); * * This approach is preferable to the old "double parentheses" hack because * double parentheses make the C99 solution worse: __FILE__ and __LINE__ can * no longer be added transparently without going through globals, which * works poorly with threading.
label: code-design

22137. prop index in 'array part', < 0 if not there

22138. caller handles sign change

22139. **comment:** load factor too low or high, count actually used entries and resize
label: code-design

22140. * Coercion and fast path processing.

22141. Not strictly necessary because if key == NULL, flag MUST be ignored.

22142. make the new thread reachable

22143. XXX: print built-ins array?

22144. temp reg

22145. * Externs and prototypes

22146. no incref

22147. Test signaling NaN and alias assignment in all endianness combinations.

22148. 1 GB

22149. XXX: fast path for array arguments?

22150. -> [... new_global new_globalenv new_global new_global]

22151. E5.1 Section 15.1.3.3: uriReserved + uriUnescaped + '#'

22152. DUK_USE_PACKED_TVAL

22153. **comment:** * Enumeration semantics come from for-in statement, E5 Section 12.6.4. * If called with 'null' or 'undefined', this opcode returns 'null' as * the enumerator, which is special cased in NEXTENUM. This simplifies * the compiler part
label: code-design

22154. Insert a reserved area somewhere in the buffer; caller fills it. * Evaluates to a (`duk_uint_t *`) pointing to the start of the reserved * area for convenience.

22155. values for the state field

22156. How large a loop detection stack to use

22157. pop enum

22158. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)

22159. <

22160. [... closure template undefined]

22161. **comment:** * To determine whether to use an optimized Ecmascript-to-Ecmascript * call, we need to know whether the final, non-bound function is an * Ecmascript function. * * This is now implemented so that we start to do an ecma-to-ecma call * setup which will resolve the bound chain as the first thing. If the * final function is not eligible, the return value indicates that the * ecma-to-ecma call is not possible. The setup will overwrite the call * target at DUK_REGP(idx) with the final, non-bound function (which * may be a lightfunc), and fudge arguments if necessary. * * XXX: If an ecma-to-ecma call is not possible, this initial call * setup will do bound function chain resolution but won't do the * "effective this binding" resolution which is quite confusing. * Perhaps add a helper for doing bound function and effective this * binding resolution - and call that explicitly? Ecma-to-ecma call * setup and normal function handling can then assume this prestep has * been done by the caller.
label: code-design

22162. dynamic buffer ops

22163. not found in 'curr', next in prototype chain; impose max depth

22164. 13: getUTCDate

22165. **comment:** Truncate to 16 bits here, so that a computed hash can be compared * against a hash stored in a 16-bit field.
label: code-design

22166. Buffer has virtual properties similar to string, but indexed values * are numbers, not 1-byte buffers/strings which would perform badly.
22167. [... this name]
22168. * Date/time parsing helper. * * Parse a datetime string into a time value. We must first try to parse * the input according to the standard format in E5.1 Section 15.9.1.15. * If that fails, we can try to parse using custom parsing, which can * either be platform neutral (custom code) or platform specific (using * existing platform API calls). * * Note in particular that we must parse whatever `toString()`, `toUTCString()`, * and `toISOString()` can produce; see E5.1 Section 15.9.4.2. * * Returns 1 to allow tail calling. * * There is much room for improvement here with respect to supporting * alternative datetime formats. For instance, V8 parses '2012-01-01' as * UTC and '2012/01/01' as local time.
22169. **comment:** XXX: check flags
label: requirement
22170. ASCII (and EOF) fast path -- quick accept and reject
22171. 0xc0...0xcf
22172. '-' as a range indicator
22173. -> [... enum_target res handler undefined target]
22174. [arg1 ... argN this loggerLevel loggerName 'fmt' arg]
22175. "123.456"
22176. Helper for creating the arguments object and adding it to the env record * on top of the value stack. This helper has a very strict dependency on * the shape of the input stack.
22177. If an implicit return value is needed by caller, it must be * initialized to 'undefined' because we don't know whether any * non-empty (where "empty" is a continuation type, and different * from an empty statement) statements will be executed. * * However, since 1st pass is a throwaway one, no need to emit * it here.
22178. 'false'
22179. Some internal code now assumes that all `duk_uint_t` values * can be expressed with a `duk_size_t`.
22180. data property or accessor without getter
22181. * Args validation
22182. DUK_TAG_NUMBER is intentionally first, as it is the default clause in code * to support the 8-byte representation. Further, it is a non-heap-allocated * type so it should come before DUK_TAG_STRING. Finally, it should not break * the tag value ranges covered by case-clauses in a switch-case.
22183. * String built-ins
22184. * Clear (reachable) flags of refzero work list.
22185. **comment:** * Number should already be in NaN-normalized form, but let's * normalize anyway.
label: code-design
22186. catchstack
22187. * C call recursion depth check, which provides a reasonable upper * bound on maximum C stack size (arbitrary C stack growth is only * possible by recursive `handle_call` / `handle_safe_call` calls).
22188. Call sites don't need the result length so it's not accumulated.
22189. non-zero: make copy
22190. Return non-zero (true) if we have a good reason to believe * the notify was delivered; if we're still attached at least * a transport error was not indicated by the transport write * callback. This is not a 100% guarantee of course.
22191. fromPresent = false
22192. Check that there's room to push one value.
22193. comma check
22194. backtrack to last output slash (dups already eliminated)
22195. **comment:** Fastint range is signed 48-bit so longest value is $-2^{47} = -140737488355328$ * (16 chars long), longest signed 64-bit value is $-2^{63} = -9223372036854775808$ * (20 chars long). Alloc space for 64-bit range to be safe.
label: code-design
22196. Flags for call handling.
22197. jump to end
22198. * RegExp built-ins
22199. allocation size
22200. no -> decref members, then free
22201. If any non-Array value had enumerable virtual own * properties, they should be serialized here. Standard * types don't.
22202. XXX: C recursion limit if proxies are allowed as handler/target values
22203. try part
22204. 'caller' must only take on 'null' or function value
22205. sign doesn't matter when writing
22206. **comment:** XXX: use macros for the repetitive tval/refcount handling.
label: code-design
22207. **comment:** Builtin-objects; may or may not be shared with other threads, * threads existing in different "compartments" will have different * built-ins. Must be stored on a per-thread basis because there * is no intermediate structure for a thread group / compartment. * This takes quite a lot of space, currently $43 \times 4 = 172$ bytes on * 32-bit platforms. * * In some cases the builtins array could be ROM based, but it's * sometimes edited (e.g. for sandboxing) so it's better to keep * this array in RAM.
label: code-design
22208. **comment:** unused
label: code-design
22209. * Other file level defines
22210. Fast path for numbers (one of which may be a fastint)
22211. maximum length for a SKIP-1 diffstream: 35 bits per entry, rounded up to bytes
22212. [enum_target enum res]
22213. orig value
22214. **comment:** XXX use get tval ptr, more efficient
label: code-design
22215. See comments in `duk_pcall()`.
22216. The 'left' value must not be a register bound variable * because it may be mutated during the rest of the expression * and E5.1 Section 11.2.1 specifies the order of evaluation * so that the base value is evaluated first. * See: `test-bug-nested-prop-mutate.js`.
22217. A bit tricky overflow test, see `doc/code-issues.rst`.
22218. key must not already exist in entry part
22219. Digit generation
22220. reset function state (prepare for pass 2)
22221. +0 = catch
22222. avoid side effects!
22223. **comment:** NOTE: "get" and "set" are not officially ReservedWords and the lexer * currently treats them always like ordinary identifiers (DUK_TOK_GET * and DUK_TOK_SET are unused). They need to be detected based on the * identifier string content.
label: code-design
22224. **comment:** XXX: regetting the pointer may be overkill - we're writing * to a side-effect free array here.
label: code-design
22225. * Because buffer values may be looped over and read/written * from, an array index fast path is important.
22226. `duk_small_uint_fast_t c = DUK_DEC_C(ins);`
22227. For all duk_hbufferobjects, get the plain buffer inside * without making a copy. This is compatible with Duktape 1.2 * but means that a slice/view information is ignored and the * full underlying buffer is returned. * * If called as a constructor, a new `Duktape.Buffer` object * pointing to the same plain buffer is created below.
22228. Copy values by index reads and writes. Let virtual * property handling take care of coercion.

22229. **comment:** No coercions or other side effects, so safe
 label: requirement

22230. 0x20-0x2f

22231. may become sparse...

22232. Pointers may be NULL for a while when 'buf' size is zero and before any * ENSURE calls have been made. Once an ENSURE has been made, the pointers * are required to be non-NULL so that it's always valid to use memcpy() and * memmove(), even for zero size.

22233. * Create and throw an Ecmascript error object based on a code and a message. * * Used when we throw errors internally. Ecmascript generated error objects * are created by Ecmascript code, and the throwing is handled by the bytecode * executor.

22234. XXXXXX--

22235. lightfuncs are treated like objects and not coerced

22236. jump to end or else part

22237. **comment:** Registers 'bc' and 'bc + 1' are written in longjmp handling * and if their previous values (which are temporaries) become * unreachable -and- have a finalizer, there'll be a function * call during error handling which is not supported now (GH-287). * Ensure that both 'bc' and 'bc + 1' have primitive values to * guarantee no finalizer calls in error handling. Scrubbing also * ensures finalizers for the previous values run here rather than * later. Error handling related values are also written to 'bc' * and 'bc + 1' but those values never become unreachable during * error handling, so there's no side effect problem even if the * error value has a finalizer.
 label: code-design

22238. no_block

22239. The original value needs to be preserved for filter(), hence * this funny order. We can't re-get the value because of side * effects.

22240. for duk_error_augment.c

22241. just one 'int' for C++ exceptions

22242. '~'

22243. reset voluntary gc trigger count

22244. If the separator is a RegExp, make a "clone" of it. The specification * algorithm calls [[Match]] directly for specific indices; we emulate this * by tweaking lastIndex and using a "force global" variant of duk_regexp_match() * which will use global-style matching even when the RegExp itself is non-global.

22245. * Defines for JSON, especially duk.bi.json.c.

22246. * Fast path tables

22247. -> [... handler trap]

22248. DUK_TOK_ALSHIFT_EQ

22249. Note: assumes 'data' is always a fixed buffer

22250. * Stringtable entry for fixed size stringtable

22251. sometimes stack and array indices need to go on the stack

22252. hash size ratio goal, must match genhashsizes.py

22253. * Heap structure. * * Heap contains allocated heap objects, interned strings, and built-in * strings for one or more threads.

22254. Require a lot of stack to force a value stack grow/shrink.

22255. **comment:** XXX: lastIndex handling produces a lot of asm
 label: code-design

22256. DUK_DEBUG_H_INCLUDED

22257. input size is good output starting point

22258. XXX: Current putvar implementation doesn't have a success flag, * add one and send to debug client?

22259. **comment:** must be able to emit code, alloc consts, etc.
 label: code-design

22260. **comment:** Should never happen but avoid infinite loop just in case.
 label: code-design

22261. unchanged from Duktape.Thread.yield()

22262. value1 -> return value, pseudo-type to indicate a return continuation (for ENDFIN)

22263. Negative offsets cause a RangeError.

22264. side effect free

22265. main expression parser function

22266. may happen if size is very close to 2^32-1

22267. fail: restore saves

22268. DUK_USE_DATE_PRS_GETDATE

22269. needed for stepping

22270. **comment:** XXX: could map/decode be unified with duk_unicode_support.c code? * Case conversion needs also the character surroundings though.
 label: code-design

22271. clen == blen -> pure ascii

22272. * The attempt to allocate may cause a GC. Such a GC must not attempt to resize * the stringtable (though it can be swept); finalizer execution and object * compaction must also be postponed to avoid the pressure to add strings to the * string table. Call site must prevent these.

22273. current setjmp() catchpoint

22274. Result is undefined.

22275. [... proplist]

22276. This may throw an error though not for valid E5 strings.

22277. * Marking functions for heap types: mark children recursively

22278. * Any catch bindings ("catch (e)") also affect identifier binding. * * Currently, the varmap is modified for the duration of the catch * clause to ensure any identifier accesses with the catch variable * name will use slow path.

22279. XXX: set with duk_hobject_set_length() when tracedata is filled directly

22280. **comment:** XXX: optimize by creating array into correct size directly, and * operating on the array part directly; values can be memcpy()'d from * value stack directly as long as refcounts are increased.
 label: code-design

22281. Slow path.

22282. 'compile'

22283. **comment:** XXX: does not work if thr->catchstack is allocated but lowest pointer
 label: code-design

22284. DUK_TOK_RCURLY

22285. **comment:** XXX: check .caller writability?
 label: code-design

22286. replacement handler

22287. 2**31-1 ~= 2G properties

22288. part of string

22289. Bytecode instructions: endian conversion needed unless * platform is big endian.

22290. DUK_TOK_EXTENDS

22291. XXX: optimize reconfig valstack operations so that resize, clamp, and setting * top are combined into one pass.

22292. fastint downgrade check for return values

22293. **comment:** bytes of indent still needed
 label: code-design

22294. DUK_JS_COMPILER_H_INCLUDED

22295. nop

together for larger parameter values. * Signed integers (e.g. jump offsets) are encoded as unsigned, with an opcode * specific bias. B and C may denote a register or a constant, see * DUK_BC_ISREG() and DUK_BC_ISCONST(). ** Note: macro naming is a bit misleading, e.g. "ABC" in macro name but * the field layout is logically "CBA".

label: code-design

22365. **comment:** There may be whitespace between the equal signs.

label: code-design

22366. **comment:** Note: first argument not really used

label: code-design

22367. * Function declarations

22368. reg_res should be smallest possible

22369. Slot B

22370. * Unreachable object, free

22371. Specific assumptions for opcode numbering.

22372. **comment:** copied so that 'ap' can be reused

label: code-design

22373. exponents 1023 to 1069

22374. * Allocate a heap. ** String table is initialized with built-in strings from genbuiltins.py, * either by dynamically creating the strings or by referring to ROM strings.

22375. (?)

22376. name

22377. Restart bytecode execution, possibly with a changed thread.

22378. -> [... func name]

22379. top of current frame (exclusive)

22380. if thisArg not supplied, behave as if undefined was supplied

22381. for side effects

22382. **comment:** XXX: match flag is awkward, rework

label: code-design

22383. fall through and continue for-loop

22384. * Module not loaded (and loading not started previously). ** Create a new require() function with 'id' set to resolved ID * of module being loaded. Also create 'exports' and 'module' * tables but don't register exports to the loaded table yet. * We don't want to do that unless the user module search callbacks * succeeds in finding the module.

22385. Careful here: state must be wiped before the call * so that we can cleanly handle a re-attach from * inside the callback.

22386. assume 0 <=> NULL

22387. skip leading digit

22388. B -> object register * C -> C+0 contains key, C+1 closure (value)

22389. caller guarantees

22390. **comment:** array part must not contain any non-unused properties, as they would * be configurable and writable.

label: code-design

22391. Note: '\b' in char class is different than outside (assertion), * '\B' is not allowed and is caught by the duk_unicode_is_identifier_part() * check below.

22392. Note: combining __FILE__, __LINE__, and __func__ into fmt would be * possible compile time, but waste some space with shared function names.

22393. at least this function exists

22394. hash part is a 'weak reference' and does not contribute

22395. 'pc'

22396. 'errThrow'

22397. DUK_TOK_SUPER

22398. This is guaranteed by debugger code.

22399. [... (sep) str1 str2 ... strN buf]

22400. 0: no change

22401. Bigin is 2^52. Used to detect normalized IEEE double mantissa values * which are at the lowest edge (next floating point value downwards has * a different exponent). The lowest mantissa has the form: * * 1000.....000 (52 zeroes; only "hidden bit" is set)

22402. reg(ignored)

22403. undefined coerces to zero which is correct

22404. overwrite any previous binding of the same name; the effect is * that last argument of a certain name wins.

22405. returned after EOF (infinite amount)

22406. **comment:** XXX: investigate whether write protect can be handled above, if we * just update length here while ignoring its protected status

label: code-design

22407. **comment:** This is a bit clunky because it is ANSI C portable. Should perhaps * relocate to another file because this is potentially platform * dependent.

label: code-design

22408. Unwind catchstack entries referring to the callstack entry we're reusing

22409. **comment:** XXX: many operations actually want toforcedtemp() -- brand new temp?

label: code-design

22410. e.g. DUK_OP_POSTINCR

22411. 'Uint8ClampedArray'

22412. No copy, leave zero bytes in the buffer. There's no * ambiguity with Float32/Float64 because zero bytes also * represent 0.0.

22413. no typedef

22414. [... func_template]

22415. Note: any entries above the callstack top are garbage and not zeroed. * Also topmost activation idx_retval is garbage (not zeroed), and must * be ignored.

22416. [arg1 ... argN obj length]

22417. Previous value of 'func' caller, restored when unwound. Only in use * when 'func' is non-strict.

22418. When doing string-to-number, lowest_mantissa is always 0 so * the exponent check, while incorrect, won't matter.

22419. If the value has a prototype loop, it's critical not to * throw here. Instead, assume the value is not to be * augmented.

22420. **comment:** unused with some debug level combinations

label: code-design

22421. Line number range for function. Needed during debugging to * determine active breakpoints.

22422. **comment:** slow init, do only for slow path cases

label: code-design

22423. DUK_BUFOBJ_DUKTAPE_BUFFER

22424. positive

22425. **comment:** XXX: Here we have a nasty dependency: the need to manipulate * the callstack means that catchstack must always be unwound by * the caller before unwinding the callstack. This should be fixed * later.

label: code-design

22426. e.g. DUK_OP_PREINCP

22427. DUK_JS_H_INCLUDED

22428. DUK_USE_DATE_NOW_WINDOWS

22429. !DUK_SINGLE_FILE

22430. this is added to checks to allow for Duktape * API calls in addition to function's own use

22431. [... key_obj key]

22432. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

22433. **is_setget**

22434. flags always encodes to 1 byte

22435. Ensure space for 1:1 output plus one escape.

22436. [... obj key value]

22437. * Byte-based matching would be possible for case-sensitive * matching but not for case-insensitive matching. So, we * match by decoding the input and bytecode character normally. ** Bytecode characters are assumed to be already canonicalized. * Input characters are canonicalized automatically by * duk_inp_get_cp() if necessary. ** There is no opcode for matching multiple characters. The * regexp compiler has trouble joining strings efficiently * during compilation. See doc/regexp.rst for more discussion.

22438. C array of duk_labelinfo

22439. **comment:** XXX: double evaluation for 'tv' argument.
label: code-design

22440. with stack depth (affects identifier lookups)

22441. [... Duktape.modSearch resolved_id last_comp fresh_require exports module]

22442. * Detect recursive invocation

22443. **comment:** known to be 32-bit
label: code-design

22444. * Directive prologue tracking.

22445. Handle lightfuncs through object coercion for now.

22446. exponent 0

22447. Identifier found in registers (always non-deletable) * or declarative environment record and non-configurable.

22448. duk_handle_ecma_call_setup: setup for a tail call

22449. match: keep wiped/resaved values

22450. E5 Section 11.7.2, steps 7 and 8

22451. **comment:** * Since there are no references in the catcher structure, * unwinding is quite simple. The only thing we need to * look out for is popping a possible lexical environment * established for an active catch clause.
label: code-design

22452. * String cache. ** Provides a cache to optimize indexed string lookups. The cache keeps * track of (byte offset, char offset) states for a fixed number of strings. * Otherwise we'd need to scan from either end of the string, as we store * strings in (extended) UTF-8.

22453. heap thread, used internally and for finalization

22454. Don't allow a constant for the object (even for a number etc), as * it goes into the 'A' field of the opcode.

22455. For now all calls except Ecma-to-Ecma calls prevent a yield.

22456. If tv1==tv2 this is a NOP, no check is needed

22457. Dispatch loop.

22458. DUK_TOK_SUB

22459. Shared Windows helpers.

22460. This is a pretty awkward control flow, but we need to recheck the * key coercion here.

22461. catches EOF and invalid digits

22462. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur. If we end up not making the * call we must restore the value.

22463. **comment:** XXX: better macro for DUK_TVAL_IS_UNDEFINED_OR_NULL(tv)
label: code-design

22464. * Call the constructor function (called in "constructor mode").

22465. minimum prime

22466. set by atom case clauses

22467. [... error func fileName]

22468. reset callstack limit

22469. right paren eaten

22470. T

22471. idx_desc

22472. E5 Section 11.7.1, steps 7 and 8

22473. coerced value is updated to value stack even when RangeError thrown

22474. Note: special DUK_EMIT_FLAG_B_IS_TARGETSOURCE * used to indicate that B is both a source and a * target register. When shuffled, it needs to be * both input and output shuffled.

22475. * Number is special because it doesn't have a specific * tag in the 8-byte representation.

22476. TypedArray views need an automatic ArrayBuffer which must be * provided as .buffer property of the view. Just create a new * ArrayBuffer sharing the same underlying buffer.

22477. Align 'p' to 4; the input data may have arbitrary alignment. * End of string check not needed because blen >= 16.

22478. Expression_opt

22479. Exact halfway, round to even.

22480. * Arguments check

22481. **comment:** Very minimal inlining to handle common idioms '!0' and '!1', * and also boolean arguments like '!false' and '!true'.
label: code-design

22482. name is empty -> return message

22483. * Rescue or free.

22484. grow by at most one

22485. [A B C D E F G H] rel_index = 2, del_count 3, item count 3 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)

22486. **comment:** XXX: helper; rely on Boolean.prototype as being non-writable, non-configurable
label: code-design

22487. Note: proxy handling must happen before key is string coerced.

22488. DUK_HBUFFEROBJECT_H_INCLUDED

22489. [... res]

22490. Note: unconditional throw

22491. DUK_ERRMSG_H_INCLUDED

22492. Neutered buffer, zero length seems * like good behavior here.

22493. no mark-and-sweep gc

22494. * DUK_BSWAP macros

22495. literals (E5 Section 7.8), except null, true, false, which are treated * like reserved words (above).

22496. loggerName

22497. will result in undefined

22498. DUK_TOK_QUESTION

22499. * Delayed initialization only occurs for 'NEWENV' functions.

22500. Allow automatic detection of octal base, overrides radix * argument and forces integer mode.

22501. Load constants onto value stack but don't yet copy to buffer.

22502. [hobject props enum(props) key desc]

22503. 'Float32Array'
22504. eat comma
22505. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT32
22506. Since stack indices are not reliable, we can't do anything useful * here. Invoke the existing setjmp catcher, or if it doesn't exist, * call the fatal error handler.
22507. [A B C D E F G H] rel_index = 2, del_count 3, item count 4 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F F G H] (actual result at this point)
22508. Commands and notifyfs initiated by Duktape.
22509. DUK_TOK_GT
22510. * Stringtable
22511. offset is now target of the pending split (right after jump)
22512. U+03A3 = GREEK CAPITAL LETTER SIGMA
22513. always 0 args
22514. exit bytecode executor by rethrowing an error to caller
22515. Note: e_next indicates the number of gc-reachable entries * in the entry part, and also indicates the index where the * next new property would be inserted. It does *not* indicate * the number of non-NULL keys present in the object. That * value could be counted separately but requires a pass through * the key list.
22516. initialize built-ins - either by copying or creating new ones
22517. Note: thr->catchstack_top may be 0, so that cat < thr->catchstack * initially. This is OK and intended.
22518. SCANBUILD: scan-build complains here about assigned value * being garbage or undefined. This is correct but operating * on undefined values has no ill effect and is ignored by the * caller in the case where this happens.
22519. IsAccessorDescriptor(desc) == true
22520. avoid issues with relocation
22521. avoid recursive string table call
22522. res.x1 -> res.x2
22523. -> [... x y fn]
22524. 'taint' result as complex -- this is conservative, * as lookaheads do not backtrack.
22525. LHS is already side effect free
22526. **comment:** XXX: hack, remove when const lookup is not O(n)
label: code-design
22527. Here the specification requires correct array index enumeration * which is a bit tricky for sparse arrays (it is handled by the * enum setup code). We now enumerate ancestors too, although the * specification is not very clear on whether that is required.
22528. object has an array part (a_size may still be 0)
22529. 0x20...0x2f
22530. eat dup slashes
22531. Get the current data pointer (caller must ensure buf != NULL) as a * duk_uint8_t ptr.
22532. initial estimate based on format string
22533. enable exotic behaviors last
22534. **comment:** if 'src < src_end_safe', safe to read 4 bytes
label: requirement
22535. **comment:** Straightforward algorithm, makes fewer compiler assumptions.
label: code-design
22536. [obj key desc]
22537. **comment:** Convenience for some situations; the above macros don't allow NULLs * for performance reasons.
label: code-design
22538. [... env callee varmap key val]
22539. currently required
22540. * Yield the current thread. * * The thread must be in yieldable state: it must have a resumer, and there * must not be any yield-preventing calls (native calls and constructor calls, * currently) in the thread's call stack (otherwise a resume would not be * possible later). This method must be called from an Ecmascript function. * * Args: * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.
22541. **comment:** No need to length check string: it will never exceed even * the 16-bit length maximum.
label: code-design
22542. (flags<<32) + (line), flags = 0
22543. insert range count
22544. XXX: Set 32-bit result (but must then handle signed and * unsigned results separately).
22545. [... new_glob new_env]
22546. Count actually used (non-NULL, non-DELETED) entries.
22547. no refcount check
22548. Parse statements until a closing token (EOF or '}') is found.
22549. duk_hthread_callstack_unwind() will decrease this on unwind
22550. Note: array dump will include elements beyond * 'length'.
22551. Note: currently the catch binding is handled without a register * binding because we don't support dynamic register bindings (they * must be fixed for an entire function). So, there is no need to * record regbases etc.
22552. Shared helper to provide toString() and valueOf(). Checks 'this', gets * the primitive value to stack top, and optionally coerces with ToString().
22553. size stored in duk_heapdr unused flag bits
22554. genbuiltins.py ensures
22555. -> [... val]
22556. CR LF again a special case
22557. Note: computes with 'idx' in assertions, so caller beware. * 'idx' is preincremented, i.e. '1' on first call, because it * is more convenient for the caller.
22558. Full, aligned 4-byte reads.
22559. stack[0] = start * stack[1] = end * stack[2] = ToObject(this) * stack[3] = ToUint32(length) * stack[4] = result array
22560. * Zero the struct, and start initializing roughly in order
22561. stack[0..nargs-1] = unshift args (vararg) * stack[nargs] = ToObject(this) * stack[nargs+1] = ToUint32(length)
22562. duk_unicode_idp_m_ids_noa[]
22563. mark-and-sweep is currently running
22564. **comment:** * Error codes: defined in duktape.h * * Error codes are used as a shorthand to throw exceptions from inside * the implementation. The appropriate Ecmascript object is constructed * based on the code. Ecmascript code throws objects directly. The error * codes are defined in the public API header because they are also used * by calling code.
label: code-design
22565. DUK_API_INTERNAL_H_INCLUDED
22566. '\xffTarget'
22567. FUNCTION EXPRESSIONS
22568. 0x50-0x5f
22569. Note: there is no standard formatter for function pointers
22570. default: no change
22571. internal type tag
22572. * Ecmascript bytecode executor.
22573. **comment:** 0 = DUK_HOBJECT_CLASS_UNUSED
label: code-design
22574. p points to digit part ('%oxy', p points to 'x')

22575. the compressed pointer is zeroed which maps to NULL, so nothing to do.
22576. 'count' is more or less comparable to normal trigger counter update * which happens in memory block (re)allocation.
22577. bottom of current frame
22578. Shared offset/length coercion helper.
22579. E5 Section 15.10.2.11
22580. 'writable'
22581. These functions have trouble working as lightfuncs. * Some of them have specific asserts and some may have * additional properties (e.g. 'require.id' may be written).
22582. Two lowest bits of opcode are used to distinguish * variants. Bit 0 = inc(0)/dec(1), bit 1 = pre(0)/post(1).
22583. internal spare
22584. value (error) is at stack top
22585. input byte length
22586. gap (if empty string, NULL)
22587. [args [crud] arguments]
22588. lightfunc
22589. higher value conserves memory; also note that linear scan is cache friendly
22590. **comment:** XXX: to be implemented?
 label: code-design
22591. **comment:** XXX: convert to fixed buffer?
 label: code-design
22592. **comment:** Debug macro to print all parts and dparts (used manually because of debug level).
 label: code-design
22593. Assume arrays are dense in the fast path.
22594. cannot happen: strings are not put into refzero list (they don't even have the next/prev pointers)
22595. * Value stack top handling
22596. Set property slot to an empty state. Careful not to invoke * any side effects while using desc.e_idx so that it doesn't * get invalidated by a finalizer mutating our object.
22597. x <= y --> not (x > y) --> not (y < x)
22598. [key setter this val]
22599. little endian
22600. Typing: use duk_small_(u)int_fast_t when decoding small * opcode fields (op, A, B, C) and duk_(u)int_fast_t when * decoding larger fields (e.g. BC which is 18 bits). Use * unsigned variant by default, signed when the value is used * in signed arithmetic. Using variable names such as 'a', 'b', * 'c', 'bc', etc makes it easier to spot typing mismatches.
22601. * Create required objects: * - 'arguments' object: array-like, but not an array * - 'map' object: internal object, tied to 'arguments' * - 'mappedNames' object: temporary value used during construction
22602. * Misc util stuff
22603. Contract, either: * - Push string to value stack and return 1 * - Don't push anything and return 0
22604. Node.js return value for failed writes is offset + #bytes * that would have been written.
22605. NaN sign bit is platform specific with unpacked, un-normalized NaNs
22606. **comment:** XXX: union would be more correct
 label: code-design
22607. * Allocate an duk_hobject. ** The allocated object has no allocation for properties; the caller may * want to force a resize if a desired size is known. ** The allocated object has zero reference count and is not reachable. * The caller MUST make the object reachable and increase its reference * count before invoking any operation that might require memory allocation.
22608. **comment:** XXX: Stack discipline is annoying, could be changed in numconv.
 label: code-design
22609. 'stack'
22610. * Heap compiled function (Ecmascript function) representation. ** There is a single data buffer containing the Ecmascript function's * bytecode, constants, and inner functions.
22611. Behavior for nargs < 2 is implementation dependent: currently we'll * set a NaN time value (matching V8 behavior) in this case.
22612. ".123"
22613. currently used -> new size
22614. caller is responsible for ensuring this
22615. must be signed
22616. uppercase
22617. **comment:** This variant is needed by String.prototype.split(); it needs to perform * global-style matching on a cloned RegExp which is potentially non-global.
 label: code-design
22618. **comment:** XXX: optimize this filling behavior later
 label: code-design
22619. replaced
22620. **comment:** * Binary bitwise operations use different coercions (ToInt32, ToUInt32) * depending on the operation. We coerce the arguments first using * ToInt32(), and then cast to an 32-bit value if necessary. Note that * such casts must be correct even if there is no native 32-bit type * (e.g., duk_int32_t and duk_uint32_t are 64-bit). ** E5 Sections 11.10, 11.7.1, 11.7.2, 11.7.3
 label: code-design
22621. [...] regexp input] -> [res_obj]
22622. Mostly API and built-in method related
22623. E5 Sections 15.9.3.1, B.2.4, B.2.5
22624. alloc and init
22625. prediction corrections for prime list (see genhashsizes.py)
22626. Make _Target and _Handler non-configurable and non-writable. * They can still be forcibly changed by C code (both user and * Duktape internal), but not by Ecmascript code.
22627. DUK_USE_REGEX_SUPPORT
22628. the DUK_TOK_RCURLY is eaten by duk_parse_stmts()
22629. * Interrupt counter fixup (for development only).
22630. Note: recursive call
22631. maximum bytecode length in instructions
22632. get const for value at valstack top
22633. value1 -> label number, pseudo-type to indicate a break continuation (for ENDFIN)
22634. line tracking
22635. first call
22636. * All done
22637. undefined is accepted
22638. * Wrap up
22639. this case can no longer occur because refcount is unsigned
22640. automatic semi will be inserted
22641. Special shuffling for INITGET/INITSET, where slot C * identifies a register pair and cannot be shuffled * normally. Use an indirect variant instead.
22642. reachable through activation
22643. [...] filename &comp_stk]

22644. **comment:** XXX: faster initialization (direct access or better primitives)
label: code-design

22645. Loop structure ensures that we don't compute $t1^2$ unnecessarily * on the final round, as that might create a bignum exceeding the * current DUK_BL_MAX_PARTS limit.

22646. **comment:** * Check for invalid backreferences; note that it is NOT an error * to back-reference a capture group which has not yet been introduced * in the pattern (as in $\backslash 1(foo)/$; in fact, the backreference will * always match! It IS an error to back-reference a capture group * which will never be introduced in the pattern. Thus, we can check * for such references only after parsing is complete.
label: code-design

22647. + 1 for rounding

22648. note: any entries above the callstack top are garbage and not zeroed

22649. must be, since env was created when catcher was created

22650. side effects (currently none though)

22651. DUK_BUILTIN_PROTOS_H_INCLUDED

22652. * Find a matching label catcher or 'finally' catcher in * the same function. ** A label catcher must always exist and will match unless * a 'finally' captures the break/continue first. It is the * compiler's responsibility to ensure that labels are used * correctly.

22653. shadowed, ignore

22654. 'Uint16Array'

22655. -> [... enum_target res]

22656. * Round-up limit. ** For even values, divides evenly, e.g. 10 -> roundup_limit=5. ** For odd values, rounds up, e.g. 3 -> roundup_limit=2. * If radix is 3, 0/3 -> down, 1/3 -> down, 2/3 -> up.

22657. +1 = one retval

22658. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. ** XXX: this is an overkill for some paths, so optimize this later * (or maybe switch to a stack arguments model entirely).
label: code-design

22659. '%p' and NULL is portable, but special case it * anyway to get a standard NULL marker in logs.

22660. [... obj key val]

22661. **comment:** XXX: for Object.keys() we should check enumerability of key
label: code-design

22662. idx_value may be < 0 (no value), set and get may be NULL

22663. numeric value (character, count)

22664. Note: first character is checked against this. But because * IdentifierPart includes all IdentifierStart characters, and * the first character (if unescaped) has already been checked * in the if condition, this is OK.

22665. assume 'var' has been eaten

22666. empty separator can always match

22667. * Close environment record(s) if they exist. ** Only variable environments are closed. If lex_env != var_env, it * cannot currently contain any register bound declarations. ** Only environments created for a NEWENV function are closed. If an * environment is created for e.g. an eval call, it must not be closed.

22668. coerce to string

22669. * Not found in registers, proceed to the parent record. * Here we need to determine what the parent would be, * if 'env' was not NULL (i.e. same logic as when initializing * the record). ** Note that environment initialization is only deferred when * DUK_HOBJECT_HAS_NEWENV is set, and this only happens for: * - Function code * - Strict eval code * * We only need to check _Lexenv here; _Varenv exists only if it * differs from _Lexenv (and thus _Lexenv will also be present).

22670. no check

22671. **comment:** XXX: optimize to 16 bytes
label: code-design

22672. * Endian conversion

22673. nrets

22674. * Second (and possibly third) pass. ** Generate actual code. In most cases the need for shuffle * registers is detected during pass 1, but in some corner cases * we'll only detect it during pass 2 and a third pass is then * needed (see GH-115).

22675. embed: integer value

22676. 0=base, 1=esc, 2=class, 3=class+esc

22677. XXX: proper flags?

22678. mark-and-sweep flags automatically active (used for critical sections)

22679. heap string indices are autogenerated in duk_strings.h

22680. some assertions

22681. array writes autoincrement length

22682. **comment:** * Special behavior in E5.1. ** Note that even though parents == 0, the conflicting property * may be an inherited property (currently our global object's * prototype is Object.prototype). Step 5.e first operates on * the existing property (which is potentially in an ancestor) * and then defines a new property in the global object (and * never modifies the ancestor). ** Also note that this logic would become even more complicated * if the conflicting property might be a virtual one. Object * prototype has no virtual properties, though. ** XXX: this is now very awkward, rework.
label: code-design

22683. Avoid zero copy with an invalid pointer. If obj->p is NULL, * the 'new_a' pointer will be invalid which is not allowed even * when copy size is zero.

22684. reserved words: keywords

22685. The #ifdef clutter here handles the JX/JC enable/disable * combinations properly.

22686. [... key val]

22687. **comment:** * Check whether or not we have an error handler. ** We must be careful of not triggering an error when looking up the * property. For instance, if the property is a getter, we don't want * to call it, only plain values are allowed. The value, if it exists, * is not checked. If the value is not a function, a TypeError happens * when it is called and that error replaces the original one.
label: code-design

22688. **comment:** XXX: 'res' setup can be moved to function body level; in fact, two 'res' * intermediate values suffice for parsing of each function. Nesting is needed * for nested functions (which may occur inside expressions).
label: code-design

22689. **comment:** This seems to waste a lot of stack frame entries, but good compilers * will compute these as needed below. Some of these initial flags are * also modified in the code below, so they can't all be removed.
label: code-design

22690. [... env callee varmap]

22691. duk_push_(u)int() is guaranteed to support at least (un)signed 32-bit range

22692. [body formals source template]

22693. User callback did not return source code, so module loading * is finished: just update modLoaded with final module.exports * and we're done.

22694. **comment:** XXX: make active breakpoints actual copies instead of pointers?
label: code-design

22695. XXX: this bound function resolution also happens elsewhere, * move into a shared helper.

22696. **comment:** XXX: non-callable .toJSON() doesn't need to cause an abort * but does at the moment, probably not worth fixing.
label: code-design

22697. num args starting from idx_argbase

22698. caller must change active thread, and set thr->resumer to NULL

22699. check that the valstack has space for the final amount and any * intermediate space needed; this is unoptimal but should be safe

22700. unary plus

22701. * Shortcut for accessing global object properties

22702. * slice()
22703. may be changed
22704. step 3.e: replace 'Desc.[[Value]]'
22705. Captures which are undefined have NULL pointers and are returned * as 'undefined'. The same is done when saved[] pointers are insane * (this should, of course, never happen in practice).
22706. Only direct eval inherits strictness from calling code * (E5.1 Section 10.1.1).
22707. Get a (possibly canonicalized) input character from current sp. The input * itself is never modified, and captures always record non-canonicalized * characters even in case-insensitive matching.
22708. **comment:** * Update idx_retval of current activation. ** Although it might seem this is not necessary (bytecode executor * does this for Ecmascript-to-Ecmascript calls; other calls are * handled here), this turns out to be necessary for handling yield * and resume. For them, an Ecmascript-to-native call happens, and * the Ecmascript call's idx_retval must be set for things to work.
label: code-design
22709. [...] varmap]
22710. * Type checking
22711. * ===== Duktape authors * ===== Copyright * ===== Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. ** Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. ** The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): *** Sami Vaarala <sami.vaarala@iki.fi> ** Niki Dobrev ** Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang \u00f3z <lango.u-szeged@partner.samsung.com> ** Legimetus <legimetus.calc@gmail.com> * * Karl Skomski <karl@skomski.com> ** Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> ** Julien Hamaide (<https://github.com/crazyjul>) ** Sebastian G\u00f6ttfert <<https://github.com/jaseg>> * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): *** Greg Burns ** Anthony Rabine ** Carlos Costa ** Aur\u00e9lien Bouilland ** Preet Desai (Pris Matic) ** judofyr (<http://www.reddit.com/user/judofyr>) ** Jason Woofenden ** Micha\u0142 Przyby\u015b ** Anthony Howe ** Conrad Pankoff ** Jim Schimpf ** Rajaran Gaunker (<https://github.com/zimbabao>) ** Andreas \u00d6man ** Doug Sanden ** Josh Engebretson (<https://github.com/JoshEngebretson>) ** Remo Eichenberger (<https://github.com/remoe>) ** Mamod Mehyar (<https://github.com/mamod>) ** David Demelier (<https://github.com/markand>) ** Tim Caswell (<https://github.com/creationix>) ** Mitchell Blank Jr (<https://github.com/mitchblank>) ** <https://github.com/yushli> * Seo Sanghyeon (<https://github.com/sanxiyn>) ** Han ChoongWoo (<https://github.com/tunz>) ** Joshua Peek (<https://github.com/josh>) ** Bruce E. Pascoe (<https://github.com/fatcerberus>) ** <https://github.com/Kelledin> ** <https://github.com/sstruktrup> ** Michael Drake (<https://github.com/tlsa>) ** <https://github.com/chris-y> ** Laurent Zubiaur (<https://github.com/lzubiaur>) ** Ole Andr\u00e9 Vadla Ravn \u00e5s (<https://github.com/oleavr>) ** If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.
22712. Grow array part for a new highest array index.
22713. no need to mask idx portion
22714. remove hash entry (no decref)
22715. index
22716. exclusive endpoint
22717. Append bytes from a slice already in the buffer.
22718. this also increases refcount by one
22719. whole, won't clip
22720. [source template]
22721. Cannot overlap.
22722. DUK__L0() cannot be a digit, because the loop doesn't terminate if it is
22723. apparently no hint?
22724. **comment:** DUK_USE_REGEXP_CANON_WORKAROUND
label: code-design
22725. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array
22726. XXX: clumsy sign extend and masking of 16 topmost bits
22727. Note: no key on stack
22728. [...] retval]
22729. DUK_USE_JX
22730. num args
22731. end of _Formals
22732. no need to pop, nothing was pushed
22733. This would be pointless: we'd return NULL for both lightfuncs and * unexpected types.
22734. 2 when called by debugger
22735. * Top level wrappers
22736. DUK_INTERNAL_H_INCLUDED
22737. Report next pc/line to be executed.
22738. Non-object argument is simply int coerced, matches * V8 behavior (except for "null", which we coerce to * 0 but V8 TypeErrors).
22739. * Build function fixed size 'data' buffer, which contains bytecode, * constants, and inner function references. ** During the building phase 'data' is reachable but incomplete. * Only incref's occur during building (no refzero or GC happens), * so the building process is atomic.
22740. fixed offsets in valstack
22741. * Math built-ins
22742. **comment:** Temporary structure used to pass a stack allocated region through * duk_safe_call().
label: code-design
22743. check that there is space for at least one new entry
22744. 0xa0...0xaf
22745. eval left argument first
22746. -> [...] closure template]
22747. 1 << 52
22748. must match genhashsizes.py
22749. setter and/or getter may be NULL
22750. * Encoding helpers ** Some of the typing is bytecode based, e.g. slice sizes are unsigned 32-bit * even though the buffer operations will use duk_size_t.
22751. current thread
22752. **comment:** Called on a read/write error: NULL all callbacks except the detached * callback so that we never accidentally call them after a read/write * error has been indicated. This is especially important for the transport * I/O callbacks to fulfill guaranteed callback semantics.
label: code-design
22753. Emit 3 bytes and backtrack if there was padding. There's * always space for the whole 3 bytes so no check needed.
22754. Duktape.Thread.yield() should prevent
22755. encode \x00Bar as _Bar if no quotes are applied, this is for * readable internal keys.
22756. LICENSE.txt
22757. [...] varname val]
22758. string is 'eval' or 'arguments'
22759. * The default property attributes are correct for all * function valued properties of built-in objects now.
22760. Heap udata, used for allocator functions but also for other heap * level callbacks like pointer compression, etc.
22761. IdentifierStart production with ASCII excluded
22762. nothing to mark
22763. **comment:** XXX: function properties

label: code-design

22764. prepend regexp to valstack 0 index
22765. act_caller->func may be NULL in some finalization cases, * just treat like we don't know the caller.
22766. convenience
22767. Previous value doesn't need refcount changes because its ownership * is transferred to prev_caller.
22768. * Pop built-ins from stack: they are now INCREF'd and * reachable from the builtins[] array or indirectly * through builtins[].
22769. [obj key value desc]
22770. stmt has non-empty value
22771. -> [... holder name val val ToString(i)]
22772. stage 2: delete configurable entries above target length
22773. When LHS is not register bound, always go through a * temporary. No optimization for top level assignment.
22774. keep key reachable for GC etc; guaranteed not to fail
22775. generate mantissa with a single leading whole number digit
22776. * Complex case conversion helper which decodes a bit-packed conversion * control stream generated by unicode/extract_caseconv.py. The conversion * is very slow because it runs through the conversion data in a linear * fashion to save space (which is why ASCII characters have a special * fast path before arriving here). * * The particular bit counts etc have been determined experimentally to * be small but still sufficient, and must match the Python script *(src/extract_caseconv.py). * * The return value is the case converted codepoint or -1 if the conversion * results in multiple characters (this is useful for regexp Canonicalization * operation). If 'buf' is not NULL, the result codepoint(s) are also * appended to the hbuffer. * * Context and locale specific rules must be checked before consulting * this function.
22777. Stored in duk_heapdr unused flags.
22778. **comment:** * Augmenting errors at their creation site and their throw site. * * When errors are created, traceback data is added by built-in code * and a user error handler (if defined) can process or replace the * error. Similarly, when errors are thrown, a user error handler * (if defined) can process or replace the error. * * Augmentation and other processing at error creation time is nice * because an error is only created once, but it may be thrown and * rethrown multiple times. User error handler registered for processing * an error at its throw site must be careful to handle rethrowing in * a useful manner. * * Error augmentation may throw an internal error (e.g. alloc error). * * Ecmascript allows throwing any values, so all values cannot be * augmented. Currently, the built-in augmentation at error creation * only augments error values which are Error instances (= have the * built-in Error.prototype in their prototype chain) and are also * extensible. User error handlers have no limitations in this respect.
label: code-design
22779. **comment:** * Emit initializers in sets of maximum max_init_values. * Corner cases such as single value initializers do not have * special handling now. * * Elided elements must not be emitted as 'undefined' values, * because such values would be enumerable (which is incorrect). * Also note that trailing elisions must be reflected in the * length of the final array but cause no elements to be actually * inserted.
label: code-design
22780. * Initialize built-in objects. Current thread must have a valstack * and initialization errors may longjmp, so a setjmp() catch point * must exist.
22781. terminator
22782. DUK_TOK_EQUALSIGN
22783. * Comparison
22784. DUK_JSON_H_INCLUDED
22785. may have side effects
22786. increases key refcount
22787. on first round, skip
22788. **comment:** * Ecmascript specification algorithm and conversion helpers. * * These helpers encapsulate the primitive Ecmascript operation * semantics, and are used by the bytecode executor and the API * (among other places). Note that some primitives are only * implemented as part of the API and have no "internal" helper. * (This is the case when an internal helper would not really be * useful; e.g. the operation is rare, uses value stack heavily, * etc.) * * The operation arguments depend on what is required to implement * the operation: * * - If an operation is simple and stateless, and has no side * effects, it won't take an duk_hthread argument and its * arguments may be duk_tval pointers (which are safe as long * as no side effects take place). * * - If complex coercions are required (e.g. a "ToNumber" coercion) * or errors may be thrown, the operation takes an duk_hthread * argument. This also implies that the operation may have * arbitrary side effects, invalidating any duk_tval pointers. * * - For operations with potential side effects, arguments can be * taken in several ways: * * a) as duk_tval pointers, which makes sense if the "common case" * can be resolved without side effects (e.g. coercion); the * arguments are pushed to the valstack for coercion if * necessary * * b) as duk_tval values * * c) implicitly on value stack top * * d) as indices to the value stack * * Future work: * * - Argument styles may not be the most sensible in every case now. * * - In-place coercions might be useful for several operations, if * in-place coercion is OK for the bytecode executor and the API.
label: code-design
22789. Note: may be triggered even if minimal new_size would not reach the limit, * plan limit accordingly (taking DUK_VALSTACK_GROW_STEP into account).
22790. **comment:** A little tricky string approach to provide the flags string. * This depends on the specific flag values in duk_regex.h, * which needs to be asserted for. In practice this doesn't * produce more compact code than the easier approach in use.
label: code-design
22791. Note: intentionally use approximations to shave a few instructions: * a_used = old_used (accurate: old_used + 1) * a_size = arr_idx (accurate: arr_idx + 1)
22792. Slot C is used in a non-standard fashion (range of regs), * emitter code has special handling for it (must not set the * "no shuffle" flag).
22793. Note: if atom were to contain e.g. captures, we would need to * re-match the atom to get correct captures. Simply quantifiers * do not allow captures in their atom now, so this is not an issue.
22794. * Criteria for augmenting: * * - augmentation enabled in build (naturally) * - error value internal prototype chain contains the built-in * Error prototype object (i.e. 'val instanceof Error') * * - Additional criteria for built-in augmenting: * * - error value is an extensible object
22795. **comment:** XXX: solve into closed form (smaller code)
label: code-design
22796. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * If the final target function cannot be handled by an ecma-to-ecma * call, return to the caller with a return value indicating this case. * The bound chain is resolved and the caller can resume with a plain * function call.
22797. must be signed for loop structure
22798. is_extra
22799. **comment:** * Memory calls: relative to heap, GC interaction, but no error throwing. * * XXX: Currently a mark-and-sweep triggered by memory allocation will run * using the heap->heap_thread. This thread is also used for running * mark-and-sweep finalization; this is not ideal because it breaks the * isolation between multiple global environments. * * Notes: * * - DUK_FREE() is required to ignore NULL and any other possible return * value of a zero-sized alloc/realloc (same as ANSI C free()). * * - There is no DUK_REALLOC_ZEROED because we don't assume to know the * old size. Caller must zero the reallocated memory. * * - DUK_REALLOC_INDIRECT() must be used when a mark-and-sweep triggered * by an allocation failure might invalidate the original 'ptr', thus * causing a realloc retry to use an invalid pointer. Example: we're * reallocating the value stack and a finalizer resizes the same value * stack during mark-and-sweep. The indirect variant requests for the * current location of the pointer being reallocated using a callback * right before every realloc attempt; this circuitous approach is used * to avoid strict aliasing issues in a more straightforward indirect * pointer (void ***) approach. Note: the pointer in the storage * location is read but is NOT updated; the caller must do that.
label: code-design
22800. Here 'x' is a Unicode codepoint
22801. [ToUInt32(len) array ToUInt32(2len)] -> [ToUInt32(len) array]
22802. Force a resize so that DELETED entries are eliminated. * Another option would be duk_recheck_strtab_size_probe(); * but since that happens on every intern anyway, this whole * check can now be disabled.
22803. * comp_ctx->curr_func is now ready to be converted into an actual * function template.
22804. Potentially truncated, NUL not guaranteed in any case. * The (int_rc < 0) case should not occur in practice.
22805. If not found, resume existence check from Function.prototype. * We can just substitute the value in this case; nothing will * need the original base value (as would be the case with e.g. * setters/getters).

22806. "-123456\0"
22807. Get the original arguments. Note that obj_index may be a relative * index so the stack must have the same top when we use it.
22808. curr value
22809. These are for rate limiting Status notifications and transport peeking.
22810. * Some E5/E5.1 algorithms require that array indices are iterated * in a strictly ascending order. This is the case for e.g. * Array.prototype.forEach() and JSON.stringify() PropertyList * handling. * * To ensure this property for arrays with an array part (and * arbitrary objects too, since e.g. forEach() can be applied * to an array), the caller can request that we sort the keys * here.
22811. DUK_USE_LEXER_SLIDING_WINDOW
22812. DUK_TOK_NULL
22813. * Must be careful to catch errors related to value stack manipulation * and property lookup, not just the call itself.
22814. The file/line arguments are NULL and 0, they're ignored by DUK_ERROR_RAW() * when non-verbose errors are used.
22815. **comment:** 8-byte format could be: * 1111 1111 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [41 bits] * * However, this format would not have a zero bit following the * leading one bits and would not allow 0xFF to be used as an * "invalid utf-8" marker for internal keys. Further, 8-byte * encodings (up to 41 bit code points) are not currently needed.
label: code-design
22816. -> [... obj value]
22817. [key trap_result] -> []
22818. replace in stack
22819. (maybe) truncated
22820. nothing to process
22821. Encode string in small chunks, estimating the maximum expansion so that * there's no need to ensure space while processing the chunk.
22822. * Object.getOwnPropertyDescriptor() (E5 Sections 15.2.3.3, 8.10.4) * * This is an actual function call.
22823. knock back "next temp" to this whenever possible
22824. * Detected a directive
22825. 'for'
22826. complete the millisecond field
22827. 'It is true'
22828. Pre/post inc/dec for register variables, important for loops.
22829. **comment:** Lightfunc name, includes Duktape/C native function pointer, which * can often be used to locate the function from a symbol table. * The name also includes the 16-bit duk_tval flags field because it * includes the magic value. Because a single native function often * provides different functionality depending on the magic value, it * seems reasonably to include it in the name. * * On the other hand, a complicated name increases string table * pressure in low memory environments (but only when function name * is accessed).
label: code-design
22830. **comment:** XXX: if duk_hobject_define_property_internal() was updated * to handle a pre-existing accessor property, this would be * a simple call (like for the ancestor case).
label: code-design
22831. Iteration solution
22832. **comment:** 16 bits would be enough for shared heaphdr flags and duk_hstring * flags. The initial parts of duk_heaphdr_string and duk_heaphdr * must match so changing the flags field size here would be quite * awkward. However, to minimize struct size, we can pack at least * 16 bits of duk_hstring data into the flags field.
label: code-design
22833. * Helper tables
22834. 'this'
22835. **comment:** XXX: could check for e16 == 0 because NULL is guaranteed to * encode to zero.
label: code-design
22836. approximate
22837. Avoid NaN-to-integer coercion as it is compiler specific.
22838. Reset to zero size, keep current limit.
22839. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.
22840. callback for indirect reallocs, request for current pointer
22841. [...]]
22842. * Portable 12-byte representation
22843. avoid multiple computations of flags address; bypasses macros
22844. **comment:** Reg/const access macros: these are very footprint and performance sensitive * so modify with care.
label: code-design
22845. must not truncate
22846. prefix matches, lengths matter now
22847. Careful with reachability here: don't pop 'obj' before pushing * proxy target.
22848. * Replace valstack top with case converted version.
22849. Shuffle decision not changed.
22850. * Stack index validation/normalization and getting a stack duk_tval ptr. * * These are called by many API entrypoints so the implementations must be * fast and "inlined". * * There's some repetition because of this; keep the functions in sync.
22851. numeric value of token
22852. [...] constructor arg1 ... argN] -> [retval]
22853. * Rebuild the hash part always from scratch (guaranteed to finish). * * Any resize of hash part requires rehashing. In addition, by rehashing * get rid of any elements marked deleted (DUK_HASH_DELETED) which is critical * to ensuring the hash part never fills up.
22854. temps
22855. 'Object'
22856. not strictly necessary
22857. IsDataDescriptor(desc) == true
22858. POSTFIX EXPRESSION
22859. If argument count is 1 and first argument is a buffer, write the buffer * as raw data into the file without a newline; this allows exact control * over stdout/stderr without an additional entrypoint (useful for now). * * Otherwise current print/alert semantics are to ToString() coerce * arguments, join them with a single space, and append a newline.
22860. * Exposed regexp compilation primitive. * * Sets up a regexp compilation context, and calls duk_parse_disjunction() to do the * actual parsing. Handles generation of the compiled regexp header and the * "boilerplate" capture of the matching substring (save 0 and 1). Also does some * global level regexp checks after recursive compilation has finished. * * An escaped version of the regexp source, suitable for use as a RegExp instance * 'source' property (see E5 Section 15.10.3), is also left on the stack. * * Input stack: [pattern flags] * Output stack: [bytecode escaped_source] (both as strings)
22861. 'Error'
22862. * Flags for __FILE__ / __LINE__ registered into tracedata
22863. Count actually used entry part entries (non-NULL keys).
22864. DUK_TOK_ARSHIFT
22865. **comment:** XXX: possibly incorrect handling of empty expression
label: code-design
22866. used entries + approx 100% -> reset load to 50%
22867. '\xffRegbase'
22868. is_decl
22869. but don't allow leading plus

22870. **comment:** * String cache should ideally be at duk_hthread level, but that would * cause string finalization to slow down relative to the number of * threads; string finalization must check the string cache for "weak" * references to the string being finalized to avoid dead pointers. ** Thus, string caches are now at the heap level now.
label: code-design

22871. For manual debugging: instruction count based on executor and * interrupt counter book-keeping. Inspect debug logs to see how * they match up.

22872. [...] holder name val enum obj_key val obj_key]

22873. lj.value1 is already set

22874. number of arguments allocated to regs

22875. Optimized for size.

22876. * Default fatal error handler

22877. named "arithmetic" because result is signed

22878. Default implicit return value.

22879. lookup name from current activation record's functions' registers

22880. custom; implies DUK_HOBJECT_IS_BUFFEROBJECT

22881. max # of key-value pairs initialized in one MPUTOBJ set

22882. unicode code points, window[0] is always next

22883. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.

22884. thread not currently running

22885. Refzero head is still the same. This is the case even if finalizer * inserted more refzero objects; they are inserted to the tail.

22886. [... key arg1 ... argN]

22887. * GETIDREF: a GetIdentifierReference-like helper. ** Provides a parent traversing lookup and a single level lookup * (for HasBinding). ** Instead of returning the value, returns a bunch of values allowing * the caller to read, write, or delete the binding. Value pointers * are duk_tval pointers which can be mutated directly as long as * refcounts are properly updated. Note that any operation which may * reallocate valstacks or compact objects may invalidate the returned * duk_tval (but not object) pointers, so caller must be very careful. ** If starting environment record 'env' is given, 'act' is ignored. * However, if 'env' is NULL, the caller may identify, in 'act', an * activation which hasn't had its declarative environment initialized * yet. The activation registers are then looked up, and its parent * traversed normally. * * The 'out' structure values are only valid if the function returns * success (non-zero).

22888. Ignore the normalize/validate helper outputs on the value stack, * they're popped automatically.

22889. **comment:** A union is used here as a portable struct size / alignment trick: * by adding a 32-bit or a 64-bit (unused) union member, the size of * the struct is effectively forced to be a multiple of 4 or 8 bytes * (respectively) without increasing the size of the struct unless * necessary.
label: code-design

22890. Chop extra retvals away / extend with undefined.

22891. MPUTARR emitted by outer loop

22892. **comment:** XXX: fastint?
label: code-design

22893. compact the exports table

22894. Code is not accepted before the first case/default clause

22895. Process one debug message. Automatically restore value stack top to its * entry value, so that individual message handlers don't need exact value * stack handling which is convenient.

22896. -> [...] key val replacer holder key val]

22897. Because new_size != 0, if condition doesn't need to be * (new_valstack != NULL || new_size == 0).

22898. No debugger support.

22899. '<'

22900. treat like property not found

22901. Bound functions don't have all properties so we'd either need to * lookup the non-bound target function or reject bound functions. * For now, bound functions are rejected.

22902. Carry is detected based on wrapping which relies on exact 32-bit * types.

22903. eat (first) input slash

22904. value resides in a register or constant

22905. **comment:** "get" and "set" are tokens but NOT ReservedWords. They are currently * parsed and identifiers and these defines are actually now unused.
label: code-design

22906. Overflow, relevant mainly when listlen is 16 bits.

22907. [...] val] --> [...] enum]

22908. E5.1 standard behavior when deleteCount is not given would be * to treat it just like if 'undefined' was given, which coerces * ultimately to 0. Real world behavior is to splice to the end * of array, see test-bi-array-proto-splice-no-delcount.js.

22909. case 0: nop

22910. **comment:** XXX: the duk_hobject_enum.c stack APIs should be reworked
label: code-design

22911. Error: try coercing error to string once.

22912. for resizing of array part, use slow path

22913. y == -Infinity

22914. Initialization and finalization (compaction), converting to other types.

22915. must match exactly the number of internal properties inserted to enumerator

22916. required

22917. + spare

22918. All element accessors are host endian now (driven by TypedArray spec).

22919. Push an arbitrary duk_tval to the stack, coerce it to string, and return * both a duk_hstring pointer and an array index (or DUK__NO_ARRAY_INDEX).

22920. must "goto restart_execution", e.g. breakpoints changed

22921. Avoid side effects that might disturb curr.e_idx until * we're done editing the slot.

22922. 23: getUTCMilliseconds

22923. no need to normalize

22924. * Windows Date providers ** Platform specific links: ** - [http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473(v=vs.85).aspx)

22925. **comment:** XXX: Change 'anon' handling here too, to use empty string for anonymous functions?
label: code-design

22926. 'this binding' is just under bottom

22927. Default exports table

22928. **comment:** XXX: macro smaller than call?
label: code-design

22929. XXX: assert? compiler is responsible for this never happening

22930. pc of label statement: * pc+1: break jump site * pc+2: continue jump site

22931. 'require'

22932. * Assertion helpers

22933. multiple codepoint conversion or non-ASCII mapped to ASCII * --> leave as is.

22934. Computation must not wrap, only srclen + 3 is at risk of * wrapping because after that the number gets smaller. * This limit works for 32-bit size_t: * 0x100000000 - 3 - 1 = 4294967292

22935. 19: getUTCMinutes

22936. Inspect a property using a virtual index into a conceptual property list * consisting of (1) all array part items from [0,a_size[(even when above * .length) and (2) all entry part items from [0,e_next[. Unused slots are * indicated using dvalue 'unused'.

22937. duk__get_min_grow_e() is always >= 1

22938. * Reachable object, keep
22939. 'eval'
22940. Array part may be larger than 'length'; if so, iterate * only up to array 'length'.
22941. reg_catch+0 and reg_catch+1 are reserved for TRYCATCH
22942. resolved id: require(id) must return this same module
22943. We rely on a few object flag / class number relationships here, * assert for them.
22944. default case control flow patchup; note that if pc_prevcase < 0 * (i.e. no case clauses), control enters default case automatically.
22945. DUK_HEAP_H_INCLUDED
22946. note: original, uncoerced base
22947. If len <= 1, middle will be 0 and for-loop bails out * immediately (0 < 0 -> false).
22948. DUK_OP_NONE marks a 'plain' assignment
22949. resume execution from pc_base or pc_base+1 (points to 'func' bytecode, stable pointer)
22950. Stack size decreases.
22951. heap->strs: not worth dumping
22952. -> [... closure template env funcname closure]
22953. step 12
22954. Section 7.8.3 (note): NumericLiteral must be followed by something other than * IdentifierStart or DecimalDigit.
22955. **comment:** XXX: duk_ssize_t
 label: code-design
22956. DUK_ERR_UNCAUGHT_ERROR: no macros needed
22957. -> [ToObject(this) item1 ... itemN arr]
22958. C call site gets blamed next, unless flagged not to do so. * XXX: file/line is disabled in minimal builds, so disable this * too when appropriate.
22959. * Debug dump type sizes
22960. needed for line number tracking (becomes prev_token.start_line)
22961. don't coerce yet to a plain value (variant 3 needs special handling)
22962. **comment:** XXX: opcode specific assertions on when consts are allowed
 label: code-design
22963. Read value pushed on stack.
22964. * When resizing the valstack, a mark-and-sweep may be triggered for * the allocation of the new valstack. If the mark-and-sweep needs * to use our thread for something, it may cause *the same valstack* to be resized recursively. This happens e.g. when mark-and-sweep * finalizers are called. This is taken into account carefully in duk_resize_valstack(). * * 'new_size' is known to be <= valstack_max, which ensures that * size_t and pointer arithmetic won't wrap in duk_resize_valstack().
22965. 'null' enumerator case -> behave as with an empty enumerator
22966. Object literal set/get functions have a name (property * name) but must not have a lexical name binding, see * test-bug-getset-func-name.js.
22967. Current compiler state (if any), used for augmenting SyntaxErrors.
22968. push to stack
22969. Note: if DecimalLiteral starts with a '0', it can only be * followed by a period or an exponent indicator which starts * with 'e' or 'E'. Hence the if-check above ensures that * OctalIntegerLiteral is the only valid NumericLiteral * alternative at this point (even if y is, say, '9').
22970. 'catch'
22971. * Special cases like NaN and +/- Infinity are handled explicitly * because a plain C coercion from double to int handles these cases * in undesirable ways. For instance, NaN may coerce to INT_MIN * (not zero), and INT_MAX + 1 may coerce to INT_MIN (not INT_MAX). * * This double-to-int coercion differs from ToInteger() because it * has a finite range (ToInteger() allows e.g. +/- Infinity). It * also differs fromToInt32() because the INT_MIN/INT_MAX clamping * depends on the size of the int type on the platform. In particular, * on platforms with a 64-bit int type, the full range is allowed.
22972. element size shift: * 0 = u8/i8 * 1 = u16/i16 * 2 = u32/i32/float * 3 = double
22973. curr_pc synced back above
22974. hash size approx. 1.25 times entries size
22975. Write in little endian
22976. source starts after dest ends
22977. **comment:** * Debugging macro calls.
 label: code-design
22978. DUK_USE_VERBOSE_ERRORS
22979. 0xxx xxxx -> fast path
22980. * Ecmascript modulus (%) does not match IEEE 754 "remainder" * operation (implemented by remainder() in C99) but does seem * to match ANSI C fmod(). * * Compare E5 Section 11.5.3 and "man fmod".
22981. unsigned shift
22982. Not a very good provider: only full seconds are available.
22983. 110x xxxx 10xx xxxx [11 bits]
22984. Note: errors are augmented when they are created, not * when they are thrown. So, don't augment here, it would * break re-throwing for instance.
22985. **comment:** Not necessary to unwind catchstack: return handling will * do it. The finally flag of 'cat' is no longer set. The * catch flag may be set, but it's not checked by return handling.
 label: code-design
22986. ensure value is 1 or 0 (not other non-zero)
22987. Accept plain buffer values like array initializers * (new in Duktape 1.4.0).
22988. Non-zero refcounts should not happen for unreachable strings, * because weRefCount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
22989. Indicate function type in the function body using a dummy * directive.
22990. 'ObjEnv'
22991. lj.value1 already set
22992. millisec -> 100ns units since jan 1, 1970
22993. **comment:** * Note: must use indirect variant of DUK_REALLOC() because underlying * pointer may be changed by mark-and-sweep.
 label: code-design
22994. DUK_TOK_SWITCH
22995. class Object, extensible
22996. idx_end
22997. _Source
22998. 1 1 0 <8 bits>
22999. **comment:** two casts to avoid gcc warning: "warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]"
 label: code-design
23000. minimum new length is highest_arr_idx + 1
23001. DUK_TOK_RBRACKET
23002. property is configurable and
23003. * Matching complete, create result array or return a 'null'. Update lastIndex * if necessary. See E5 Section 15.10.6.2. * * Because lastIndex is a character (not byte) offset, we need the character * length of the match which we conveniently get as a side effect of interning * the matching substring (0th index of result array). * * saved[0] start pointer (~ byte offset) of current match * saved[1] end pointer (~ byte offset) of current match (exclusive) * char_offset start character offset of current match (-> .index of result) * char_end_offset end character offset (computed below)
23004. **comment:** XXX: Refactor key coercion so that it's only called once. It can't * be trivially lifted here because the object must be type checked * first.
 label: code-design
23005. Caller must ensure 'tv' is indeed a double and not a fastint!

23006. Caller has already eaten the first character ('|') which we don't need.
23007. * String scanning helpers * * All bytes other than UTF-8 continuation bytes ([0x80,0xbff]) are * considered to contribute a character. This must match how string * character length is computed.
23008. [... func this | arg1 ... argN] ('this' must precede new bottom)
23009. structs from duk_forwdecl.h
23010. assumed to also be PC of "LABEL"
23011. Check that prev/next links are consistent: if e.g. h->prev is != NULL, * h->prev->next should point back to h.
23012. {'_undef':true}
23013. DUK_TOK_ADD_EQ
23014. roughly 1.0 kB -> but rounds up to DUK_VALSTACK_GROW_STEP in practice
23015. remove the original 'template' atom
23016. The ANSI C pow() semantics differ from Ecmascript. * * E.g. when x==1 and y is +/- infinite, the Ecmascript required * result is NaN, while at least Linux pow() returns 1.
23017. * Heap native function representation.
23018. Note: DUK_HEAP_HAS_REFZERO_FREE_RUNNING(heap) may be true; a refcount * finalizer may trigger a mark-and-sweep.
23019. 'Invalid Date'
23020. dynamic
23021. **comment:** * Node.js Buffer.concat()
 label: code-design
23022. **comment:** XXX: substring in-place at idx_place?
 label: code-design
23023. * Object's finalizer was executed on last round, and * object has been happily rescued.
23024. embed: 0 or 1 (false or true)
23025. Labels can be used for iteration statements but also for other statements, * in particular a label can be used for a block statement. All cases of a * named label accept a 'break' so that flag is set here. Iteration statements * also allow 'continue', so that flag is updated when we figure out the * statement type.
23026. string 2 of token (borrowed, stored to ctx->slot2_idx)
23027. UNARY EXPRESSIONS
23028. * Not found (even in global object). * * In non-strict mode this is a silent SUCCESS (!), see E5 Section 11.4.1, * step 3.b. In strict mode this case is a compile time SyntaxError so * we should not come here.
23029. should actually never happen, but check anyway
23030. input offset for window leading edge (not window[0])
23031. * Init object properties * * Properties should be added in decreasing order of access frequency. * (Not very critical for function templates.)
23032. [... fallback constructor fallback(this) arg1 ... argN]; * Note: idx_cons points to first 'fallback', not 'constructor'.
23033. **comment:** * Canonicalize a range, generating result ranges as necessary. * Needs to exhaustively scan the entire range (at most 65536 * code points). If 'direct' is set, caller (lexer) has ensured * that the range is already canonicalization compatible (this * is used to avoid unnecessary canonicalization of built-in * ranges like \W, which are not affected by canonicalization). * * NOTE: here is one place where we don't want to support chars * outside the BMP, because the exhaustive search would be * massively larger.
 label: code-design
23034. continuation byte
23035. Note: the realloc may have triggered a mark-and-sweep which may * have resized our valstack internally. However, the mark-and-sweep * MUST NOT leave the stack bottom/top in a different state. Particular * assumptions and facts: * * - The thr->valstack pointer may be different after realloc, * and the offset between thr->valstack_end <-> thr->valstack * may have changed. * - The offset between thr->valstack_bottom <-> thr->valstack * and thr->valstack_top <-> thr->valstack MUST NOT have changed, * because mark-and-sweep must adhere to a strict stack policy. * In other words, logical bottom and top MUST NOT have changed. * - All values above the top are unreachable but are initialized * to UNDEFINED, up to the post-realloc valstack_end. * - 'old_end_offset' must be computed after realloc to be correct.
23036. Note: hstring is in heap but has refcount zero and is not strongly reachable. * Caller should increase refcount and make the hstring reachable before any * operations which require allocation (and possible gc).
23037. **comment:** * "name" is a non-standard property found in at least V8, Rhino, smjs. * For Rhino and smjs it is non-writable, non-enumerable, and non-configurable; * for V8 it is writable, non-enumerable, non-configurable. It is also defined * for an anonymous function expression in which case the value is an empty string. * We could also leave name 'undefined' for anonymous functions but that would * differ from behavior of other engines, so use an empty string. * * XXX: make optional? costs something per function.
 label: code-design
23038. **comment:** * duk_hbuffer operations such as resizing and inserting/appending data to * a dynamic buffer. * * Append operations append to the end of the buffer and they are relatively * efficient: the buffer is grown with a "spare" part relative to the buffer * size to minimize reallocations. Insert operations need to move existing * data forward in the buffer with memmove() and are not very efficient. * They are used e.g. by the regexp compiler to "backpatch" regexp bytecode.
 label: code-design
23039. steps 11.c.ii.1 - 11.c.ii.4, but our internal book-keeping * differs from the reference model
23040. ASCII fast path
23041. Most input bytes go through here.
23042. -> [... holder name val]
23043. Assume that year, month, day are all coerced to whole numbers. * They may also be NaN or infinity, in which case this function * must return NaN or infinity to ensure time value becomes NaN. * If 'day' is NaN, the final return will end up returning a NaN, * so it doesn't need to be checked here.
23044. useful for debugging
23045. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()! * * A conservative approach would be to use duk__ivalue_totempconst() * for 'left'. However, allowing a reg-bound variable seems safe here * and is nice because "foo.bar" is a common expression. If the ivalue * is used in an expression a GETPROP will occur before any changes to * the base value can occur. If the ivalue is used as an assignment * RHS, the assignment code will ensure the base value is safe from * RHS mutation.
23046. We should never exit the loop above.
23047. may have FINALIZED
23048. LAYOUT 3
23049. **comment:** unused
 label: code-design
23050. these computations are guaranteed to be exact for the valid * E5 time value range, assuming milliseconds without fractions.
23051. 'Int32Array'
23052. an error handler (user callback to augment/replace error) is running
23053. without refcounts
23054. arbitrary marker, outside valid exp range
23055. extra -1 for buffer
23056. For invalid characters the value -1 gets extended to * at least 16 bits. If either nybble is invalid, the * resulting 't' will be < 0.
23057. required to guarantee success of rehashing, * intentionally use unadjusted new_e_size
23058. XXX: A fast path for usual integers would be useful when * fastint support is not enabled.
23059. -> [... ToObject(this) ToUInt32(length) final_len final_len]
23060. [...val] -> [...]
23061. allow empty expression
23062. [... closure template name]
23063. # total registers target function wants on entry (< 0 => "as is")
23064. **comment:** unnecessary shift for last byte
 label: code-design

23065. varenv remains reachable through 'obj'
23066. Special coercion for Uint8ClampedArray.
23067. valstack slot for temp buffer
23068. * Node.js Buffer: constructor
23069. this optimization is important to handle negative literals * (which are not directly provided by the lexical grammar)
23070. Nothing worked -> not equal.
23071. E5 Section 11.13.1 (and others for other assignments), step 4.
23072. Note: unshift() may operate on indices above unsigned 32-bit range * and the final length may be $\geq 2^{**32}$. However, we restrict the * final result to 32-bit range for practicality.
23073. String and buffer enumeration behavior is identical now, * so use shared handler.
23074. $37 \times 32 = 1184$ bits
23075. heap->dbg_detached_cb: keep
23076. This check used to be for $(t < 0)$ but on some platforms * time_t is unsigned and apparently the proper way to detect * an mktime() error return is the cast above. See e.g.: * <http://pubs.opengroup.org/onlinepubs/009695299/functions/mktime.html>
23077. **comment:** Fast variants, inline refcount operations except for refzero handling. * Can be used explicitly when speed is always more important than size. * For a good compiler and a single file build, these are basically the * same as a forced inline.
label: code-design
23078. [... key] -> [...]
23079. * Use current token to decide whether a RegExp can follow. * * We can use either 't' or 't_nores'; the latter would not * recognize keywords. Some keywords can be followed by a * RegExp (e.g. "return"), so using 't' is better. This is * not trivial, see doc/compiler.rst.
23080. Khronos/ES6 requires zeroing even when DUK_USE_ZERO_BUFFER_DATA * is not set.
23081. * Refcount memory freeing loop. * * Frees objects in the refzero_pending list until the list becomes * empty. When an object is freed, its references get decref'd and * may cause further objects to be queued for freeing. * * This could be expanded to allow incremental freeing: just bail out * early and resume at a future alloc/decref/refzero.
23082. Note: 'fast' breaks will jump to pc_label_site + 1, which will * then jump here. The double jump will be eliminated by a * peephole pass, resulting in an optimal jump here. The label * site jumps will remain in bytecode and will waste code size.
23083. **comment:** * Prototype walking starting from 'env'. * * ('act' is not needed anywhere here.)
label: code-design
23084. mark range 'direct', bypass canonicalization (see Wiki)
23085. Do debugger forwarding before raw() because the raw() function * doesn't get the log level right now.
23086. estimate year upwards (towards positive infinity), then back down; * two iterations should be enough
23087. return module.exports
23088. * Parsing an expression starting with 'new' is tricky because * there are multiple possible productions deriving from * LeftHandSideExpression which begin with 'new'. * * We currently resort to one-token lookahead to distinguish the * cases. Hopefully this is correct. The binding power must be * such that parsing ends at an LPAREN (CallExpression) but not at * a PERIOD or LBRACKET (MemberExpression). * * See doc/compiler.rst for discussion on the parsing approach, * and testcases/test-dev-new.js for a bunch of documented tests.
23089. exptop is the top level variant which resets nud/led counts
23090. control props
23091. push value to stack
23092. * shift()
23093. k+argCount-1; note that may be above 32-bit range
23094. * Assert macro: failure causes panic.
23095. **comment:** Similar workaround for INFINITY.
label: code-design
23096. * Fallback marking handler if recursion limit is reached. * * Iterates 'temproots' until recursion limit is no longer hit. Note * that temproots may reside either in heap allocated list or the * refzero work list. This is a slow scan, but guarantees that we * finish with a bounded C stack. * * Note that nodes may have been marked as temproots before this * scan begun, OR they may have been marked during the scan (as * we process nodes recursively also during the scan). This is * intended behavior.
23097. **comment:** If number would need zero padding (for whole number part), use * exponential format instead. E.g. if input number is 12300, 3 * digits are generated ("123"), output "1.23e+4" instead of "12300". * Used for toPrecision().
label: code-design
23098. number
23099. Must be element size multiple from * start offset to end of buffer.
23100. empty ("") is allowed in some formats (e.g. Number(")), as zero
23101. slow path is shared
23102. * If matching with a regexp: * - non-global RegExp: lastIndex not touched on a match, zeroed * on a non-match * - global RegExp: on match, lastIndex will be updated by regexp * executor to point to next char after the matching part (so that * characters in the matching part are not matched again) * * If matching with a string: * - always non-global match, find first occurrence * * We need: * - The character offset of start-of-match for the replacer function * - The byte offsets for start-of-match and end-of-match to implement * the replacement values \$\$, \$` and \$', and to copy non-matching * input string portions (including header and trailer) verbatim. * * NOTE: the E5.1 specification is a bit vague how the RegExp should * behave in the replacement process; e.g. is matching done first for * all matches (in the global RegExp case) before any replacer calls * are made? See: test-bi-string-proto-replace.js for discussion.
23103. -> [... func]
23104. * Various defines and file specific helper macros
23105. no net exponent
23106. **comment:** integer, but may be +/- Infinite, +/- zero (not NaN, though)
label: code-design
23107. 11 bits
23108. **comment:** Lookup active label information. Break/continue distinction is necessary to handle switch * statement related labels correctly: a switch will only catch a 'break', not a 'continue'. * * An explicit label cannot appear multiple times in the active set, but empty labels (unlabelled * iteration and switch statements) can. A break will match the closest unlabelled or labelled * statement. A continue will match the closest unlabelled or labelled iteration statement. It is * a syntax error if a continue matches a labelled switch statement; because an explicit label cannot * be duplicated, the continue cannot match any valid label outside the switch. * * A side effect of these rules is that a LABEL statement related to a switch should never actually * catch a continue abrupt completion at run-time. Hence an INVALID opcode can be placed in the * continue slot of the switch's LABEL statement.
label: code-design
23109. get before side effects
23110. DUK_HSTRING_H_INCLUDED
23111. avoid empty label at the end of a compound statement
23112. no specific action, resume normal execution
23113. for manual torture testing: tight allocation, useful with valgrind
23114. XXX: skip count_free w/o debug?
23115. everything except object stay as is
23116. identifiers and environment handling
23117. back to fast loop
23118. DUK_USE_JX || DUK_USE_JC
23119. Run fake finalizer. Avoid creating new refzero queue entries * so that we are not forced into a forever loop.
23120. In some cases it may be that lo > hi, or hi < 0; these * degenerate cases happen e.g. for empty arrays, and in * recursion leaves.
23121. is resume, not a tail call
23122. * Basic stack manipulation: swap, dup, insert, replace, etc

23123. **comment:** Note: nargs (and nregs) may be negative for a native, * function, which indicates that the function wants the * input stack "as is" (i.e. handles "vararg" arguments).
label: code-design

23124. * Indirect magic value lookup for Date methods. * * Date methods don't put their control flags into the function magic value * because they wouldn't fit into a LIGHTFUNC's magic field. Instead, the * magic value is set to an index pointing to the array of control flags * below. * * This must be kept in strict sync with genbuiltins.py!

23125. strings may have inner refs (extdata) in some cases

23126. class masks

23127. keep label catcher

23128. state updated, restart bytecode execution

23129. Flags for duk_js_equals_helper().

23130. [...] func this <some bound args> arg1 ... argN _Args]

23131. [requested_id require require.id resolved_id last_comp Duktape modLoaded undefined fresh_require exports module result(ignored)]

23132. DUK_USE_TAILCALL

23133. **comment:** should never be called
label: code-design

23134. denormal

23135. -> loop body

23136. note: this updates refcounts

23137. avoid calling at end of input, will DUK_ERROR (above check suffices to avoid this)

23138. t1 <- (+ r m+)

23139. denormal or zero

23140. mark-and-sweep control

23141. A 'return' statement is only allowed inside an actual function body, * not as part of eval or global code.

23142. tailcall cannot be flagged to resume calls, and a * previous frame must exist

23143. DUK_TOK_LET

23144. Remove a slice from inside buffer.

23145. valstack slot for 2nd token value

23146. one-based -> zero-based

23147. Set .buffer

23148. terminate

23149. max size: radix=2 + sign

23150. remaining actual steps are carried out if standard DefineOwnProperty succeeds

23151. **comment:** The prev/next pointers of the removed duk_heapheader are left as garbage. * It's up to the caller to ensure they're written before inserting the * object back.
label: code-design

23152. {'_func':true}'

23153. **comment:** XXX: support unary arithmetic ivalue (useful?)
label: code-design

23154. E5 Section 15.5.5.2

23155. True if slice is full, i.e. offset is zero and length covers the entire * buffer. This status may change independently of the duk_hbufferobject if * the underlying buffer is dynamic and changes without the hbufferobject * being changed.

23156. avoid calling write callback in detach1()

23157. http://en.wikipedia.org/wiki/Replacement_character#Replacement_character

23158. -> [Object res]

23159. * Property attribute defaults are defined in E5 Section 15 (first * few pages); there is a default for all properties and a special * default for 'length' properties. Variation from the defaults is * signaled using a single flag bit in the bitstream.

23160. IdentifierStart production with Letter and ASCII excluded

23161. **comment:** * DecimalLiteral, HexIntegerLiteral, OctalIntegerLiteral * "pre-parsing", followed by an actual, accurate parser step. * * Note: the leading sign character ('+' or '-') is -not- part of * the production in E5 grammar, and that a DecimalLiteral * starting with a '0' must be followed by a non-digit. Leading * zeroes are syntax errors and must be checked for. * * XXX: the two step parsing process is quite awkward, it would * be more straightforward to allow numconv to parse the longest * valid prefix (it already does that, it only needs to indicate * where the input ended). However, the lexer decodes characters * using a lookup window, so this is not a trivial change.
label: code-design

23162. * The escapes are same as outside a character class, except that \b has a * different meaning, and \B and backreferences are prohibited (see E5 * Section 15.10.2.19). However, it's difficult to share code because we * handle e.g. "\n" very differently: here we generate a single character * range for it.

23163. **comment:** Ensure there are no stale active breakpoint pointers. * Breakpoint list is currently kept - we could empty it * here but we'd need to handle refcounts correctly, and * we'd need a 'thr' reference for that. * * XXX: clear breakpoint on either attach or detach?
label: code-design

23164. end switch

23165. DUK_TOK_LE

23166. XXX: Currently no inspection of threads, e.g. value stack, call * stack, catch stack, etc.

23167. [...] v1(func) v2(pc+flags)]

23168. Matches both +0 and -0 on purpose.

23169. Note: ToPrimitive(object,hint) == [[DefaultValue]](object,hint), * so use [[DefaultValue]] directly.

23170. The code for writing reg_temp + 0 to the left hand side has already * been emitted.

23171. start line of token (first char)

23172. 10: getMonth

23173. If caller is global/eval code, 'caller' should be set to * 'null'. * * XXX: there is no exotic flag to infer this correctly now. * The NEWENV flag is used now which works as intended for * everything (global code, non-strict eval code, and functions) * except strict eval code. Bound functions are never an issue * because 'func' has been resolved to a non-bound function.

23174. x <- y

23175. eat 'var'

23176. cannot have >4G captures

23177. **comment:** not caught by current thread, thread terminates (yield error to resumer); * note that this may cause a cascade if the resumer terminates with an uncaught * exception etc (this is OK, but needs careful testing)
label: code-design

23178. normalize NaN which may not match our canonical internal NaN

23179. Recompute argument count: bound function handling may have shifted.

23180. [...] key val] -> [...]

23181. minval

23182. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

23183. DUK_TOK_ARSHIFT_EQ

23184. Slightly smaller code without explicit flag, but explicit flag * is very useful when 'clen' is dropped.

23185. Here again we parse bytes, and non-ASCII UTF-8 will cause end of * parsing (which is correct except if there are non-shortest encodings). * There is also no need to check explicitly for end of input buffer as * the input is NUL padded and NUL will exit the parsing loop. * * Because no unescaping takes place, we can just scan to the end of the * plain string and intern from the input buffer.

23186. **comment:** XXX: merge this with duk_js_call.c, as this function implements * core semantics (or perhaps merge the two files altogether).
label: code-design

23187. * New length lower than old length => delete elements, then * update length. ** Note: even though a bunch of elements have been deleted, the 'desc' is * still valid as properties haven't been resized (and entries compacted).

23188. * Program code (global and eval code) has an implicit return value * from the last statement value (e.g. eval("1; 2+3;") returns 3). * This is not the case with functions. If implicit statement return * value is requested, all statements are coerced to a register * allocated here, and used in the implicit return statement below.

23189. * Pass 1

23190. Use unsigned arithmetic to optimize comparison.

23191. type and control flags, label number

23192. DUK_TOK_RSHIFT_EQ

23193. probe steps (see genhashsizes.py), currently assumed to be 32 entries long * (DUK_UTIL_GET_HASH_PROBE_STEP macro).

23194. [... enum_target res trap_result]

23195. 1111 1110 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [36 bits]

23196. * Conversions and coercions * * The conversion/coercions are in-place operations on the value stack. * Some operations are implemented here directly, while others call a * helper in duk_js_ops.c after validating arguments.

23197. **comment:** Select a thread for mark-and-sweep use. ** XXX: This needs to change later.
label: code-design

23198. Automatic filename: 'eval' or 'input'.

23199. cumulative opcode execution count (overflows are OK)

23200. Fast path for fastints

23201. **comment:** XXX: any chance of merging these three similar but still slightly * different algorithms so that footprint would be reduced?
label: code-design

23202. comp_ctx->lex has been pre-initialized by caller: it has been * zeroed and input/input_length has been set.

23203. 'input'

23204. countdown state

23205. switch

23206. * Return to bytecode executor, which will resume execution from * the topmost activation.

23207. Shared exit handling for object/array serialization.

23208. input string scan

23209. as elements

23210. * Rewind lexer. ** duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp * literal" mode with current strictness. ** curr_token line number info should be initialized for pass 2 before * generating prologue, to ensure prologue bytecode gets nice line numbers.

23211. Shared helper for Object.getOwnPropertyNames() and Object.keys(). * Magic: 0=getOwnPropertyNames, 1=Object.keys.

23212. no need to unwind

23213. When looking for .fileName/.lineNumber, blame first * function which has a .fileName.

23214. **comment:** XXX: if attempt to push beyond allocated valstack, this double fault * handling fails miserably. We should really write the double error * directly to thr->heap->lj.value1 and avoid valstack use entirely.
label: code-design

23215. Strict functions don't get their 'caller' updated.

23216. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined fresh_require exports module mod_func]

23217. currently nop

23218. [... closure template]

23219. * A tiny random number generator. ** Currently used for Math.random(). ** http://www.woodmann.com/forum/archive/index.php/t-3100.html

23220. 'func' on stack (borrowed reference)

23221. **comment:** * "prototype" is, by default, a fresh object with the "constructor" * property. ** Note that this creates a circular reference for every function * instance (closure) which prevents refcount-based collection of * function instances. ** XXX: Try to avoid creating the default prototype object, because * many functions are not used as constructors and the default * prototype is unnecessary. Perhaps it could be created on-demand * when it is first accessed?
label: code-design

23222. -> [... val this]

23223. **comment:** code emission temporary
label: code-design

23224. * TypedArray.prototype.set() ** TypedArray set() is pretty interesting to implement because: ** - The source argument may be a plain array or a typedarray. If the * source is a TypedArray, values are decoded and re-encoded into the * target (not as a plain byte copy). This may happen even when the * element byte size is the same, e.g. integer values may be re-encoded * into floats. ** - Source and target may refer to the same underlying buffer, so that * the set() operation may overlap. The specification requires that this * must work as if a copy was made before the operation. Note that this * is NOT a simple memmove() situation because the source and target * byte sizes may be different -- e.g. a 4-byte source (Int8Array) may * expand to a 16-byte target (Uint32Array) so that the target overlaps * the source both from beginning and the end (unlike in typical memmove). ** - Even if 'buf' pointers of the source and target differ, there's no * guarantee that their memory areas don't overlap. This may be the * case with external buffers. ** Even so, it is nice to optimize for the common case: ** - Source and target separate buffers or non-overlapping. ** - Source and target have a compatible type so that a plain byte copy * is possible. Note that while e.g. uint8 and int8 are compatible * (coercion one way or another doesn't change the byte representation), * e.g. int8 and uint8clamped are NOT compatible when writing int8 * values into uint8clamped typedarray (-1 would clamp to 0 for instance). ** See test-bi-typedarray-proto-set.js.

23225. (quotient-remainder (* r B) s) using a dummy subtraction loop

23226. Note: either pointer may be NULL (at entry), so don't assert

23227. String-number-buffer/object -> coerce object to primitive (apparently without hint), then try again.

23228. * Property handling ** The API exposes only the most common property handling functions. * The caller can invoke Ecmascript built-ins for full control (e.g. * defineProperty, getOwnPropertyDescriptor).

23229. Format of magic, bits: * 0...1: elem size shift (0-3) * 2...5: elem type (DUK_HBUFFEROBJECT_ELEM_xxx)

23230. **comment:** Slow gathering of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. Handling * of negative numbers is a bit non-obvious in both cases.
label: code-design

23231. Built-in 'name' is not writable by default. Function '.name' * is writable to allow user code to set a '.name' on a native * function.

23232. reuse last emitted atom for remaining 'infinite' quantifier

23233. TypeError if rval is not an object (or lightfunc which should behave * like a Function instance).

23234. * Dragon4 slow path digit generation.

23235. Rule control flags.

23236. USE_PROP_HASH_PART

23237. don't compact objects; needed during object property allocation resize

23238. r <- (remainder (* r B) s)

23239. [... error arr]

23240. String is an own (virtual) property of a lightfunc.

23241. 16: getHours

23242. 'debugger'

23243. * String manipulation

23244. fmod() return value has same sign as input (negative) so * the result here will be in the range]-2,0], 1 indicates * odd. If x is -Infinity, NaN is returned and the odd check * always concludes "not odd" which results in desired outcome.

23245. DUK_USE_DEBUGGER_INSPECT

23246. Lookup 'key' from arguments internal 'map', perform a variable write if mapped. * Used in E5 Section 10.6 algorithm for [[DefineOwnProperty]] (used by [[Put]]). * Assumes stack top contains 'put' value (which is NOT popped).

23247. * Macro for object validity check ** Assert for currently guaranteed relations between flags, for instance.

23248. **comment:** Process debug messages until we are no longer paused.

label: code-design

23249. * Entries part
23250. [in_val in_val]
23251. DUK_TOK_LBRACKET
23252. 36 bits
23253. for zero size allocations NULL is allowed
23254. -> [... obj key value]
23255. * Some change(s) need to be made. Steps 7-11.
23256. -1 = error, -2 = allowed whitespace, -3 = padding ('='), 0...63 decoded bytes
23257. duk_new() will call the constructor using duk_handle_call(). * A constructor call prevents a yield from inside the constructor, * even if the constructor is an EcmaScript function.
23258. Select a safe loop count where no output checks are * needed assuming we won't encounter escapes. Input * bound checks are not necessary as a NUL (guaranteed) * will cause a SyntaxError before we read out of bounds.
23259. end of diff run
23260. * Buffer
23261. Note: ctx.steps is intentionally not reset, it applies to the entire unanchored match
23262. **comment:** XXX: use the pointer as a seed for now: mix in time at least
label: code-design
23263. XXX: stringtable emergency compaction?
23264. thr->ptr_curr_pc is set by bytecode executor early on entry
23265. Allow leading plus sign
23266. if slice not fully valid, treat as error
23267. Now two entries in the same slot, alloc list
23268. * The loop below must avoid issues with potential callstack * reallocations. A resize (and other side effects) may happen * e.g. due to finalizer/errhandler calls caused by a refzero or * mark-and-sweep. Arbitrary finalizers may run, because when * an environment record is refzero'd, it may refer to arbitrary * values which also become refzero'd. * So, the pointer 'p' is re-looked-up below whenever a side effect * might have changed it.
23269. thr->heap->lj.value1 is already the value to throw
23270. Trailing whitespace has been eaten by duk_dec_value(), so if * we're not at end of input here, it's a SyntaxError.
23271. If user callback did not return source code, module loading * is finished (user callback initialized exports table directly).
23272. **comment:** * Emit a final RETURN. ** It would be nice to avoid emitting an unnecessary "return" opcode * if the current PC is not reachable. However, this cannot be reliably * detected; even if the previous instruction is an unconditional jump, * there may be a previous jump which jumps to current PC (which is the * case for iteration and conditional statements, for instance).
label: code-design
23273. * Because buffer values are often looped over, a number fast path * is important.
23274. starting line number
23275. x <- x - y, use t as temp
23276. EQUALITY EXPRESSION
23277. * Unicode support tables automatically generated during build.
23278. return thread
23279. **comment:** No catch variable, e.g. a try-finally; replace LDCONST with * NOP to avoid a bogus LDCONST.
label: code-design
23280. 0x08583b00
23281. Signed shift, named "arithmetic" (asl) because the result * is signed, e.g. 4294967295 << 1 -> -2. Note that result * must be masked.
23282. fmax() with args -0 and +0 is not guaranteed to return * +0 as EcmaScript requires.
23283. not a wordchar
23284. XXX: a "first" flag would suffice
23285. reg count
23286. Buffer used for generated digits, values are in the range [0,B-1].
23287. assume memset with zero size is OK
23288. If ctx->offset >= ctx->length, we "shift zeroes in" * instead of croaking.
23289. Define own property without inheritance looks and such. This differs from * [[DefineOwnProperty]] because special behaviors (like Array 'length') are * not invoked by this method. The caller must be careful to invoke any such * behaviors if necessary.
23290. Parse a number, other than NaN or +/- Infinity
23291. Must be a "string object", i.e. class "String"
23292. no decref needed for a number
23293. [...] | (crud)
23294. x <- y - z, require x >= y => z >= 0, i.e. y >= z
23295. 26 bits
23296. Used to support single-byte stream lookahead.
23297. Filename/line from C macros (_FILE_, _LINE_) are added as an * entry with a special format: (string, number). The number contains * the line and flags.
23298. -> [ToObject(this)]
23299. Buffer sizes for some UNIX calls. Larger than strictly necessary * to avoid Valgrind errors.
23300. '\Udeadbeef'
23301. * every(), some(), forEach(), map(), filter()
23302. Test without volatiles
23303. Lexer codepoint with additional info like offset/line number
23304. Low memory options
23305. Constants should be signed so that signed arithmetic involving them * won't cause values to be coerced accidentally to unsigned.
23306. Note: either pointer may be NULL (at entry), so don't assert.
23307. **comment:** Portability workaround is required for platforms without * unaligned access. The replacement code emulates little * endian access even on big endian architectures, which is * OK as long as it is consistent for a build.
label: code-design
23308. tail call -> reuse current "frame"
23309. refcount is unsigned, so always true
23310. work list for objects to be finalized (by mark-and-sweep)
23311. **comment:** XXX: remove the preventcount and make yield walk the callstack? * Or perhaps just use a single flag, not a counter, faster to just * set and restore?
label: code-design
23312. **comment:** still in use with verbose messages
label: code-design
23313. stack is unbalanced, but: [<something> buf]
23314. parse ranges until character class ends
23315. Note: buffer is dynamic so that we can 'steal' the actual * allocation later.
23316. 'delete'
23317. **comment:** unused now, not needed until Turkish/Azeri
label: requirement
23318. No platform-specific parsing, this is not an error.
23319. mark finalizable as reachability roots

23320. **comment:** XXX: Add a proper condition. If formals list is omitted, recheck * handling for 'length' in duk_js_push_closure(); it currently relies * on _Formals being set. Removal may need to be conditional to debugging * being enabled/disabled too.
label: code-design

23321. **comment:** XXX: naming is inconsistent: ATOM_END_GROUP ends an ASSERT_START_LOOKAHEAD
label: code-design

23322. ToUint32() coercion required

23323. 'global'

23324. not popped by side effect

23325. Add a number containing: pc, activation flags. * * PC points to next instruction, find offending PC. Note that * PC == 0 for native code.

23326. Check array density and indicate whether or not the array part should be abandoned.

23327. * Duktape.Buffer: toString(), valueOf()

23328. descriptor type specific checks

23329. DUK_USE_ERRTHROW

23330. * Longjmp and other control flow transfer for the bytecode executor. * * The longjmp handler can handle all longjmp types: error, yield, and * resume (pseudotypes are never actually thrown). * * Error policy for longjmp: should not ordinarily throw errors; if errors * occur (e.g. due to out-of-memory) they bubble outwards rather than being * handled recursively.

23331. unary minus

23332. ignore non-strings

23333. Clamp to target's end if too long. * * NOTE: there's no overflow possibility in the comparison; * both target_ustart and copy_size are >= 0 and based on * values in duk_int_t range. Adding them as duk_uint_t * values is then guaranteed not to overflow.

23334. move pivot to its final place

23335. Setting to NULL causes 'caller' to be set to * 'null' as desired.

23336. **comment:** * Global state for working around missing variadic macros
label: code-design

23337. * Allocate and initialize a new entry, resizing the properties allocation * if necessary. Returns entry index (e_idx) or throws an error if alloc fails. * * Sets the key of the entry (increasing the key's refcount), and updates * the hash part if it exists. Caller must set value and flags, and update * the entry value refcount. A decref for the previous value is not necessary.

23338. tolerates NULL h_buf

23339. forced_reg

23340. DUK_TOK_PACKAGE

23341. B -> target register * C -> constant index of identifier name

23342. no match, next case

23343. * Interned identifier is compared against reserved words, which are * currently interned into the heap context. See genbuiltins.py. * * Note that an escape in the identifier disables recognition of * keywords; e.g. "\u0069f = 1;" is a valid statement (assigns to * identifier named "if"). This is not necessarily compliant, * see test-dec-escaped-char-in-keyword.js. * * Note: "get" and "set" are awkward. They are not officially * ReservedWords (and indeed e.g. "var set = 1;" is valid), and * must come out as DUK_TOK_IDENTIFIER. The compiler needs to * work around this a bit.

23344. NULL if tv_obj is primitive

23345. lj value2: thread

23346. Note: use 'curr' as a temp propdesc

23347. Setter APIs detect special year numbers (0...99) and apply a +1900 * only in certain cases. The legacy getYear() getter applies -1900 * unconditionally.

23348. * substring(), substr(), slice()

23349. 30: setHours

23350. Expression, ']' terminates

23351. * Number checkers

23352. E5 Section 11.2.3, step 6.a.i

23353. -> [hobject props]

23354. XXX: shared handling for 'duk__expr_lhs'?

23355. Fix for <https://github.com/svaarala/duktape/issues/155>: * If 'default' is first clause (detected by pc_prevcase < 0) * we need to ensure we stay in the matching chain.

23356. eat 'in'

23357. This may trigger mark-and-sweep with arbitrary side effects, * including an attempted resize of the object we're resizing, * executing a finalizer which may add or remove properties of * the object we're resizing etc.

23358. Call with count == DUK_LEXER_WINDOW_SIZE to fill buffer initially.

23359. zero-based month

23360. * Must be very careful here, every DECREF may cause reallocation * of our valstack.

23361. expect_token

23362. periods

23363. idx of first argument on stack

23364. Add trailer if: * a) non-empty input * b) empty input and no (zero size) match found (step 11)

23365. check stack first

23366. mod_id may be NULL

23367. http://en.wikipedia.org/wiki/ANSI_escape_code

23368. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway

23369. DUK_TOK_SET

23370. * Lookup variable and update in-place if found.

23371. 'Int8Array'

23372. Last component name in resolved path

23373. Must match src/extract_chars.py, generate_match_table3().

23374. Resize target buffer for requested size. Called by the macro only when the * fast path test (= there is space) fails.

23375. Ready to make the copy. We must proceed element by element * and must avoid any side effects that might cause the buffer * validity check above to become invalid. * * Although we work through the value stack here, only plain * numbers are handled which should be side effect safe.

23376. **comment:** * Compared to a direct macro expansion this wrapper saves a few * instructions because no heap dereferencing is required.
label: code-design

23377. Force pause if we were doing "step into" in another activation.

23378. allow fmt==NULL

23379. **comment:** * Reallocate property allocation, moving properties to the new allocation. * * Includes key compaction, rehashing, and can also optionally abandoning * the array part, 'migrating' array entries into the beginning of the * new entry part. Arguments are not validated here, so e.g. new_h_size * MUST be a valid prime. * * There is no support for in-place reallocation or just compacting keys * without resizing the property allocation. This is intentional to keep * code size minimal. * * The implementation is relatively straightforward, except for the array * abandonment process. Array abandonment requires that new string keys * are interned, which may trigger GC. All keys interned so far must be * reachable for GC at all times; valstack is used for that now. * * Also, a GC triggered during this reallocation process must not interfere * with the object being resized. This is currently controlled by using * heap->mark_and_sweep_base_flags to indicate that no finalizers will be * executed (as they can affect ANY object) and no objects are compacted * (it would suffice to protect this particular object only, though). * * Note: a non-checked variant would be nice but is a bit tricky to * implement for the array abandonment process. It's easy for * everything else. * * Note: because we need to potentially resize the valstack (as part * of abandoning the array part), any tval pointers to the valstack * will become invalid after this call.
label: code-design

23380. Note: target registers a and a+1 may overlap with DUK_REGP(b) * and DUK_REGCONSTP(c). Careful here.

23381. duk_hcompiledfunction flags; quite version specific

23382. these are not necessarily 0 or 1 (may be other non-zero), that's ok

23383. +1 = finally

23384. Clamping needed if duk_int_t is 64 bits.
23385. 'id'
23386. output bufwriter
23387. -> x in [-2**31,2**31[
23388. DUK_USE_SELF_TESTS
23389. built-in strings
23390. * Process messages and send status if necessary. ** If we're paused, we'll block for new messages. If we're not * paused, we'll process anything we can peek but won't block * for more. Detach (and re-attach) handling is all localized * to duk_debug_process_messages() too. ** Debugger writes outside the message loop may cause debugger * detach1 phase to run, after which dbg_read_cb == NULL and * dbg_detaching != 0. The message loop will finish the detach * by running detach2 phase, so enter the message loop also when * detaching.
23391. * The error object has been augmented with a traceback and other * info from its creation point -- usually the current thread. * The error handler, however, is called right before throwing * and runs in the yielder's thread.
23392. * Error throwing helpers
23393. [... new_glob new_env new_glob new_glob]
23394. For lightfuncs, simply read the virtual property.
23395. 0x70-0x7f
23396. stable
23397. XXX: Could add fast path (for each transform callback) with direct byte * lookups (no shifting) and no explicit check for x < 0x80 before table * lookup.
23398. Duktape.Thread.resume()
23399. **comment:** * Note: of native Ecmascript objects, only Function instances * have a [[HasInstance]] internal property. Custom objects might * also have it, but not in current implementation. ** XXX: add a separate flag, DUK_HOBJECT_FLAG_ALLOW_INSTANCEOF?
 label: code-design
23400. nothing to NULL
23401. use SameValue instead of non-strict equality
23402. a value is left on stack regardless of rc
23403. Detach is pending; can be triggered from outside the * debugger loop (e.g. Status notify write error) or by * previous message handling. Call detached callback * here, in a controlled state, to ensure a possible * reattach inside the detached_cb is handled correctly. ** Recheck for detach in a while loop: an immediate * reattach involves a call to duk_debugger_attach() * which writes a debugger handshake line immediately * inside the API call. If the transport write fails * for that handshake, we can immediately end up in a * "transport broken, detaching" case several times here. * Loop back until we're either cleanly attached or * fully detached. ** NOTE: Reset dbg_processing = 1 forcibly, in case we * re-attached; duk_debugger_attach() sets dbg_processing * to 0 at the moment.
23404. Run the finalizer, duk_hobject_run_finalizer() sets FINALIZED. * Next mark-and-sweep will collect the object unless it has * become reachable (i.e. rescued). FINALIZED prevents the * finalizer from being executed again before that.
23405. Note: this must match ToObject() behavior
23406. 'new' MemberExpression
23407. The code below is incorrect if .toString() or .valueOf() have * have been overridden. The correct approach would be to look up * the method(s) and if they resolve to the built-in function we * can safely bypass it and look up the internal value directly. * Unimplemented for now, abort fast path for boxed values.
23408. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx [21 bits]
23409. **comment:** XXX: avoid this at least when enum_target is an Array, it has an * array part, and no ancestor properties were included? Not worth * it for JSON, but maybe worth it for forEach().
 label: code-design
23410. Report it to the debug client
23411. no compact
23412. restricted to regs
23413. dummy so sp won't get updated
23414. * Replace global object.
23415. require a (mutable) temporary as a result (or a const if allowed)
23416. contains value
23417. %s' and NULL is not portable, so special case * it for debug printing.
23418. Compiler SyntaxErrors (and other errors) come first, and are * blamed by default (not flagged "noblame").
23419. Relevant PC is just before current one because PC is * post-incremented. This should match what error augment * code does.
23420. Parse formals.
23421. [... key arg1 ... argN func]
23422. exit bytecode execution with return value
23423. * Status message and helpers
23424. Note: 'func' is popped from valstack here, but it is * already reachable from the activation.
23425. points to LABEL; pc+1 = jump site for break; pc+2 = jump site for continue
23426. DUK_SELFTEST_H_INCLUDED
23427. Duktape object
23428. 0xb0-0xbf
23429. * 'key_obj' tracks keys encountered so far by associating an * integer with flags with already encountered keys. The checks * below implement E5 Section 11.1.5, step 4 for production: ** PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment
23430. [... res]
23431. * UTF-8 decoder which accepts longer than standard byte sequences. * This allows full 32-bit code points to be used.
23432. accessor with defined getter
23433. **comment:** XXX: optimize for packed duk_tval directly?
 label: code-design
23434. * Augment created errors upon creation (not when they are thrown or * rethrown). __FILE__ and __LINE__ are not desirable here; the call * stack reflects the caller which is correct.
23435. allow source elements
23436. one token before
23437. 'filename'
23438. Neutered. We could go into the switch-case safely with * buf == NULL because check_length == 0. To avoid scanbuild * warnings, fail directly instead.
23439. -> x in J-2**32, 2**32[
23440. no need to incref
23441. Ecmascript function
23442. embed: void ptr
23443. Lightfuncs are always native functions and have "newenv".
23444. This upper value has been experimentally determined; debug build will check * bigint size with assertions.
23445. **comment:** XXX: cover this with the generic >1 case?
 label: code-design
23446. * Variant 1 or 3
23447. 'enumerable'
23448. **comment:** * There are two [[Construct]] operations in the specification: ** - E5 Section 13.2.2: for Function objects * - E5 Section 15.3.4.5.2: for "bound" Function objects ** The chain of bound functions is resolved in Section 15.3.4.5.2, * with arguments "piling up" until the [[Construct]] internal * method is called on the final, actual Function object. Note * that the "prototype" property is looked up *only* from the * final object, *before* calling the constructor. ** Currently we follow the bound function chain here to get the * "prototype" property value from the final, non-bound function. * However, we let duk_handle_call() handle the argument "piling" * when the constructor is called. The bound function chain is * thus now processed twice. ** When constructing new Array instances, an unnecessary object is * created and discarded now: the standard [[Construct]] creates an * object, and calls the Array constructor. The Array constructor * returns

an Array instance, which is used as the result value for * the "new" operation; the object created before the Array constructor * call is discarded. ** This would be easy to fix, e.g. by knowing that the Array constructor * will always create a replacement object and skip creating the fallback * object in that case. ** Note: functions called via "new" need to know they are called as a * constructor. For instance, built-in constructors behave differently * depending on how they are called.

label: code-design

23449. zero size not an issue: pointers are valid

23450. fromPresent = true

23451. The counter value is one less than the init value: init value should * indicate how many instructions are executed before interrupt. To * execute 1 instruction (after interrupt handler return), counter must * be 0.

23452. assumes that allocated pointers and alloc funcs are valid * if res exists

23453. DUK_TOK_EQ

23454. * >>> struct.pack('>d', 102030405060).encode('hex') * '4237c17c6dc40000'

23455. XXX: add 'language' argument when locale/language sensitive rule * support added.

23456. Currently there are no deletable virtual properties, but * with force_flag we might attempt to delete one.

23457. **comment:** XXX: This helper is a bit awkward because the handling for the different iteration * callers is quite different. This now compiles to a bit less than 500 bytes, so with * 5 callers the net result is about 100 bytes / caller.

label: code-design

23458. Infinity

23459. **comment:** XXX: unoptimal use of temps, resetting

label: code-design

23460. -> [... regexp string] -> [... res_obj]

23461. token type (with reserved words as DUK_TOK_IDENTIFIER)

23462. Note: assumes that these string indexes are 8-bit, genstrings.py must ensure that

23463. start variant 3/4 left-hand-side code (L1 in doc/compiler.rst example)

23464. # argument registers target function wants (< 0 => "as is")

23465. 'magic' constants for Murmurhash2

23466. If something is thrown with the debugger attached and nobody will * catch it, execution is paused before the longjmp, turning over * control to the debug client. This allows local state to be examined * before the stack is unwound. Errors are not intercepted when debug * message loop is active (e.g. for Eval).

23467. Object built-in methods

23468. Skip fully.

23469. switch (ch2)

23470. type

23471. * Allocate memory with garbage collection

23472. * Main switch for statement / source element type.

23473. Shared entry handling for object/array serialization.

23474. overwrite sep

23475. [message error message]

23476. [... parent stash stash] -> [... parent stash]

23477. Non-verbose errors for low memory targets: no file, line, or message.

23478. DUK_BUFOBJ_ARRAYBUFFER

23479. **comment:** * Recursion limit check. ** Note: there is no need for an "ignore recursion limit" flag * for duk_handle_safe_call now.

label: code-design

23480. Exponent handling: if exponent format is used, record exponent value and * fake k such that one leading digit is generated (e.g. digits=123 -> "1.23"). * * toFixed() prevents exponent use; otherwise apply a set of criteria to * match the other API calls (toString(), toPrecision, etc).

23481. **comment:** We want the argument expression value to go to "next temp" * without additional moves. That should almost always be the * case, but we double check after expression parsing. ** This is not the cleanest possible approach.

label: code-design

23482. seconds, doesn't fit into 16 bits

23483. unreferenced w/o tracebacks

23484. We want to avoid making a copy to process set() but that's * not always possible: the source and the target may overlap * and because element sizes are different, the overlap cannot * always be handled with a memmove() or choosing the copy * direction in a certain way. For example, if source type is * uint8 and target type is uint32, the target area may exceed * the source area from both ends! ** Note that because external buffers may point to the same * memory areas, we must ultimately make this check using * pointers. ** NOTE: careful with side effects: any side effect may cause * a buffer resize (or external buffer pointer/length update)!

23485. **comment:** NEXTENUM needs a jump slot right after the main instruction. * When the JUMP is taken, output spilling is not needed so this * workaround is possible. The jump slot PC is exceptionally * plumbed through comp_ctx to minimize call sites.

label: code-design

23486. must be updated to work properly (e.g. creation of 'arguments')

23487. Unescaped '/' ANYWHERE in the regexp (in disjunction, * inside a character class, ...) => same escape works.

23488. 1:1 or special conversions, but not locale/context specific: script generated rules

23489. roughly 0.5 kB

23490. * Value stack resize and stack top adjustment helper. ** XXX: This should all be merged to duk_valstack_resize_raw().

23491. **comment:** * Macro support functions for reading/writing raw data. ** These are done using mempcy to ensure they're valid even for unaligned * reads/writes on platforms where alignment counts. On x86 at least gcc * is able to compile these into a bswap+mov. "Always inline" is used to * ensure these macros compile to minimal code. ** Not really bufwriter related, but currently used together.

label: code-design

23492. If tracebacks are enabled, the '_Tracedata' property is the only * thing we need: 'fileName' and 'lineNumber' are virtual properties * which use '_Tracedata'.

23493. **comment:** XXX: could duk_is_undefined() provide defaulting undefined to 'len' * (the upper limit)?

label: code-design

23494. DUK_TOK_WITH

23495. 0xa0-0xaf

23496. * Table for base-64 decoding

23497. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor).

label: code-design

23498. [... buf]

23499. * Regexp compilation. ** See doc/regexp.rst for a discussion of the compilation approach and * current limitations. ** Regexp bytecode assumes jumps can be expressed with signed 32-bit * integers. Consequently the bytecode size must not exceed 0x7fffffffL. * The implementation casts duk_size_t (buffer size) to duk_(u)int32_t * in many places. Although this could be changed, the bytecode format * limit would still prevent regexps exceeding the signed 32-bit limit * from working. ** XXX: The implementation does not prevent bytecode from exceeding the * maximum supported size. This could be done by limiting the maximum * input string size (assuming an upper bound can be computed for number * of bytecode bytes emitted per input byte) or checking buffer maximum * size when emitting bytecode (slower).

23500. Production allows 'DecimalDigits', including leading zeroes

23501. Note: must coerce to a (writable) temp register, so that e.g. "!x" where x * is a reg-mapped variable works correctly (does not mutate the variable register).

23502. [] -> []

23503. key

23504. * Exception for Duktape internal throws when C++ exceptions are used * for long control transfers. ** Doesn't inherit from any exception base class to minimize the chance * that user code would accidentally catch this exception.

23505. -> [array totalLength buf]
23506. Maximum prototype traversal depth. Sanity limit which handles e.g. * prototype loops (even complex ones like 1->2->3->4->2->3->4).
23507. is valid size
23508. [O Properties obj]
23509. extra coercion of strings is OK
23510. **comment:** XXX: increase ctx->expr_tokens here for every consumed token * (this would be a nice statistic)?
 label: code-design
23511. FAIL
23512. Two digit octal escape, digits validated. * * The if-condition is a bit tricky. We could catch e.g. * '\039' in the three-digit escape and fail it there (by * validating the digits), but we want to avoid extra * additional validation code.
23513. For n == 0, Node.js ignores totalLength argument and * returns a zero length buffer.
23514. * Push result object and init its flags
23515. number of continuation (non-initial) bytes in [0x80,0xbf]
23516. External buffer with 'curr_alloc' managed by user code and pointing to an * arbitrary address. When heap pointer compression is not used, this struct * has the same layout as duk_hbuffer_dynamic.
23517. **comment:** Note: we can reuse 'desc' here
 label: code-design
23518. empty statement with an explicit semicolon
23519. * Constructor
23520. * Array exotic behaviors can be implemented at this point. The local variables * are essentially a 'value copy' of the input descriptor (Desc), which is modified * by the Array [[DefineOwnProperty]] (E5 Section 15.4.5.1).
23521. * Expression parsing. * * Upon entry to 'expr' and its variants, 'curr_tok' is assumed to be the * first token of the expression. Upon exit, 'curr_tok' will be the first * token not part of the expression (e.g. semicolon terminating an expression * statement).
23522. The index range space is conceptually the array part followed by the * entry part. Unlike normal enumeration all slots are exposed here as * is and return 'unused' if the slots are not in active use. In particular * the array part is included for the full a_size regardless of what the * array .length is.
23523. * toString()
23524. shuffle registers if large number of regs/consts
23525. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8
23526. **comment:** XXX: add support for variables to not be register bound always, to * handle cases with a very large number of variables?
 label: code-design
23527. entry_top + 2
23528. [result]
23529. * Determine call type, then finalize activation, shift to * new value stack bottom, and call the target.
23530. [source]
23531. * Arguments object creation. * * Creating arguments objects involves many small details, see E5 Section * 10.6 for the specific requirements. Much of the arguments object exotic * behavior is implemented in duk_hobject_props.c, and is enabled by the * object flag DUK_HOBJECT_FLAG_EXOTIC_ARGUMENTS.
23532. **comment:** XXX: space in valstack? see discussion in duk_handle_call_xxx().
 label: code-design
23533. combines steps 2 and 5; -len ensures max() not needed for step 5
23534. for register-bound and declarative env identifiers
23535. bound functions are never in act 'func'
23536. Lightfuncs cannot be bound.
23537. **comment:** The resulting buffer object gets the same class and prototype as * the buffer in 'this', e.g. if the input is a Node.js Buffer the * result is a Node.js Buffer; if the input is a Float32Array, the * result is a Float32Array. * * For the class number this seems correct. The internal prototype * is not so clear: if 'this' is a bufferobject with a non-standard * prototype object, that value gets copied over into the result * (instead of using the standard prototype for that object type).
 label: code-design
23538. Don't allow mixed padding and actual chars.
23539. reject RegExp literal on next advance() call; needed for handling IdentifierName productions
23540. DUK_TOK_CONTINUE
23541. unbalanced stack on purpose
23542. arguments MUST have an initialized lexical environment reference
23543. **comment:** XXX: sanitize to printable (and maybe ASCII)
 label: code-design
23544. tags
23545. regexp compiler should catch these
23546. **comment:** Current 'this' binding is the value just below idx_bottom. * Previously, 'this' binding was handled with an index to the * (calling) valstack. This works for everything except tail * calls, which must not "cumulate" valstack temps.
 label: code-design
23547. * ctx->prev_token token to process with duk__expr_nud() * ctx->curr_token updated by caller * * Note: the token in the switch below has already been eaten.
23548. -> [... voidp true]
23549. The E5.1 specification does not seem to allow IdentifierPart characters * to be used as identity escapes. Unfortunately this includes '\$', which * cannot be escaped as '\$'; it needs to be escaped e.g. as '\u0024'. * Many other implementations (including V8 and Rhino, for instance) do * accept '\$' as a valid identity escape, which is quite pragmatic. * See: test-regexp-identity-escape-dollar.js.
23550. mp <- b/(e+1)
23551. * Debugger handling for executor restart * * Check for breakpoints, stepping, etc, and figure out if we should execute * in checked or normal mode. Note that we can't do this when an activation * is created, because breakpoint status (and stepping status) may change * later, so we must recheck every time we're executing an activation. * This primitive should be side effect free to avoid changes during check.
23552. at most sizeof(buf) - 1
23553. val
23554. Old solution: don't iterate, incorrect
23555. * API calls related to general value stack manipulation: resizing the value * stack, pushing and popping values, type checking and reading values, * coercing values, etc. * * Also contains internal functions (such as duk_get_tval()), defined * in duk_api_internal.h, with semantics similar to the public API.
23556. No BC shuffling now.
23557. syntactically left-associative but parsed as right-associative
23558. * Local result type for duk_get_identifier_reference() lookup.
23559. **comment:** XXX: would be nice to enumerate an object at specified index
 label: code-design
23560. ''
23561. * Heap stringtable handling, string interning.
23562. By default the global object is read-only which is often much * more of an issue than having read-only built-in objects (like * RegExp, Date, etc). Use a RAM-based copy of the global object * and the global environment object for convenience.
23563. Creation time error augmentation
23564. The casts through duk_intr_pt is to avoid the following GCC warning: * * warning: cast from pointer to integer of different size [-Wpointer-to-int-cast] * * This still generates a /Wp64 warning on VS2010 when compiling for x86.
23565. **comment:** * Note: currently register bindings must be fixed for the entire * function. So, even though the catch variable is in a register * we know, we must use an explicit environment record and slow path * accesses to read/write the catch binding to make closures created * within the catch clause work correctly. This

restriction should * be fixable (at least in common cases) later. ** See: test-bug-catch-binding-2.js. ** XXX: improve to get fast path access to most catch clauses.

label: code-design

23566. check that byte has the form 10xx xxxx

23567. * Memory constants

23568. return 'res' (of right part) as our result

23569. -> [func funcname env]

23570. this variant is used when an assert would generate a compile warning by * being always true (e.g. ≥ 0 comparison for an unsigned value

23571. Stringcache is used for speeding up char-offset-to-byte-offset * translations for non-ASCII strings.

23572. **comment:** Note: strictness is *not* inherited from the current Duktape/C. * This would be confusing because the current strictness state * depends on whether we're running inside a Duktape/C activation * (= strict mode) or outside of any activation (= non-strict mode). * See tests/api/test-eval-strictness.c for more discussion.

label: code-design

23573. * Finalizer check. * * Note: running a finalizer may have arbitrary side effects, e.g. * queue more objects on refzero_list (tail), or even trigger a * mark-and-sweep.

* * Note: quick reject check should match vast majority of * objects and must be safe (not throw any errors, ever).

23574. Get Proxy target object. If the argument is not a Proxy, return it as is. * If a Proxy is revoked, an error is thrown.

23575. * Assertion helpers.

23576. is_setget

23577. * Compute new alloc size and alloc new area. * * The new area is allocated as a dynamic buffer and placed into the * valstack for reachability. The actual buffer is then detached at * the end. * * Note: heap_mark_and_sweep_base_flags are altered here to ensure * no-one touches this object while we're resizing and rehashing it. * The flags must be reset on every exit path after it. Finalizers * and compaction is prevented currently for all objects while it * would be enough to restrict it only for the current object.

23578. replacer function

23579. [...]

23580. **comment:** remove the string (mark DELETED), could also call * duk_heap_string_remove() but that would be slow and * pointless because we already know the slot.

label: code-design

23581. +(function(){}) -> NaN

23582. **comment:** * Date built-ins * * Unlike most built-ins, Date has some platform dependencies for getting * UTC time, converting between UTC and local time, and parsing and * formatting time values. These are all abstracted behind DUK_USE_xxx * config options. There are built-in platform specific providers for * POSIX and Windows, but external providers can also be used. * * See doc/datetime.rst. *

label: code-design

23583. Reject attempt to change a read-only object.

23584. final result is already in 'res'

23585. 21: getUTCSeconds

23586. Eval is just a wrapper now.

23587. With ROM objects "needs refcount update" is true when the value is * heap allocated and is not a ROM object.

23588. 22: getMilliseconds

23589. binary arithmetic using extraops; DUK_EXTRAOP_INSTOF etc

23590. combined algorithm matching E5 Sections 9.5 and 9.6

23591. Evaluates to (duk_uint8_t *) pointing to start of area.

23592. module.id = resolved_id

23593. Property access expressions ('a[b]') are critical to correct * LHS evaluation ordering, see test-dev-assign-eval-order*.js. * We must make sure that the LHS target slot (base object and * key) don't change during RHS evaluation. The only concrete * problem is a register reference to a variable-bound register * (i.e., non-temp). Require temp regs for both key and base. * * Don't allow a constant for the object (even for a number * etc), as it goes into the 'A' field of the opcode.

23594. Maximum number of characters in formatted value.

23595. Read fully.

23596. * Prototypes

23597. For indirect allocs.

23598. always allow 'in', coerce to 'tr' just in case

23599. DUK_USE_DEBUGGER_SUPPORT && (DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT)

23600. Pointer object internal value is immutable

23601. Leave 'value' on stack

23602. prev char

23603. * [[DefaultValue]] (E5 Section 8.12.8) * * ==> implemented in the API.

23604. 'extends'

23605. result: hash_prime(floor(1.2 * e_size))

23606. bp is assumed to be even

23607. stridx_logfunc[] must be static to allow initializer with old compilers like BCC

23608. Positive index can be higher than valstack top but must * not go above allocated stack (equality is OK).

23609. out_desc is left untouched (possibly garbage), caller must use return * value to determine whether out_desc can be looked up

23610. If buffer has been exhausted, truncate bitstream

23611. Can be called multiple times with no harm.

23612. Helper for component setter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), modify one or more components as specified, recompute * the time value, set it as the internal value. Finally, push the * new time value as a return value to the value stack and return 1 * (caller can then tail call us).

23613. error

23614. 1:1 conversion

23615. Encode into a double constant (53 bits can encode $6 \times 8 = 48$ bits + 3-bit length

23616. not LS/PS

23617. invalid codepoint encoding or codepoint

23618. **comment:** packed advance/token number macro used by multiple functions

label: code-design

23619. The JO(value) operation: encode object. * * Stack policy: [object] -> [object].

23620. DUK_TOK_BAND_EQ

23621. remaining

23622. **comment:** * Duktape built-ins * * Size optimization note: it might seem that vararg multipurpose functions * like fin(), enc(), and dec() are not very size optimal, but using a single * user-visible Ecmascript function saves a lot of run-time footprint; each * Function instance takes >100 bytes. Using a shared native helper and a * 'magic' value won't save much if there are multiple Function instances * anyway.

label: code-design

23623. Value is required to be a number in the fast path so there * are no side effects in write coercion.

23624. Faster but value stack overruns are memory unsafe.

23625. register for writing value of 'non-empty' statements (global or eval code), -1 is marker

23626. * Possible pending array length update, which must only be done * if the actual entry write succeeded.

23627. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.

23628. If XUTF-8 decoding fails, treat the offending byte as a codepoint directly * and go forward one byte. This is of course very lossy, but allows some kind * of output to be produced even for internal strings which don't conform to * XUTF-8. All standard Ecmascript strings are always CESU-8, so this behavior * does not violate

the Ecmascript specification. The behavior is applied to * all modes, including Ecmascript standard JSON. Because the current XUTF-8 * decoding is not very strict, this behavior only really affects initial bytes * and truncated codepoints. * * Another alternative would be to scan forwards to start of next codepoint * (or end of input) and emit just one replacement codepoint.

23629. For initial entry use default value; zero forces an * interrupt before executing the first instruction.

23630. already in correct reg

23631. the next 'if' is guaranteed to match after swap

23632. Encoding state. Heap object references are all borrowed.

23633. -> [... repl_value]

23634. no prototype

23635. Note: 'q' is top-1

23636. **comment:** * Abandon array part because all properties must become non-configurable. * Note that this is now done regardless of whether this is always the case * (skips check, but performance problem if caller would do this many times * for the same object; not likely).

label: code-design

23637. **comment:** XXX: incref by count (2) directly

label: code-design

23638. ')

23639. source is a function expression (used for Function constructor)

23640. * Variant 2

23641. Stored in duk_heapheader h_extra16.

23642. Note: intentionally signed.

23643. 'Date'

23644. input linenumber at input_offset (not window[0]), init to 1

23645. **comment:** * Missing bytes snip base64 example * 0 4 XXXX * 1 3 XXX= * 2 2 XX==

label: code-design

23646. * The remaining matches are emitted as sequence of SPLITs and atom * copies; the SPLITs skip the remaining copies and match the sequel. * This sequence needs to be emitted starting from the last copy * because the SPLITs are variable length due to the variable length * skip offset. This causes a lot of memory copying now. * * Example structure (greedy, match maximum # atoms): ** SPLIT1 LSEQ * (atom) * SPLIT1 LSEQ ; <- the byte length of this instruction is needed * (atom) ; to encode the above SPLIT1 correctly * ... * LSEQ:

23647. idx_value

23648. [... this_new | arg1 ... argN]

23649. -----XX XXXX-----

23650. * Get old and new length

23651. **comment:** Hot variables for interpretation. Critical for performance, * but must add sparingly to minimize register shuffling.

label: code-design

23652. 38: getYear

23653. initjs data is NUL terminated

23654. this.raw(buffer)

23655. return h_bufres

23656. Here we rely on duk_hstring instances always being zero * terminated even if the actual string is not.

23657. Must coerce key: if key is an object, it may coerce to e.g. 'length'.

23658. -> [... regexp_instance]

23659. Final result is at stack top.

23660. Strict equality is NOT enough, because we cannot use the same * constant for e.g. +0 and -0.

23661. right exists, [[Put]] regardless whether or not left exists

23662. * Return target object.

23663. reuse 'res' as 'left'

23664. st_used remains the same, DELETED is counted as used

23665. is_join

23666. binding power must be high enough to NOT allow comma expressions directly

23667. Note: these variables must reside in 'curr_func' instead of the global * context: when parsing function expressions, expression parsing is nested.

23668. Abandon array part, moving array entries into entries part. * This requires a props resize, which is a heavy operation. * We also compact the entries part while we're at it, although * this is not strictly required.

23669. _Formals

23670. base of known return values

23671. TimeClip(), which also handles Infinity -> NaN conversion

23672. [... obj]

23673. Encoding endianness must match target memory layout, * build scripts and genbuiltins.py must ensure this.

23674. flags used for property attributes in duk_propdesc and packed flags

23675. directly uses slow accesses

23676. * Finalizer torture. Do one fake finalizer call which causes side effects * similar to one or more finalizers on actual objects.

23677. add AC*2^32

23678. result: updated refcount

23679. Entry level info.

23680. catch depth from current func

23681. indexOf: clamp fromIndex to [-len, len] * (if fromIndex == len, for-loop terminates directly) * * lastIndexOf: clamp fromIndex to [-len - 1, len - 1] * (if clamped to -len-1 -> fromIndex becomes -1, terminates for-loop directly)

23682. decoding

23683. inline initializer for coercers[] is not allowed by old compilers like BCC

23684. * Create normalized 'source' property (E5 Section 15.10.3).

23685. variable access

23686. Double casting for pointer to avoid gcc warning (cast from pointer to integer of different size)

23687. **comment:** XXX: refactor out?

label: code-design

23688. same for both error and each subclass like TypeError

23689. [... res_obj re_obj input bc saved_buf]

23690. B -> register for writing enumerator object * C -> value to be enumerated (register)

23691. entire string is whitespace

23692. 'multiline'

23693. '+'

23694. * Parse an identifier and then check whether it is: * - reserved word (keyword or other reserved word) * - "null" (NullLiteral) * - "true" (BooleanLiteral) * - "false" (BooleanLiteral) * - anything else => identifier * * This does not follow the E5 productions cleanly, but is * useful and compact. * * Note that identifiers may contain Unicode escapes, * see E5 Sections 6 and 7.6. They must be decoded first, * and the result checked against allowed characters. * The above if-clause accepts an identifier start and an * '\ character -- no other token can begin with a '\'. * * Note that "get" and "set" are not reserved words in E5 * specification so they are recognized as plain identifiers * (the tokens DUK_TOK_GET and DUK_TOK_SET are actually not * used now). The compiler needs to work around this. * * Strictly speaking, following Ecmascript longest match * specification, an invalid escape for the first character * should cause a syntax error. However, an invalid escape * for IdentifierParts should just terminate the identifier * early (longest match), and let the next tokenization * fail. For instance Rhino croaks with 'foo\v' when * parsing the identifier. This has little practical impact.

23695. 'lineNumber'

23696. refer to callstack entries below current
23697. statement parsing
23698. **comment:** Pick a destination register. If either base value or key * happens to be a temp value, reuse it as the destination. ** XXX: The temp must be a "mutable" one, i.e. such that no * other expression is using it anymore. Here this should be * the case because the value of a property access expression * is neither the base nor the key, but the lookup result.
label: code-design
23699. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()
23700. ignore retval -> [key]
23701. %ld
23702. * Octal escape or zero escape: * \0 (lookahead not DecimalDigit) * \1 ... \7 (lookahead not DecimalDigit) * \ZeroToThree OctalDigit (lookahead not DecimalDigit)
* \FourToSeven OctalDigit (no lookahead restrictions) * \ZeroToThree OctalDigit OctalDigit (no lookahead restrictions) ** Zero escape is part of the standard syntax. Octal escapes are * defined in E5 Section B.1.2, and are only allowed in non-strict mode. * Any other productions starting with a decimal digit are invalid.
23703. **comment:** XXX: for Object.keys() the [[OwnPropertyKeys]] result (trap result) * should be filtered so that only enumerable keys remain. Enumerability * should be checked with [[GetOwnProperty]] on the original object * (i.e., the proxy in this case). If the proxy has a getOwnPropertyDescriptor * trap, it should be triggered for every property. If the proxy doesn't have * the trap, enumerability should be checked against the target object instead. * We don't do any of this now, so Object.keys() and Object.getOwnPropertyNames() * return the same result now for proxy traps. We still do clean up the trap * result, so that Object.keys() and Object.getOwnPropertyNames() will return a * clean array of strings without gaps.
label: code-design
23704. syntax error
23705. don't push value
23706. 'buffer'
23707. inherit initial strictness from parent
23708. module
23709. Without tracebacks the concrete .fileName and .lineNumber need * to be added directly.
23710. not emergency
23711. * Raw memory calls: relative to heap, but no GC interaction
23712. Lookup 'key' from arguments internal 'map', perform a variable lookup * if mapped, and leave the result on top of stack (and return non-zero). * Used in E5 Section 10.6 algorithms [[Get]] and [[GetOwnProperty]].
23713. Evaluate RHS only when LHS is safe.
23714. debugger detaching; used to avoid calling detach handler recursively
23715. no catchers
23716. * Variant 4
23717. 'package'
23718. nothing now
23719. * Reset trigger counter
23720. * Preliminaries
23721. count is already incremented, take into account
23722. **comment:** * Setters. * * Setters are a bit more complicated than getters. Component setters * break down the current time value into its (normalized) component * parts, replace one or more components with -unnormalized- new values, * and the components are then converted back into a time value. As an * example of using unnormalized values: * * var d = new Date(1234567890); * * is equivalent to: * * var d = new Date(0); * d.setUTCSeconds(1234567890); * * A shared native helper to provide almost all setters. Magic value * contains a set of flags and also packs the "maxnargs" argument. The * helper provides: * * setMilliseconds() * setUTCSeconds() * setUTCMinutes() * setHours() * setUTCHours() * setDate() * setUTCDate() * setMonth() * setUTCMonth() * setFullYear() * setUTCFullYear() * setYear() * * Notes: * * - Date.prototype.setYear() (Section B addition): special year check * is omitted. NaN / Infinity will just flow through and ultimately * result in a NaN internal time value. * * - Date.prototype.setYear() does not have optional arguments for * setting month and day-in-month (like setFullYear()), but we indicate * 'maxnargs' to be 3 to get the year written to the correct component * index in duk__set_part_helper(). The function has nargs == 1, so only * the year will be set regardless of actual argument count.
label: code-design
23723. inline string concatenation
23724. DUK_USE_JX && DUK_USE_JC
23725. **comment:** XXX: ideally this would be just one flag (maybe a derived one) so * that a single bit test is sufficient to check the condition.
label: test
23726. [key] (coerced)
23727. **comment:** XXX: no optimized variants yet
label: requirement
23728. Encode a double (checked by caller) from stack top. Stack top may be * replaced by serialized string but is not popped (caller does that).
23729. This function is only called for objects with array exotic behavior. * The [[DefineOwnProperty]] algorithm for arrays requires that * 'length' can never have a value outside the unsigned 32-bit range, * attempt to write such a value is a RangeError. Here we can thus * assert for this. When Duktape internals go around the official * property write interface (doesn't happen often) this assumption is * easy to accidentally break, so such code must be written carefully. * See test-bi-array-push-maxlen.js.
23730. **comment:** DUK__ADVANCECHARS(lex_ctx, 2) would be correct here, but it unnecessary
label: code-design
23731. 'base64'
23732. [sep ToObject(this) len sep]
23733. **comment:** * Various sanity checks for typing
label: code-design
23734. Check whether statement list ends.
23735. invalidates tv1, tv2
23736. Note: MUST NOT wipe_and_return here, as heap->lj must remain intact
23737. -> sets 'f' and 'e'
23738. Check for maximum string length
23739. * Parse an individual source element (top level statement) or a statement. ** Handles labeled statements automatically (peeling away labels before * parsing an expression that follows the label(s)). ** Upon entry, 'curr Tok' contains the first token of the statement (parsed * in "allow regexp literal" mode). Upon exit, 'curr Tok' contains the first * token following the statement (if the statement has a terminator, this is * the token after the terminator).
23740. **comment:** loop_stack_index could be perhaps be replaced by 'depth', but it's nice * to not couple these two mechanisms unnecessarily.
label: code-design
23741. * Fast path for defining array indexed values without interning the key. * This is used by e.g. code for Array prototype and traceback creation so * must avoid interning.
23742. accessor
23743. set to 1 if any match exists (needed for empty input special case)
23744. this should never be possible, because the switch-case is * comprehensive
23745. avoid double coercion
23746. 'DataView'
23747. * Sweep stringtable
23748. [A B C D E F G H] rel_index = 2, del_count 3, item count 1 * -> [A B F G H] (conceptual intermediate step) * -> [A B . F G H] (placeholder marked) * [A B C F G H] (actual result at this point, C will be replaced)
23749. DUK_TOK_IN
23750. jump to false
23751. **comment:** XXX: resolve macro definition issue or call through a helper function?

label: code-design

23752. **comment:** Sanity workaround for handling functions with a large number of * constants at least somewhat reasonably. Otherwise checking whether * we already have the constant would grow very slow (as it is O(N^2)).

label: code-design

23753. heapobj recursion depth when deep printing is selected

23754. basic types

23755. temp reset is not necessary after duk_parse_stmt(), which already does it

23756. mark-and-sweep: reachable

23757. Error object is augmented at its creation here.

23758. DUK_USE_COMMONJS_MODULES

23759. at least effective 'this'

23760. * Default allocation functions. * * Assumes behavior such as malloc allowing zero size, yielding * a NULL or a unique pointer which is a no-op for free.

23761. DUK_USE_LIGHTFUNC_BUILTINS

23762. some derived types

23763. functions always have a NEWENV flag, i.e. they get a * new variable declaration environment, so only lex_env * matters here.

23764. x == +Infinity

23765. temp_start+0 = key, temp_start+1 = closure

23766. E5.1 Section 15.1.3.2: empty

23767. DUK_TOK_FINALY

23768. **comment:** Static call style.

label: code-design

23769. 'deleteProperty'

23770. Compiling state of one function, eventually converted to duk_hcompiledfunction

23771. to body

23772. Coerce top into Object.prototype.toString() output.

23773. XXX: this is quite clunky. Add Unicode helpers to scan backwards and * forwards with a callback to process codepoints?

23774. actual update happens once write has been completed without * error below.

23775. * 'func' is now a non-bound object which supports [[HasInstance]] * (which here just means DUK_HOBJECT_FLAG_CALLABLE). Move on * to execute E5 Section 15.3.5.3.

23776. Target object must be checked for a conflicting * non-configurable property.

23777. property exists, either 'desc' is empty, or all values * match (SameValue)

23778. [func thisArg argArray]

23779. p overshoots

23780. prefer direct

23781. **comment:** XXX: what about statements which leave a half-cooked value in 'res' * but have no stmt value? Any such statements?

label: code-design

23782. Must restart in all cases because we NULLed thr->ptr_curr_pc.

23783. MAXVAL is inclusive

23784. "release" preallocated temps since we won't need them

23785. temp reg value for start of loop

23786. checked by Duktape.Thread.resume()

23787. * Union aliasing, see misc/clang_aliasing.c.

23788. [offset value littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)

23789. * Replacements for missing platform functions. * * Unlike the originals, fpclassify() and signbit() replacements don't * work on any floating point types, only doubles. The C typing here * mimics the standard prototypes.

23790. eat closing quote

23791. setup many variables in nc_ctx

23792. **comment:** XXX: fast int-to-double

label: code-design

23793. Native function pointer may be different from a void pointer, * and we serialize it from memory directly now (no byte swapping etc).

23794. **comment:** XXX: shorter version for 12-byte representation?

label: code-design

23795. a fresh require() with require.id = resolved target module id

23796. Constructor

23797. elision - flush

23798. temp reg for key literal

23799. Unlike break/continue, throw statement does not allow an empty value.

23800. DUK_TOK_VAR

23801. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is NULL if value is neither an object nor a * lightfunc.

23802. char offset in [0, h_input->clen] (both ends inclusive), checked before entry

23803. NOTE: fmod(x) result sign is same as sign of x, which * differs from what Javascript wants (see Section 9.6).

23804. [... bytecode escaped_source]

23805. [obj key desc value get set curr_value]

23806. >>

23807. msec

23808. unary plus of a number is identity

23809. equals and strict equals

23810. * Loose equality, strict equality, and SameValue (E5 Sections 11.9.1, 11.9.4, * 9.12). These have much in common so they can share some helpers. * * Future work notes: * * - Current implementation (and spec definition) has recursion; this should * be fixed if possible. * * - String-to-number coercion should be possible without going through the * value stack (and be more compact) if a shared helper is invoked.

23811. The debugger protocol doesn't support a plain integer encoding for * the full 32-bit unsigned range (only 32-bit signed). For now, * unsigned 32-bit values simply written as signed ones. This is not * a concrete issue except for 32-bit heaphdr fields. Proper solutions * would be to (a) write such integers as IEEE doubles or (b) add an * unsigned 32-bit dvalue.

23812. **comment:** * XXX: attempt to get the call result to "next temp" whenever * possible to avoid unnecessary register shuffles. * * XXX: CSPROP (and CSREG) can overwrite the call target register, and save one temp, * if the call target is a temporary register and at the top of the temp reg "stack".

label: code-design

23813. tv points to element just below prev top

23814. * Duktape/C function magic

23815. * Identifier handling

23816. error gets its 'name' from the prototype

23817. Use a new environment but there's no 'arguments' object; * delayed environment initialization. This is the most * common case.

23818. E5 Sections 11.9.1, 11.9.3

23819. Capital sigma occurred at "end of word", lowercase to * U+03C2 = GREEK SMALL LETTER FINAL SIGMA. Otherwise * fall through and let the normal rules lowercase it to * U+03C3 = GREEK SMALL LETTER SIGMA.

23820. current digit

23821. * Extern

23822. * Mark objects on finalize_list. *

23823. **comment:** XXX: repetition of stack pre-checks -> helper or macro or inline
label: code-design

23824. **comment:** * Data follows the struct header. The struct size is padded by the * compiler based on the struct members. This guarantees that the * buffer data will be aligned-by-4 but not necessarily aligned-by-8. ** On platforms where alignment does not matter, the struct padding * could be removed (if there is any). On platforms where alignment * by 8 is required, the struct size must be forced to be a multiple * of 8 by some means. Without it, some user code may break, and also * Duktape itself breaks (e.g. the compiler stores duk_tvals in a * dynamic buffer).
label: code-design

23825. 'valueOf'
23826. XXX: potential issue with signed pointers, p_end < p.
23827. * Helpers to resize properties allocation on specific needs.

23828. **comment:** The line number tracking is a bit inconsistent right now, which * affects debugger accuracy. Mostly call sites emit opcodes when * they have parsed a token (say a terminating semicolon) and called * duk__advance(). In this case the line number of the previous * token is the most accurate one (except in prologue where * prev_token.start_line is 0). This is probably not 100% correct * right now.
label: code-design

23829. shared object part
23830. "Have" flags must not be conflicting so that they would * apply to both a plain property and an accessor at the same * time.
23831. [... re_obj input]
23832. Backpointers.
23833. string access cache (codepoint offset -> byte offset) for fast string * character looping; 'weak' reference which needs special handling in GC.
23834. bottom of new func
23835. re_ctx->captures at the start of the atom parsed in this loop
23836. * Scan from shortest distance: * - start of string * - end of string * - cache entry (if exists)
23837. refzero_list treated as reachability roots
23838. * Memory calls.
23839. +0.5 is handled by floor, this is on purpose
23840. Duplicate (shadowing) labels are not allowed, except for the empty * labels (which are used as default labels for switch and iteration * statements). ** We could also allow shadowing of non-empty pending labels without any * other issues than breaking the required label shadowing requirements * of the E5 specification, see Section 12.12.
23841. internal temporary value, used for char classes
23842. **comment:** format is too large, abort
label: code-design
23843. -> [... key val]
23844. 0x40-0x4f
23845. MPUTOBJ emitted by outer loop
23846. Errors are augmented when they are created, not when they are * thrown or re-thrown. The current error handler, however, runs * just before an error is thrown.
23847. -0 is accepted here as index 0 because ToString(-0) == "0" which is * in canonical form and thus an array index.
23848. be_ctx->offset == length of encoded bitstream
23849. 'in'
23850. Duktape.modLoaded[resolved_id] = module
23851. Caller must ensure 'tv' is indeed a fastint!
23852. stage 1 guarantees
23853. typed for duk_unicode_decode_xutf8()
23854. Macros for creating and checking bitmasks for character encoding. * Bit number is a bit counterintuitive, but minimizes code size.
23855. * Compile input string into an executable function template without * arguments. ** The string is parsed as the "Program" production of EcmaScript E5. * Compilation context can be either global code or eval code (see E5 * Sections 14 and 15.1.2.1). * * Input stack: [... filename] * Output stack: [... func_template]
23856. **comment:** XXX: macro? sets both heaphdr and object flags
label: code-design
23857. [... varname]
23858. preinitialize lexer state partially
23859. **comment:** XXX: slightly awkward
label: code-design
23860. Once recursion depth is increased, exit path must decrease * it (though it's OK to abort the fast path).
23861. require
23862. rnd in [lo,hi]
23863. must have catch and/or finally
23864. V
23865. Technically Array.prototype.push() can create an Array with length * longer than 2^32-1, i.e. outside the 32-bit range. The final length * is *not* wrapped to 32 bits in the specification. ** This implementation tracks length with a uint32 because it's much * more practical. ** See: test-bi-array-push-maxlen.js
23866. DUK_TOK_BXOR
23867. [... re_obj input bc saved_buf res_obj val]
23868. No arr_idx update or limit check
23869. [... varname]
23870. fresh require (argument)
23871. * Resolution loop. At the top of the loop we're expecting a valid * term: '.', '..', or a non-empty identifier not starting with a period.
23872. String object internal value is immutable
23873. At this point 'res' holds the potential expression value. * It can be basically any ivalue here, including a reg-bound * identifier (if code above deems it safe) or a unary/binary * operation. Operations must be resolved to a side effect free * plain value, and the side effects must happen exactly once.
23874. highest capture number emitted so far (used as: ++captures)
23875. arbitrary remove only works with double linked heap, and is only required by * reference counting so far.
23876. **comment:** never reached, but avoids warnings of * potentially unused variables.
label: code-design
23877. new entry: previous value is garbage; set to undefined to share write_value
23878. [... name name]
23879. nargs
23880. each call this helper serves has nargs==2
23881. assertion disabled
23882. * Types are different; various cases for non-strict comparison ** Since comparison is symmetric, we use a "swap trick" to reduce * code size.
23883. Ensure dirty state causes a Status even if never process any * messages. This is expected by the bytecode executor when in * the running state.
23884. * A C * B A * C <- sce ==> B * D D
23885. Sign extend: 0x0000##00 -> 0x##000000 -> sign extend to 0xssssss##
23886. * Virtual properties. ** String and buffer indices are virtual and always enumerable, * 'length' is virtual and non-enumerable. Array and arguments * object props have special behavior but are concrete.
23887. * Case conversion
23888. * Helpers for dealing with the input string
23889. const limits
23890. 0xc0-0xcf
23891. The issues below can be solved with better flags
23892. curr and desc are data

23893. If forced reg, use it as destination. Otherwise try to * use either coerced ispec if it is a temporary. ** When using extraops, avoid reusing arg2 as dest because that * would lead to an LDREG shuffle below. We still can't guarantee * dest != arg2 because we may have a forced_reg.

23894. **comment:** XXX: putvar takes a duk_tval pointer, which is awkward and * should be reworked.
label: code-design

23895. bottom of current func

23896. statement is guaranteed to be terminal (control doesn't flow to next statement)

23897. Use a fast break/continue when possible. A fast break/continue is * just a jump to the LABEL break/continue jump slot, which then jumps * to an appropriate place (for break, going through ENDLABEL correctly). * The peephole optimizer will optimize the jump to a direct one.

23898. * Heap buffer representation. ** Heap allocated user data buffer which is either: * 1. A fixed size buffer (data follows header statically) * 2. A dynamic size buffer (data pointer follows header) ** The data pointer for a variable size buffer of zero size may be NULL.

23899. Bernstein hash init value is normally 5381; XOR it in in case pointer low bits are 0

23900. **comment:** Allow headroom for calls during error handling (see GH-191). * We allow space for 10 additional recursions, with one extra * for, e.g. a print() call at the deepest level.
label: code-design

23901. http://en.wikipedia.org/wiki/Exponentiation_by_squaring

23902. chain

23903. [... closure]

23904. **comment:** Calling as a non-constructor is not meaningful.
label: code-design

23905. * Voluntary periodic GC (if enabled)

23906. Step types

23907. 'default'

23908. **comment:** XXX: awkward; we assume there is space for this, overwrite * directly instead?
label: code-design

23909. E5 Section steps 7-8

23910. result object is created and discarded; wasteful but saves code space

23911. Gather in little endian

23912. 'fmt'

23913. not caught by anything before entry level; rethrow and let the * final catcher unwind everything

23914. NUL terminate for convenient C access

23915. 'BYTES_PER_ELEMENT'

23916. We don't log or warn about freeing zero refcount objects * because they may happen with finalizer processing.

23917. Don't allow a zero divisor. Restrict both v1 and * v2 to positive values to avoid compiler specific * behavior.

23918. **comment:** XXX: logic for handling character ranges is now incorrect, it will accept * e.g. [d-z] whereas it should croak from it? SMJS accepts this too, though. *
* Needs a read through and a lot of additional tests.
label: code-design

23919. * Node.js Buffer.prototype.fill()

23920. * Exposed debug macros: debugging disabled

23921. **comment:** No NUL term checks in this debug code.
label: code-design

23922. DUK_BUFOBJ_FLOAT64ARRAY

23923. embed: duk_hobject ptr

23924. getters

23925. DUK_TOK_DELETE

23926. DUK_TOK_BOR

23927. **comment:** pseudotypes, not used in actual longjmps
label: code-design

23928. A -> target reg * BC -> inner function index

23929. refcount code is processing refzero list

23930. [... retval]; popped below

23931. 0x80...0x8f

23932. Get current Ecmascript time (= UNIX/Posix time, but in milliseconds).

23933. Note: all references inside 'data' need to get their refcounts * upped too. This is the case because refcounts are decreased * through every function referencing 'data' independently.

23934. **comment:** NOTE: level is used only by the debugger and should never be present * for an Ecmascript eval().
label: code-design

23935. entry_top + 1

23936. Set datetime parts from stack arguments, defaulting any missing values. * Day-of-week is not set; it is not required when setting the time value.

23937. * ToBoolean() (E5 Section 9.2)

23938. * Shift to new valstack_bottom.

23939. 0 = don't push current value

23940. "." is not accepted in any format

23941. tmp -> res

23942. [body formals source]

23943. **comment:** XXX: for simple cases like x[y] an unnecessary LDREG is * emitted for the base value; could avoid it if we knew that * the key expression is safe (e.g. just a single literal).
label: code-design

23944. Other initial bytes.

23945. Similar to String comparison.

23946. Finally, blame the innermost callstack entry which has a * .fileName property.

23947. skip finalizers; queue finalizable objects to heap_allocated

23948. Note that duk_exprtop() here can clobber any reg above current temp_next, * so any loop variables (e.g. enumerator) must be "preallocated".

23949. * Parse a statement list. ** Handles automatic semicolon insertion and implicit return value. ** Upon entry, 'curr_tok' should contain the first token of the first * statement (parsed in the "allow regexp literal" mode). Upon exit, * 'curr_tok' contains the token following the statement list terminator * (EOF or closing brace).

23950. fixed top level hashtable size (separate chaining)

23951. not present

23952. end marker

23953. handle negative values

23954. invalidated by pushes, so get out of the way

23955. h is NULL for lightfunc

23956. * Error throwing

23957. Non-buffer value is first ToString() coerced, then converted * to a buffer (fixed buffer is used unless a dynamic buffer is * explicitly requested).

23958. **comment:** unused: not accepted in inbound messages
label: code-design

23959. Reuse buffer as is unless buffer has grown large.

23960. Offset is coerced first to signed integer range and then to unsigned. * This ensures we can add a small byte length (1-8) to the offset in * bound checks and not wrap.

23961. internal align target for props allocation, must be 2*n for some n

23962. Set timeval to 'this' from dparts, push the new time value onto the * value stack and return 1 (caller can then tail call us). Expects * the value stack to contain 'this' on the stack top.

23963. target

23964. DUK_FLD_FLOAT

23965. XXX: shared helper fortoFixed(), toExponential(), toPrecision()?

23966. **comment:** * Run an duk_hobject finalizer. Used for both reference counting * and mark-and-sweep algorithms. Must never throw an error. ** There is no return value. Any return value or error thrown by * the finalizer is ignored (although errors are debug logged). ** Notes: ** - The thread used for calling the finalizer is the same as the * 'thr' argument. This may need to change later. ** - The finalizer thread 'top' assertions are there because it is * critical that strict stack policy is observed (i.e. no cruft * left on the finalizer stack).

label: code-design

23967. -> x in [0, 2**32[

23968. rethrow original error

23969. \xffThis'

23970. Round up digits to a given position. If position is out-of-bounds, * does nothing. If carry propagates over the first digit, a '1' is * prepended to digits and 'k' will be updated. Return value indicates * whether carry propagated over the first digit. ** Note that nc_ctx->count is NOT updated based on the rounding position * (it is updated only if carry overflows over the first digit and an * extra digit is prepended).

23971. Check if any lookup above had a negative result.

23972. DUK_USE_DEBUGGER_SUPPORT

23973. * For/for-in main variants are: ** 1. for (ExpressionNoIn_opt; Expression_opt; Expression_opt) Statement * 2. for (var VariableDeclarationNoIn; Expression_opt; Expression_opt) Statement * 3. for (LeftHandSideExpression in Expression) Statement * 4. for (var VariableDeclarationNoIn in Expression) Statement ** Parsing these without arbitrary lookahead or backtracking is relatively * tricky but we manage to do so for now. ** See doc/compiler.rst for a detailed discussion of control flow * issues, evaluation order issues, etc.

23974. The specification is a bit vague what to do if the return * value is not a number. Other implementations seem to * tolerate non-numbers but e.g. V8 won't apparently do a * ToNumber().

23975. DUK_TOK_SUB_EQ

23976. * Do a longjmp call, calling the fatal error handler if no * catchpoint exists.

23977. Fixed header info.

23978. nothing to finalize

23979. throw flag irrelevant (false in std alg)

23980. **comment:** This approach is a bit shorter than a straight * if-else-ladder and also a bit faster.

label: code-design

23981. All lightfunc own properties are non-writable and the lightfunc * is considered non-extensible. However, the write may be captured * by an inherited setter which means we can't stop the lookup here.

23982. name

23983. accept string if next char is NUL (otherwise reject)

23984. * "WhiteSpace" production check.

23985. value1 -> yield value, iserror -> error / normal

23986. Number of characters that the atom matches (e.g. 3 for 'abc'), * -1 if atom is complex and number of matched characters either * varies or is not known.

23987. **comment:** 'funcs' is quite rarely used, so no local for it

label: code-design

23988. reachable so pop OK

23989. may be NULL, too

23990. No entry in the catchstack which would actually catch a * throw can refer to the callstack entry being reused. * There *can* be catchstack entries referring to the current * callstack entry as long as they don't catch (e.g. label sites).

23991. * Heap creation and destruction

23992. [... regexp_object escaped_source]

23993. now expected to fit into a 32-bit integer

23994. Duktape.modSearch(resolved_id, fresh_require, exports, module).

23995. bits 0...1: shift

23996. Architecture, OS, and compiler strings

23997. overflow

23998. must fit into duk_small_int_t

23999. 'continue'

24000. 'value'

24001. Non-critical.

24002. **comment:** XXX: tail call: return duk_push_false(ctx)

label: code-design

24003. **comment:** * Sweep garbage and remove marking flags, and move objects with * finalizers to the finalizer work list. ** Objects to be swept need to get their refcounts finalized before * they are swept. In other words, their target object refcounts * need to be decreased. This has to be done before freeing any * objects to avoid dereferencing dangling pointers (which may happen * even without bugs, e.g. with reference loops) ** Because strings don't point to other heap objects, similar * finalization is not necessary for strings.

label: code-design

24004. Note: target registers a and a+1 may overlap with DUK_REGCONSTP(b) * and DUK_REGCONSTP(c). Careful here.

24005. DUK_USE_DATE_NOW_GETTIMEofday

24006. **comment:** XXX: what's the safest way of creating a negative zero?

label: code-design

24007. **comment:** XXX: several nice-to-have improvements here: * - Use direct reads avoiding value stack operations * - Avoid triggering getters, indicate getter values to debug client * - If side effects are possible, add error catching

label: code-design

24008. ptr may be NULL

24009. E5 Sections 15.10.2.8, 7.3

24010. Target must be stored so that we can recheck whether or not * keys still exist when we enumerate. This is not done if the * enumeration result comes from a proxy trap as there is no * real object to check against.

24011. * Note: currently no fast path for array index writes. * They won't be possible anyway as strings are immutable.

24012. Important to do a fastint check so that constants are * properly read back as fastints.

24013. remove artificial bump

24014. important to use normalized NaN with 8-byte tagged types

24015. Only objects in heap_allocated may have finalizers. Check that * the object itself has a _Finalizer property (own or inherited) * so that we don't execute finalizers for e.g. Proxy objects.

24016. crossed offsets or zero size

24017. [... escaped_source bytecode]

24018. * E5 Section 11.4.8

24019. * Manipulate valstack so that args are on the current bottom and the * previous caller's 'this' binding (which is the value preceding the * current bottom) is replaced with the new 'this' binding: ** [... this_old | (crud) func this_new arg1 ... argN] * --> [... this_new | arg1 ... argN] ** For tail calling to work properly, the valstack bottom must not grow * here; otherwise crud would accumulate on the valstack.

24020. * ToInteger() (E5 Section 9.4)

24021. invalidates tv

24022. * Fatal error * * There are no fatal error macros at the moment. There are so few call * sites that the fatal error handler is called directly.

24023. num_stack_args
24024. property is virtual: used in duk_propdesc, never stored * (used by e.g. buffer virtual properties)
24025. '\xffVarmap'
24026. Compute (extended) utf-8 length without codepoint encoding validation, * used for string interning. ** NOTE: This algorithm is performance critical, more so than string hashing * in some cases. It is needed when interning a string and needs to scan * every byte of the string with no skipping. Having an ASCII fast path * is useful if possible in the algorithm. The current algorithms were * chosen from several variants, based on x64 gcc -O2 testing. See: * <https://github.com/svaarala/duktape/pull/422> ** NOTE: must match src/dukutil.py:duk_unicode_unvalidated_utf8_length().
24027. Only allow Duktape.Buffer when support disabled.
24028. * Digit generation loop. ** Different termination conditions: * * 1. Free format output. Terminate when shortest accurate * representation found. * * 2. Fixed format output, with specific number of digits. * Ignore termination conditions, terminate when digits * generated. Caller requests an extra digit and rounds. * * 3. Fixed format output, with a specific absolute cut-off * position (e.g. 10 digits after decimal point). Note * that we always generate at least one digit, even if * the digit is below the cut-off point already.
24029. number of digits changed
24030. **comment:** re-lookup to update curr.flags * XXX: would be faster to update directly
 label: code-design
24031. * Heap Buffer object representation. Used for all Buffer variants.
24032. **comment:** Note: reuse variables
 label: code-design
24033. _Varmap
24034. = 1 + arg count
24035. * Tables generated with src/gennumdigits.py. ** duk__str2num_digits_for_radix indicates, for each radix, how many input * digits should be considered significant for string-to-number conversion. * The input is also padded to this many digits to give the Dragon4 * conversion enough (apparent) precision to work with. ** duk__str2num_exp_limits indicates, for each radix, the radix-specific * minimum/maximum exponent values (for a Dragon4 integer mantissa) * below and above which the number is guaranteed to underflow to zero * or overflow to Infinity. This allows parsing to keep bigint values * bounded.
24036. 'case'
24037. num_pairs and temp_start reset at top of outer loop
24038. Because size > 0, NULL check is correct
24039. 17: getUTCHours
24040. [... enum] -> [... next_key]
24041. if density < L, abandon array part, L = 3-bit fixed point, e.g. 2 -> 2/8 = 25%
24042. Encode to CESU-8; 'out' must have space for at least * DUK_UNICODE_MAX_CESU8_LENGTH bytes; codepoints above U+10FFFF * will encode to garbage but won't overwrite the output buffer.
24043. * Finalize stack
24044. Only a reg fits into 'A' so coerce 'res' into a register * for PUTVAR. ** XXX: here the current A/B/C split is suboptimal: we could * just use 9 bits for reg_res (and support constants) and 17 * instead of 18 bits for the varname const index.
24045. * Helper for setting up var_env and lex_env of an activation, * assuming it does NOT have the DUK_HOBJECT_FLAG_NEWENV flag.
24046. no need for 'caller' post-check, because 'key' must be an array index
24047. [arg undefined]
24048. "
24049. * String-to-number conversion
24050. We can't shortcut zero here if it goes through special formatting * (such as forced exponential notation).
24051. 0x10...0x1f
24052. Main operation
24053. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8CLAMPED * Note: INT8 is -not- copy compatible, e.g. -1 would coerce to 0x00.
24054. invalidated
24055. DUK_USE_CPP_EXCEPTIONS
24056. max(incl)
24057. * Arguments objects have exotic [[DefineOwnProperty]] which updates * the internal 'map' of arguments for writes to currently mapped * arguments. More concretely, writes to mapped arguments generate * a write to a bound variable. ** The [[Put]] algorithm invokes [[DefineOwnProperty]] for existing * data properties and new properties, but not for existing accessors. * Hence, in E5 Section 10.6 ([[DefinedOwnProperty]] algorithm), we * have a Desc with 'Value' (and possibly other properties too), and * we end up in step 5.b.i.
24058. A top-level assignment is e.g. "x = y;". For these it's safe * to use the RHS as-is as the expression value, even if the RHS * is a reg-bound identifier. The RHS ('res') is right associative * so it has consumed all other assignment level operations; the * only relevant lower binding power construct is comma operator * which will ignore the expression value provided here. Usually * the top level assignment expression value is ignored, but it * is relevant for e.g. eval code.
24059. **comment:** XXX: use helper for size optimization
 label: code-design
24060. 'ignoreCase'
24061. Note: env_thr != thr is quite possible and normal, so careful * with what thread is used for valstack lookup.
24062. frozen and sealed
24063. tv -> value just before prev top value; must relookup
24064. * Found existing inherited plain property. * Do an access control check, and if OK, write * new property to 'orig'.
24065. initial estimate for ASCII only codepoints
24066. Parser separator masks.
24067. just in case callback is broken and won't write 'x'
24068. **comment:** NEXTENUM needs a jump slot right after the main opcode. * We need the code emitter to reserve the slot: if there's * target shuffling, the target shuffle opcodes must happen * after the jump slot (for NEXTENUM the shuffle opcodes are * not needed if the enum is finished).
 label: code-design
24069. then to a register
24070. retval to result[i]
24071. Sanity limit on max number of properties (allocated, not necessarily used). * This is somewhat arbitrary, but if we're close to 2**32 properties some * algorithms will fail (e.g. hash size selection, next prime selection). * Also, we use negative array/entry table indices to indicate 'not found', * so anything above 0x80000000 will cause trouble now.
24072. expect EOF instead of {
24073. resizing parameters
24074. value from target
24075. this is especially critical to avoid another write call in detach1()
24076. **comment:** Stripping the filename might be a good idea * ("foo/bar/quux.js" -> logger name "quux"), * but now used verbatim.
 label: code-design
24077. return the argument object
24078. new buffer of specified size
24079. * Conversion helpers
24080. %x; only 16 bits are guaranteed
24081. mix-in value for computing string hashes; should be reasonably unpredictable
24082. 'debug'
24083. keep default instance
24084. Make buffer compact, matching current written size.
24085. explicit semi follows
24086. * Packed 8-byte representation

24087. **comment:** XXX: currently no handling for non-allowed identifier characters, * e.g. a '{' in the function name.
label: code-design
24088. No support for lvalues returned from new or function call expressions. * However, these must NOT cause compile-time SyntaxErrors, but run-time * ReferenceErrors. Both left and right sides of the assignment must be * evaluated before throwing a ReferenceError. For instance: ** f() = g(); ** must result in f() being evaluated, then g() being evaluated, and * finally, a ReferenceError being thrown. See E5 Section 11.13.1.
24089. **comment:** XXX: awkward helpers
label: code-design
24090. advance, expecting current token to be a specific token; parse next token in regexp context
24091. stack prepped for func call: [... trap handler]
24092. '{"_nan":true}'
24093. tagged null pointers should never occur
24094. * Heap header definition and assorted macros, including ref counting. * Access all fields through the accessor macros.
24095. * Finally, longjmp
24096. e.g. DUK_OP_POSTINCP
24097. Allow leading minus sign
24098. copy to buffer with spare to avoid Valgrind gripes from strftime
24099. throw
24100. 4: toLocaleDateString
24101. eat 'with'
24102. [... this func]
24103. setters
24104. step 4.c
24105. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: ** signbit(x) == signbit(y)
24106. check func supports [[HasInstance]] (this is checked for every function * in the bound chain, including the final one)
24107. There is no need to NUL delimit the sscanf() call: trailing garbage is * ignored and there is always a NUL terminator which will force an error * if no error is encountered before it. It's possible that the scan * would scan further than between [js_ctx->p,p[though and we'd advance * by less than the scanned value. * * Because pointers are platform specific, a failure to scan a pointer * results in a null pointer which is a better placeholder than a missing * value or an error.
24108. **comment:** XXX: relocate
label: code-design
24109. match fail
24110. * Make a value copy of the input val. This ensures that * side effects cannot invalidate the pointer.
24111. DUK_USE_DATE_TZO_GMTIME
24112. Treat like debugger statement: nop
24113. Get valstack index for the func argument or throw if insane stack.
24114. **comment:** 'd3' is never NaN, so no need to normalize
label: code-design
24115. XXX: push_uint_string / push_u32_string
24116. return value from Duktape.Thread.resume()
24117. For the case n==1 Node.js doesn't seem to type check * the sole member but we do it before returning it. * For this case only the original buffer object is * returned (not a copy).
24118. end of loop (careful with len==0)
24119. Result is always fastint compatible.
24120. Handle a BREAK/CONTINUE opcode. Avoid using longjmp() for BREAK/CONTINUE * handling because it has a measurable performance impact in ordinary * environments and an extreme impact in Emscripten (GH-342).
24121. DUK_TOK_TRY
24122. Optional UTC conversion.
24123. 6
24124. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module]
24125. * Self test main
24126. prototype is lex_env before catcher created
24127. ToInteger() coercion; NaN -> 0, infinities are clamped to 0 and 10
24128. Keep throwing an error whenever we get here. The unusual values * are set this way because no instruction is ever executed, we just * throw an error until all try/catch/finally and other catchpoints * have been exhausted. Duktape/C code gets control at each protected * call but whenever it enters back into Duktape the RangeError gets * raised. User exec timeout check must consistently indicate a timeout * until we've fully bubbled out of Duktape.
24129. tc1 = true, tc2 = false
24130. no need to handle thread state book-keeping here
24131. Not found. Empty string case is handled specially above.
24132. 4 low bits
24133. **comment:** XXX: option to pretend property doesn't exist if sanity limit is * hit might be useful.
label: code-design
24134. Use explicit steps in computation to try to ensure that * computation happens with intermediate results coerced to * double values (instead of using something more accurate). * E.g. E5.1 Section 15.9.1.11 requires use of IEEE 754 * rules (= EcmaScript '+' and '*' operators). * * Without 'volatile' even this approach fails on some platform * and compiler combinations. For instance, gcc 4.8.1 on Ubuntu * 64-bit, with -m32 and without -std=c99, test-bi-date-canceling.js * would fail because of some optimizations when computing tmp_time * (MakeTime below). Adding 'volatile' to tmp_time solved this * particular problem (annoyingly, also adding debug prints or * running the executable under valgrind hides it).
24135. **comment:** Smaller heaphdr than for other objects, because strings are held * in string intern table which requires no link pointers. Much of * the 32-bit flags field is unused by flags, so we can stuff a 16-bit * field in there.
label: code-design
24136. **comment:** Delete semantics are a bit tricky. The description in E5 specification * is kind of confusing, because it distinguishes between resolvability of * a reference (which is only known at runtime) seemingly at compile time * (= SyntaxError throwing).
label: code-design
24137. [... enum] -> [...]
24138. * Augment an error being created using Duktape specific properties * like _Tracedata or .fileName/.lineNumber.
24139. misc constants and helper macros
24140. zero-based day
24141. 28: setMinutes
24142. non-object base, no offending virtual property
24143. Fixed seed value used with ROM strings.
24144. duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp literal" mode with current strictness
24145. E5 Sections 11.8.1, 11.8.5; x < y
24146. may be < thr->catchstack initially
24147. complex, multicharacter conversion
24148. Not enough data to provide a full window, so "scroll" window to * start of buffer and fill up the rest.
24149. E5.1 Section 15.1.3.1: uriReserved + '#'
24150. **comment:** XXX: should there be an error or an automatic detach if * already attached?
label: code-design

24151. For user code this could just return 1 (strict) always * because all Duktape/C functions are considered strict, * and strict is also the default when nothing is running.
* However, Duktape may call this function internally when * the current activation is an Ecmascript function, so * this cannot be replaced by a 'return 1' without fixing * the internal call sites.

24152. "" was eaten by caller

24153. expression is left-hand-side compatible

24154. string is a reserved word (strict)

24155. **comment:** XXX: Output type could be encoded into native function 'magic' value to * save space. For setters the stridx could be encoded into 'magic'.
label: code-design

24156. **comment:** * Manipulation of thread stacks (valstack, callstack, catchstack). * * Ideally unwinding of stacks should have no side effects, which would * then favor separate unwinding and shrink check primitives for each * stack type. A shrink check may realloc and thus have side effects. * * However, currently callstack unwinding itself has side effects, as it * needs to DECREF multiple objects, close environment records, etc. * Stacks must thus be unwound in the correct order by the caller. * * (XXX: This should be probably reworked so that there is a shared * unwind primitive which handles all stacks as requested, and knows * the proper order for unwinding.) * * Valstack entries above 'top' are always kept initialized to * "undefined unused". Callstack and catchstack entries above 'top' * are not zeroed and are left as garbage. * * Value stack handling is mostly a part of the API implementation.
label: code-design

24157. Step 9: copy elements-to-be-deleted into the result array

24158. 0x30-0x3f

24159. 2 bits for heap type

24160. setjmp/catchpoint setup

24161. **comment:** not sure whether or not needed; Thursday
label: requirement

24162. emergency mode: try extra hard

24163. Set object 'length'.

24164. byte index limit for element access, exclusive

24165. We could rely on max temp/const checks: if they don't exceed BC * limit, nothing here can either (just asserts would be enough). * Currently we check for the limits, which provides additional * protection against creating invalid bytecode due to compiler * bugs.

24166. require initializer for var/const

24167. DUK_TOK_FALSE

24168. fall-through control flow patchup; note that pc_prevstmt may be * < 0 (i.e. no case clauses), in which case this is a no-op.

24169. [... eval "eval" eval_input level]

24170. * ToObject() (E5 Section 9.9) * * ==> implemented in the API.

24171. DUK_USE_MARKANDSWEEP_FINALIZER_TORTURE

24172. **comment:** * Handle integers in 32-bit range (that is, [-(2**32-1),2**32-1]) * specially, as they're very likely for embedded programs. This * is now done for all radix values. We must be careful not to use * the fast path when special formatting (e.g. forced exponential) * is in force. * * XXX: could save space by supporting radix 10 only and using * sprintf "%lu" for the fast path and for exponent formatting.
label: code-design

24173. !(p < r)

24174. Maximum exponent value when parsing numbers. This is not strictly * compliant as there should be no upper limit, but as we parse the * exponent without a bigint, impose some limit.

24175. num_stack_args

24176. **comment:** Executor interrupt counter check, used to implement breakpoints, * debugging interface, execution timeouts, etc. The counter is heap * specific but is maintained in the current thread to make the check * as fast as possible. The counter is copied back to the heap struct * whenever a thread switch occurs by the DUK_HEAP_SWITCH_THREAD() macro.
label: code-design

24177. complete 'sub atom'

24178. * Intern key via the valstack to ensure reachability behaves * properly. We must avoid longjmp's here so use non-checked * primitives. * * Note: duk_check_stack() potentially reallocs the valstack, * invalidating any duk_tval pointers to valstack. Callers * must be careful.

24179. round out to 8 bytes

24180. for convenience

24181. buffer limits

24182. Handle TypedArray vs. Node.js Buffer arg differences

24183. DUK_TOK_WHILE

24184. offset/value order different from Node.js

24185. * Other cases (function declaration, anonymous function expression, * strict direct eval code). The "outer" environment will be whatever * the caller gave us.

24186. * Setup environment record properties based on the template and * its flags. * * If DUK_HOBJECT_HAS_NEWENV(fun_temp) is true, the environment * records represent identifiers "outside" the function; the * "inner" environment records are created on demand. Otherwise, * the environment records are those that will be directly used * (e.g. for declarations). * * _Lexenv is always set; _Varenv defaults to _Lexenv if missing, * so _Varenv is only set if _Lexenv != _Varenv. * * This is relatively complex, see doc/identifier-handling.rst.

24187. * Detach actual buffer from dynamic buffer in valstack, and * pop it from the stack. * * XXX: the buffer object is certainly not reachable at this point, * so it would be nice to free it forcibly even with only * mark-and-sweep enabled. Not a big issue though.

24188. continue matched this label -- we can only continue if this is the empty * label, for which duplication is allowed, and thus there is hope of * finding a match deeper in the label stack.

24189. Plus sign must be accepted for positive exponents * (e.g. '1.5e+2'). This clause catches NULs.

24190. DUK_TOK_VOID

24191. DUK_USE_PC2LINE

24192. DUK_HOBJECT_FLAG_STRICT varies

24193. [... errval errhandler]

24194. **comment:** XXX: Valstack, callstack, and catchstack are currently assumed * to have non-NULL pointers. Relaxing this would not lead to big * benefits (except perhaps for terminated threads).
label: code-design

24195. serialize the wrapper with empty string key

24196. * Free a heap object. * * Free heap object and its internal (non-heap) pointers. Assumes that * caller has removed the object from heap allocated list or the string * intern table, and any weak references (which strings may have) have * been already dealt with.

24197. BITWISE EXPRESSIONS

24198. **comment:** * Copy some internal properties directly * * The properties will be writable and configurable, but not enumerable.
label: code-design

24199. Valstack resize flags

24200. **comment:** XXX: Hex encoded, length limited buffer summary here?
label: code-design

24201. finish up pending jump and split for last alternative

24202. nop

24203. * Local declarations.

24204. stack[0] = object (this) * stack[1] = ToUint32(length) * stack[2] = elem at index 0 (retval)

24205. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers). A prototype loop must not cause * an error to be thrown here; duk_hobject_hasprop_raw() will ignore a * prototype loop silently and indicate that the property doesn't exist.

24206. **comment:** XXX: coerce to regs? it might be better for enumeration use, where the * same PROP ivalue is used multiple times. Or perhaps coerce PROP further * there?
label: code-design

24207. $x > y \rightarrow y < x$

24208. * Found existing accessor property (own or inherited). * Call setter with 'this' set to orig, and value as the only argument. * Setter calls are OK even for ROM objects. * * Note: no exotic arguments object behavior, because [[Put]] never * calls [[DefineOwnProperty]] (E5 Section 8.12.5, step 5.b).

24209. resumee: [... initial_func] (currently actually: [initial_func])

24210. Lookup 'key' from arguments internal 'map', and leave replacement value * on stack top if mapped (and return non-zero). * Used in E5 Section 10.6 algorithm for [[GetOwnProperty]] (used by [[Get]]).

24211. proto can also be NULL here (allowed explicitly)

24212. * split()

24213. Validate that the whole slice [0,length[is contained in the underlying * buffer. Caller must ensure 'buf' != NULL.

24214. could also just pop?

24215. Parser part indices.

24216. Compiler SyntaxError (or other error) gets the primary blame. * Currently no flag to prevent blaming.

24217. We don't check the zero padding bytes here right now * (that they're actually zero). This seems to be common * behavior for base-64 decoders.

24218. An object may be in heap_allocated list with a zero * refcount if it has just been finalized and is waiting * to be collected by the next cycle.

24219. * Local prototypes

24220. low-level property functions

24221. helpers for defineProperty() and defineProperties()

24222. w/o refcounts

24223. Skip dvalues to EOM.

24224. all native functions have NEWENV

24225. +1, right after inserted jump

24226. only accept lowercase 'utf8' now.

24227. **comment:** XXX: duplicates should be eliminated here
label: code-design

24228. **comment:** XXX: block removal API primitive
label: code-design

24229. LOGICAL EXPRESSIONS

24230. %lf, %ld etc

24231. * Switch is pretty complicated because of several conflicting concerns: * * - Want to generate code without an intermediate representation, * i.e., in one go * * - Case selectors are expressions, not values, and may thus e.g. throw * exceptions (which causes evaluation order concerns) * * - Evaluation semantics of case selectors and default clause need to be * carefully implemented to provide correct behavior even with case value * side effects * * - Fall through case and default clauses; avoiding dead JUMPs if case * ends with an unconditional jump (a break or a continue) * * - The same case value may occur multiple times, but evaluation rules * only process the first match before switching to a "propagation" mode * where case values are no longer evaluated * * See E5 Section 12.11. Also see doc/compiler.rst for compilation * discussion.

24232. 8: getFullYear

24233. * Debug connection peek and flush primitives

24234. 'do'

24235. duk_xdef_prop() will define an own property without any array * special behaviors. We'll need to set the array length explicitly * in the end. For arrays with elisions, the compiler will emit an * explicit SETALEN which will update the length.

24236. exponential notation forced

24237. NB: 'val' may be invalidated here because put_value may realloc valstack, * caller beware.

24238. Here we'd have the option to normalize -0 to +0.

24239. **comment:** XXX: share helper from lexer; duk_lexer.c / hexval().
label: code-design

24240. **comment:** * Create an internal enumerator object E, which has its keys ordered * to match desired enumeration ordering. Also initialize internal control * properties for enumeration. * * Note: if an array was used to hold enumeration keys instead, an array * scan would be needed to eliminate duplicates found in the prototype chain.
label: code-design

24241. **comment:** This is essentially the 'scale' algorithm, with recursion removed. * Note that 'k' is either correct immediately, or will move in one * direction in the loop. There's no need to do the low/high checks * on every round (like the Scheme algorithm does). * * The scheme algorithm finds 'k' and updates 's' simultaneously, * while the logical algorithm finds 'k' with 's' having its initial * value, after which 's' is updated separately (see the Burger-Dybvig * paper, Section 3.1, steps 2 and 3). * * The case where m+ == m- (almost always) is optimized for, because * it reduces the bigint operations considerably and almost always * applies. The scale loop only needs to work with m+, so this works.
label: code-design

24242. A -> flags * BC -> regCatch; base register for two registers used both during * trycatch setup and when catch is triggered * * If DUK_BC_TRYCATCH_FLAG_CATCH_BINDING set: * regCatch + 0: catch binding variable name (string). * Automatic declarative environment is established for * the duration of the 'catch' clause. * * If DUK_BC_TRYCATCH_FLAG_WITH_BINDING set: * regCatch + 0: with 'target value', which is coerced to * an object and then used as a bindind object for an * environment record. The binding is initialized here, for * the 'try' clause. * * Note that a TRYCATCH generated for a 'with' statement has no * catch or finally parts.

24243. default: false

24244. **comment:** fill new_h with u32 0xff = UNUSED
label: code-design

24245. A non-extensible object cannot gain any more properties, * so this is a good time to compact.

24246. if direct eval, calling activation must exist

24247. **comment:** XXX: because of the final check below (that the literal is not * followed by a digit), this could maybe be simplified, if we bail * out early from a leading zero (and if there are no periods etc). * Maybe too complex.
label: code-design

24248. * Create escaped RegExp source (E5 Section 15.10.3). * * The current approach is to special case the empty RegExp * (" -> '(?:')") and otherwise replace unescaped '/' characters * with '\/' regardless of where they occur in the regexp. * * Note that normalization does not seem to be necessary for * RegExp literals (e.g. '/foo/') because to be acceptable as * a RegExp literal, the text between forward slashes must * already match the escaping requirements (e.g. must not contain * unescaped forward slashes or be empty). Escaping IS needed * for expressions like 'new RegExp("...","")' however. * Currently, we re-escape in either case. * * Also note that we process the source here in UTF-8 encoded * form. This is correct, because any non-ASCII characters are * passed through without change.

24249. Value stack is used to ensure reachability of constants and * inner functions being loaded. Require enough space to handle * large functions correctly.

24250. **comment:** Automatic defaulting of logger name from caller. This would * work poorly with tail calls, but constructor calls are currently * never tail calls, so tail calls are not an issue now.
label: code-design

24251. Fast check for extending array: check whether or not a slow density check is required.

24252. **comment:** XXX: using duk_put_prop_index() would cause obscure error cases when Array.prototype * has write-protected array index named properties. This was seen as DoubleErrors * in e.g. some test262 test cases. Using duk_xdef_prop_index() is better and heavier. * The best fix is to fill in the tracedata directly into the array part. There are * no side effect concerns if the array part is allocated directly and only INCREFs * happen after that.
label: code-design

24253. filename may be NULL in which case file/line is not recorded

24254. unsigned

24255. **comment:** XXX: function flag to make this automatic?
label: requirement

24256. Must have array part and no conflicting exotic behaviors. * Doesn't need to have array special behavior, e.g. Arguments * object has array part.

24257. Breakpoint entries above the used area are left as garbage.

24258. allow_source_elem

24259. JSON parsing code is allowed to read [p_start,p_end]: p_end is * valid and points to the string NUL terminator (which is always * guaranteed for duk_hstrings).
24260. fall through if overflow etc
24261. inner function numbering
24262. **comment:** XXX: could write output in chunks with fewer ensure calls, * but relative benefit would be small here.
 label: code-design
24263. Assumes that saved[0] and saved[1] are always * set by regexp bytecode (if not, char_end_offset * will be zero). Also assumes clen reflects the * correct char length.
24264. **comment:** XXX: V8 throws a TypeError for negative values. Would it * be more useful to interpret negative offsets here from the * end of the buffer too?
 label: code-design
24265. function needs shuffle registers
24266. The compiler should never emit DUK_OP_REGEXP if there is no * regexp support.
24267. invalid padding
24268. Note: here we must be wary of the fact that a pointer may be * valid and be a NULL.
24269. **comment:** Here we can choose either to end parsing and ignore * whatever follows, or to continue parsing in case * multiple (possibly padded) base64 strings have been * concatenated. Currently, keep on parsing.
 label: code-design
24270. [requested_id require require.id resolved_id last_comp]
24271. advance to get one step of lookup
24272. Write signed 32-bit integer
24273. * Canonicalize() abstract operation needed for canonicalization of individual * codepoints during regexp compilation and execution, see E5 Section 15.10.2.8. *
 Note that codepoints are canonicalized one character at a time, so no context * specific rules can apply. Locale specific rules can apply, though.
24274. This should not be necessary because no-one should tamper with the * regexp bytecode, but is prudent to avoid potential segfaults if that * were to happen for some reason.
24275. Needs lookahead
24276. * Churn refzero_list until empty
24277. Note: 'q_instr' is still used below
24278. duk_unicode_caseconv_uc[]
24279. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. * * XXX: this is now an overkill for many fast paths. Rework this * to be faster (although switching to a valstack discipline might * be a better solution overall).
 label: code-design
24280. failed, resize and try again
24281. extend with undefined
24282. **comment:** * Traceback handling * * The unified helper decodes the traceback and produces various requested * outputs. It should be optimized for size, and may leave garbage on stack, * only the topmost return value matters. For instance, traceback separator * and decoded strings are pushed even when looking for filename only. * * NOTE: although _Tracedata is an internal property, user code can currently * write to the array (or replace it with something other than an array). * The code below must tolerate arbitrary _Tracedata. It can throw errors * etc, but cannot cause a segfault or memory unsafe behavior.
 label: code-design
24283. Error; error value is in heap->jl.value1.
24284. -> [... key val toJSON val key]
24285. callstack
24286. read 3 bytes into 't', padded by zero
24287. XXX: remove this feature entirely? it would only matter for * emergency GC. Disable for lowest memory builds.
24288. * Computed values (e.g. INFINITY)
24289. DUK_TOK_DIV_EQ
24290. already set
24291. input 'd' in Windows UTC, 100ns units
24292. Keep current size
24293. 50x heap size
24294. Dynamic buffer with 'curr_alloc' pointing to a dynamic area allocated using * heap allocation primitives. Also used for external buffers when low memory * options are not used.
24295. [name this]
24296. [... holder name val enum obj_key new_elem]
24297. patched later
24298. [... regexp_object]
24299. **comment:** * Note: the description in E5 Section 15.10.2.6 has a typo, it * contains 'A' twice and lacks 'a'; the intent is [0-9a-zA-Z_].
 label: documentation
24300. This was the last term, and q_last was * updated to match this term at loop top.
24301. x < b/y; use t1 and t2 as temps
24302. prev_token.start_offset points to the closing brace here; when skipping * we're going to reparse the closing brace to ensure semicolon insertion * etc work as expected.
24303. more initializers
24304. ref.holder is global object, holder is the object with the * conflicting property.
24305. essentially tracks digit position of lowest 'f' digit
24306. **comment:** Prevent any side effects on the string table and the caller provided * str/blen arguments while interning is in progress. For example, if * the caller provided str/blen from a dynamic buffer, a finalizer might * resize that dynamic buffer, invalidating the call arguments.
 label: code-design
24307. binding power "levels" (see doc/compiler.rst)
24308. Enter message processing loop for sending Status notifs and * to finish a pending detach.
24309. result reg
24310. strings are only tracked by stringtable
24311. to avoid relookups
24312. abandoning requires a props allocation resize and * 'rechecks' the valstack, invalidating any existing * valstack value pointers!
24313. If object has a .toJSON() property, we can't be certain * that it wouldn't mutate any value arbitrarily, so bail * out of the fast path. * * If an object is a Proxy we also can't avoid side effects * so abandon.
24314. ':'
24315. **comment:** Unlike in duk_hex_encode() 'dst' is not necessarily aligned by 2. * For platforms where unaligned accesses are not allowed, shift 'dst' * ahead by 1 byte to get alignment and then DUK_MEMMOVE() the result * in place. The faster encoding loop makes up the difference. * There's always space for one extra byte because a terminator always * follows the hex data and that's been accounted for by the caller.
 label: code-design
24316. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
24317. **comment:** * 'nregs' registers are allocated on function entry, at most 'nargs' * are initialized to arguments, and the rest to undefined. Arguments * above 'nregs' are not mapped to registers. All registers in the * active stack range must be initialized because they are GC reachable. * 'nargs' is needed so that if the function is given more than 'nargs' * arguments, the additional arguments do not 'clobber' registers * beyond 'nregs' which must be consistently initialized to undefined. * * Usually there is no need to know which registers are mapped to * local variables. Registers may be allocated to variable in any * way (even including gaps). However, a register-variable mapping * must be the same for the duration of the function execution and * the register cannot be used for anything else. * * When looking up variables by name, the '_Varmap' map is used. * When an activation closes, registers mapped to arguments are * copied into the environment record based on the same map. The * reverse map (from register to variable) is not currently needed * at run time, except for debugging, so it is not maintained.

label: code-design

24318. Slice starting point is beyond current length.

24319. ... target is extensible

24320. duk_unicode_support.c

24321. **comment:** XXX: easier check with less code?**label:** code-design

24322. Used when precision is undefined; also used for NaN (-> "NaN"), * and +/- infinity (-> "Infinity", "-Infinity").

24323. x <- x * y, use t as temp

24324. [... v1 v2 str] -> [... str v2]

24325. Zero 'count' is also allowed to make call sites easier.

24326. **comment:** XXX: size optimize**label:** code-design

24327. != 0 => post-update for array 'length' (used when key is an array index)

24328. **comment:** has inner functions which may slow access (XXX: this can be optimized by looking at the inner functions)**label:** code-design24329. **comment:** XXX: need to determine LHS type, and check that it is LHS compatible**label:** code-design

24330. false

24331. * Built-in strings

24332. * Clear (reachable) flags of finalize_list * * We could mostly do in the sweep phase when we move objects from the * heap into the finalize_list. However, if a finalizer run is skipped * during a mark-and-sweep, the objects on the finalize_list will be marked * reachable during the next mark-and-sweep. Since they're already on the * finalize_list, no-one will be clearing their REACHABLE flag so we do it * here. (This now overlaps with the sweep handling in a harmless way.)

24333. * Variant 3

24334. * On second pass, skip the function.

24335. Note: key issue here is to re-lookup the base pointer on every attempt. * The pointer being reallocated may change after every mark-and-sweep.

24336. -> [... key val toJSON val]

24337. **comment:** XXX: It would be nice to build the string directly but ToUint16() * coercion is needed so a generic helper would not be very * helpful (perhaps coerce the value stack first here and then * build a string from a duk_tval number sequence in one go?).**label:** code-design

24338. Note: getprop may invoke any getter and invalidate any * duk_tval pointers, so this must be done first.

24339. [obj trap_result res_arr proppname]

24340. split1: prefer direct execution (no jump)

24341. in resumer's context

24342. Conceptually we'd extract the plain underlying buffer * or its slice and then do a type mask check below to * see if we should reject it. Do the mask check here * instead to avoid making a copy of the buffer slice.

24343. 'try'

24344. -> [in_val]

24345. **comment:** Currently all built-in native functions are strict. * This doesn't matter for many functions, but e.g. * String.prototype.charAt (and other string functions) * rely on being strict so that their 'this' binding is * not automatically coerced.**label:** code-design24346. **comment:** * Helper for handling an Ecmascript-to-Ecmascript call or an Ecmascript * function (initial) Duktape.Thread.resume(). * * Compared to normal calls handled by duk_handle_call(), there are a * bunch of differences: * * - the call is never protected * - there is no C recursion depth increase (hence an "ignore recursion * limit" flag is not applicable) * - instead of making the call, this helper just performs the thread * setup and returns; the bytecode executor then restarts execution * internally * - ecmascript functions are never 'vararg' functions (they access * varargs through the 'arguments' object) * * The callstack of the target contains an earlier Ecmascript call in case * of an Ecmascript-to-Ecmascript call (whose idx_retval is updated), or * is empty in case of an initial Duktape.Thread.resume(). * * The first thing to do here is to figure out whether an ecma-to-ecma * call is actually possible. It's not always the case if the target is * a bound function; the final function may be native. In that case, * return an error so caller can fall back to a normal call path.**label:** code-design

24347. Note: 0xff != DUK_BC_B_MAX

24348. * Unions and structs for self tests

24349. Note: masking is done for 'i' to deal with negative numbers correctly

24350. * For property layout 1, tweak e_size to ensure that the whole entry * part (key + val + flags) is a suitable multiple for alignment * (platform specific). * * Property layout 2 does not require this tweaking and is preferred * on low RAM platforms requiring alignment.

24351. don't run finalizers; queue finalizable objects back to heap_allocated

24352. val1 = count

24353. * Function declaration, function expression, or (non-standard) * function statement. * * The E5 specification only allows function declarations at * the top level (in "source elements"). An ExpressionStatement * is explicitly not allowed to begin with a "function" keyword * (E5 Section 12.4). Hence any non-error semantics for such * non-top-level statements are non-standard. Duktape semantics * for function statements are modelled after V8, see * test-dev-func-decl-outside-top.js.

24354. Get minimum array part growth for a certain size.

24355. **comment:** XXX: very slow - better to bulk allocate a gap, and copy * from args_array directly (we know it has a compact array * part, etc).**label:** code-design

24356. 'enumerate'

24357. DUK_TOK_RSHIFT

24358. roughly 256 bytes

24359. **comment:** For now, just use duk_safe_call() to wrap duk_new(). We can't * simply use a protected duk_handle_call() because there's post * processing which might throw. It should be possible to ensure * the post processing never throws (except in internal errors and * out of memory etc which are always allowed) and then remove this * wrapper.**label:** code-design

24360. object is now reachable

24361. flags: "im"

24362. Copy trap result keys into the enumerator object.

24363. * Mark-and-sweep flags * * These are separate from heap level flags now but could be merged. * The heap structure only contains a 'base mark-and-sweep flags' * field and the GC caller can impose further flags.

24364. Success path.

24365. * If entering a 'catch' block which requires an automatic * catch variable binding, create the lexical environment. * * The binding is mutable (= writable) but not deletable. * Step 4 for the catch production in E5 Section 12.14; * no value is given for CreateMutableBinding 'D' argument, * which implies the binding is not deletable.

24366. t1 <- (* r B)

24367. DUK_BUFOBJ_UINT8ARRAY

24368. Accept any duk_hbufferobject, though we're only normally * called for Duktape.Buffer values.

24369. Lenient: allow function declarations outside top level in * non-strict mode but reject them in strict mode.

24370. assertions: env must be closed in the same thread as where it runs

24371. EOF (-1) will be cast to an unsigned value first * and then re-cast for the switch. In any case, it * will match the default case (syntax error).

24372. refzero not running -> must be empty

24373. **comment:** XXX: differentiate null pointer from empty string?**label:** code-design

24374. Normal object which doesn't get automatically coerced to a * primitive value. Functions are checked for specially. The * primitive value coercions for Number, String, Pointer, and * Boolean can't result in functions so suffices to check here.

24375. DUK_USE_MARK_AND_SWEEP
24376. * Coercion and fast path processing
24377. y == +Infinity
24378. caller required to know
24379. * Function name (if any) ** We don't check for prohibited names here, because we don't yet know whether the function will be strict. Function body * parsing handles this retroactively. ** For function expressions and declarations function name must * be an Identifier (excludes reserved words). For setter/getter * it is a PropertyName which allows reserved words and also * strings and numbers (e.g. "{ get 1() { ... } }").
24380. **comment:** XXX: There are no format strings in duk_config.h yet, could add * one for formatting duk_int64_t. For now, assumes "%lld" and that * "long long" type exists. Could also rely on C99 directly but that * won't work for older MSVC.
label: code-design
24381. 'toLocaleString'
24382. * Manipulate value stack so that exactly 'num_stack_rets' return * values are at 'idx_retbase' in every case, assuming there are * 'rc' return values on top of stack. * This is a bit tricky, because the called C function operates in * the same activation record and may have e.g. popped the stack * empty (below idx_retbase).
24383. Exotic behaviors are only enabled for arguments objects * which have a parameter map (see E5 Section 10.6 main * algorithm, step 12). * In particular, a non-strict arguments object with no * mapped formals does *NOT* get exotic behavior, even * for e.g. "caller" property. This seems counterintuitive * but seems to be the case.
24384. buffer is automatically zeroed
24385. num_args
24386. DUK_OP_EXTRA, sub-operation in A
24387. insert the required matches (qmin) by copying the atom
24388. DUK_USE_INTEGER_BE
24389. out_clamped
24390. adv = 2 default OK
24391. There is no need to decref anything else than 'env': if 'env' * becomes unreachable, refzero will handle decref'ing its prototype.
24392. '-0'
24393. module table remains registered to modLoaded, minimize its size
24394. * Detach handling
24395. write back
24396. [... filename (temps) func]
24397. **comment:** It's not strictly necessary to track the current size, but * it is useful for writing robust native code.
label: code-design
24398. * Second pass parsing.
24399. **comment:** Note: cannot portably debug print a function pointer, hence 'func' not printed!
label: code-design
24400. * Arbitrary byteswap for potentially unaligned values ** Used to byteswap pointers e.g. in debugger code.
24401. **comment:** XXX: There's some overlap with duk_js_closure() here, but * seems difficult to share code. Ensure that the final function * looks the same as created by duk_js_closure().
label: code-design
24402. **comment:** It is important not to call this if the last byte read was an EOM. * Reading ahead in this scenario would cause unnecessary blocking if * another message is not available.
label: code-design
24403. DUK_USE_EXEC_TIMEOUT_CHECK
24404. * Misc defines
24405. E5.1 Section 15.1.3.4: uriUnescaped
24406. Combined size of separators already overflows
24407. -> [... res]
24408. [... this name message]
24409. "/" and not in regexp mode
24410. DUK_USE_BYTECODE_DUMP_SUPPORT
24411. **comment:** may indirectly slow access through a direct eval
label: code-design
24412. no incref needed
24413. variable binding register if register-bound (otherwise < 0)
24414. How much to advance before next loop; note that next loop * will advance by 1 anyway, so -1 from the total escape * length (e.g. len("\uXXXXX") - 1 = 6 - 1). As a default, * 1 is good.
24415. **comment:** 'sce' points to the wrong entry here, but is no longer used
label: code-design
24416. Note: this check relies on the fact that even a zero-size string * has a non-NUL pointer.
24417. [... varname val this] (because throw_flag == 1, always resolved)
24418. No need to reinit setjmp() catchpoint, as call handling * will store and restore our state.
24419. embed: duk_hbuffer ptr
24420. t >= 0 always true, unsigned
24421. Maximum expansion per input byte is 6: * - invalid UTF-8 byte causes "\uXXXX" to be emitted (6/1 = 6). * - 2-byte UTF-8 encodes as "\uXXXX" (6/2 = 3). * - 4-byte UTF-8 encodes as "\Uxxxxxxxx" (10/4 = 2.5).
24422. Shared helper to implement Object.getPrototypeOf and the ES6 * Object.prototype.__proto__ getter. ** http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype._proto_
24423. xxx -> DUK_HBUFFEROBJECT_ELEM_INT16
24424. vararg function, careful arg handling (e.g. thisArg may not be present)
24425. implicit this value always undefined for * declarative environment records.
24426. **comment:** * IsCallable() (E5 Section 9.11) ** XXX: API equivalent is a separate implementation now, and this has * currently no callers.
label: code-design
24427. Extract 'top' bits of curval; note that the extracted bits do not need * to be cleared, we just ignore them on next round.
24428. duk_unicode_ids_noabmp[]
24429. Return value of 'sz' or more indicates output was (potentially) * truncated.
24430. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers).
24431. Encoding/decoding flags
24432. Shared helper to implement ES6 Object.setPrototypeOf and * Object.prototype.__proto__ setter. ** http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype._proto_ * <http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.setprototypeof>
24433. no need to write 'out_bufdata'
24434. new_used / size <= 1 / DIV <=> new_used <= size / DIV
24435. Register the module table early to modLoaded[] so that we can * support circular references even in modSearch(). If an error * is thrown, we'll delete the reference.
24436. label handling
24437. Expression
24438. **comment:** This is performance critical because it appears in every DECREF.
label: code-design
24439. code_idx = entry_top + 0
24440. -> [... lval rval]

24441. * Standard algorithm succeeded without errors, check for exotic post-behaviors. ** Arguments exotic behavior in E5 Section 10.6 occurs after the standard * [[DefineOwnProperty]] has completed successfully. ** Array exotic behavior in E5 Section 15.4.5.1 is implemented partly * prior to the default [[DefineOwnProperty]], but: * - for an array index key (e.g. "10") the final 'length' update occurs here * - for 'length' key the element deletion and 'length' update occurs here

24442. * All defined array-indexed properties are in the array part * (we assume the array part is comprehensive), and all array * entries are writable, configurable, and enumerable. Thus, * nothing can prevent array entries from being deleted.

24443. **comment:** * Helpers for creating and querying pc2line debug data, which * converts a bytecode program counter to a source line number. ** The run-time pc2line data is bit-packed, and documented in: * * doc/function-objects.rst

label: documentation

24444. Represent a null pointer as 'null' to be consistent with * the JX format variant. Native '%p' format for a NULL * pointer may be e.g. '(nil)'.

24445. eat 'throw'

24446. always grow the array, no sparse / abandon support here

24447. leave formals on stack for later use

24448. since index non-negative

24449. tzoffset

24450. duk_has_prop() popped the second 'name'

24451. varname

24452. * E5 Section 15.10.2.6 "IsWordChar" abstract operation. Assume * $x < 0$ for characters read outside the string.

24453. -----XXXX

24454. stack top: new time value, return 1 to allow tail calls

24455. basereg

24456. E5 Section 15.4.5.1, step 3, steps a - i are implemented here, j - n at the end

24457. jump for 'catch' case

24458. Module object containing module.exports, etc

24459. 'prototype' property for all built-in objects (which have it) has attributes: * [[Writable]] = false, * [[Enumerable]] = false, * [[Configurable]] = false

24460. stack[0] = start * stack[1] = deleteCount * stack[2...nargs-1] = items * stack[nargs] = ToObject(this) -3 * stack[nargs+1] = ToUint32(length) -2 * stack[nargs+2] = result array -1

24461. -> [... escaped_source bytecode]

24462. * Write to array part? ** Note: array abandoning requires a property resize which uses * 'rechecks' valstack for temporaries and may cause any existing * valstack pointers to be invalidated. To protect against this, * tv_obj, tv_key, and tv_val are copies of the original inputs.

24463. copy existing entries as is

24464. 5: toLocaleTimeString

24465. mark-and-sweep: children not processed

24466. Catches EOF of JSON input.

24467. radix

24468. **comment:** XXX: when optimizing for guaranteed property slots, use a guaranteed * slot for internal value; this call can then access it directly.

label: code-design

24469. no wrapping

24470. **comment:** * Create and throw an error (originating from Duktape internally) ** Push an error object on top of the stack, possibly throw augmenting * the error, and finally longjmp. ** If an error occurs while we're dealing with the current error, we might * enter an infinite recursion loop. This is prevented by detecting a * "double fault" through the heap->handling_error flag; the recursion * then stops at the second level.

label: code-design

24471. if property key begins with underscore, encode it with * forced quotes (e.g. "_Foo") to distinguish it from encoded * internal properties (e.g. \xffBar -> _Bar).

24472. object or array

24473. no need to check now: both success and error are OK

24474. Note: allow backtracking from p == ptr_end

24475. Relookup if relocated

24476. module.filename for .fileName, default to resolved ID if * not present.

24477. -> [... toLocaleString ToObject(val)]

24478. DUK_TOK_LBRACKET already eaten, current token is right after that

24479. **comment:** * Array needs to grow, but we don't want it becoming too sparse. * If it were to become sparse, abandon array part, moving all * array entries into the entries part (for good). ** Since we don't keep track of actual density (used vs. size) of * the array part, we need to estimate somehow. The check is made * in two parts: * * - Check whether the resize need is small compared to the * current size (relatively); if so, resize without further * checking (essentially we assume that the original part is * "dense" so that the result would be dense enough). * * - Otherwise, compute the resize using an actual density * measurement based on counting the used array entries.

label: code-design

24480. Restore stack top after unbalanced code paths.

24481. position of highest digit changed

24482. * duk_heap allocation and freeing.

24483. Careful with borrow condition: * - If borrow not subtracted: 0x12345678 - 0 - 0xffffffff = 0x12345679 (> 0x12345678) * - If borrow subtracted: 0x12345678 - 1 - 0xffffffff = 0x12345678 (== 0x12345678)

24484. no statement value (unlike function expression)

24485. Three digit octal escape, digits validated.

24486. **comment:** * Addition operator is different from other arithmetic * operations in that it also provides string concatenation. * Hence it is implemented separately. ** There is a fast path for number addition. Other cases go * through potentially multiple coercions as described in the * E5 specification. It may be possible to reduce the number * of coercions, but this must be done carefully to preserve * the exact semantics. ** E5 Section 11.6.1. ** Custom types also have special behavior implemented here.

label: code-design

24487. Quite approximate but should be useful for little and big endian.

24488. For X <op>= Y we need to evaluate the pre-op * value of X before evaluating the RHS: the RHS * can change X, but when we do <op> we must use * the pre-op value.

24489. Count is incremented before DUK_DRAGON4_OUTPUT_PREINC() call * on purpose, which is taken into account by the macro.

24490. NUL also comes here. Comparison order matters, 0x20 * is most common whitespace.

24491. retval for success path

24492. * Wrapping duk_safe_call() will mangle the stack, just return stack top

24493. * Node.js Buffer.prototype: toJSON()

24494. We'll need to interrupt early so recompute the init * counter to reflect the number of bytecode instructions * executed so that step counts for e.g. debugger rate * limiting are accurate.

24495. 'fileName'

24496. * Convert duk_compiler_func to a function template and add it * to the parent function table.

24497. parse args starting from "next temp"

24498. This seems like a good overall approach. Fast path for ASCII in 4 byte * blocks.

24499. * Mark-and-sweep garbage collection.

24500. **comment:** * Object compaction (emergency only). ** Object compaction is a separate step after sweeping, as there is * more free memory for it to work with. Also, currently compaction * may insert new objects into the heap allocated list and the string * table which we don't want to do during a sweep (the reachability * flags of such objects would be incorrect). The objects inserted * are currently: * * - a temporary duk_hbuffer for a new properties allocation * - if array part is abandoned, string keys are interned * * The object insertions go to the front of the list, so they do not * cause an infinite loop (they are not compacted).

label: code-design

24501. **comment:** unnecessary div for last byte
label: code-design

24502. buffer values are coerced first to string here

24503. sanity limit for function pointer size

24504. [sep ToObject(this) len sep str0 ... str(count-1)]

24505. DUK_TOK_YIELD

24506. **comment:** XXX: is valstack top best place for argument?
label: code-design

24507. reset bytecode buffer but keep current size; pass 2 will * require same amount or more.

24508. duk_hobject_run_finalizer() sets

24509. [source template closure]

24510. Successful return: restore jmpbuf and return to caller.

24511. negative -> no atom matched on previous round

24512. [... name]

24513. Note: may be NULL if first call

24514. unresolvable, no stack changes

24515. **comment:** The DataView .buffer property is ordinarily set to the argument * which is an ArrayBuffer. We accept any duk_hbufferobject as * an argument and .buffer will be set to the argument regardless * of what it is. This may be a bit confusing if the argument * is e.g. a DataView or another TypedArray view. ** XXX: Copy .buffer property from a DataView/TypedArray argument? * Create a fresh ArrayBuffer for Duktape.Buffer and Node.js Buffer * arguments? See: test-bug-dataview-buffer-prop.js.
label: code-design

24516. DUK_USE_DATE_NOW_TIME

24517. XXX: valstack handling is awkward. Add a valstack helper which * avoids dup():ing; valstack_copy(src, dst)?

24518. On entry, item at idx_func is a bound, non-lightweight function, * but we don't rely on that below.

24519. * Reference counting implementation.

24520. use strict equality instead of non-strict equality

24521. This assert depends on the input parts representing time inside * the Ecmascript range.

24522. controls for minimum array part growth

24523. -> [... key]

24524. * Returns non-zero if a key and/or value was enumerated, and: * * [enum] -> [key] (get_value == 0) * [enum] -> [key value] (get_value == 1) * * Returns zero without pushing anything on the stack otherwise.

24525. allow basic ASCII whitespace

24526. Heap allocated: return heap pointer which is NOT useful * for the caller, except for debugging.

24527. activation is a direct eval call

24528. current lex_env of the activation (created for catcher)

24529. **comment:** Figure out all active breakpoints. A breakpoint is * considered active if the current function's fileName * matches the breakpoint's fileName, AND there is no * inner function that has matching line numbers * (otherwise a breakpoint would be triggered both * inside and outside of the inner function which would * be confusing). Example: * * function foo() { * print('foo'); * function bar() { <-, breakpoints in these * print('bar'); | lines should not affect * } <- 'foo()' execution * bar(); * } * * We need a few things that are only available when * debugger support is enabled: (1) a line range for * each function, and (2) access to the function * template to access the inner functions (and their * line ranges). * * It's important to have a narrow match for active * breakpoints so that we don't enter checked execution * when that's not necessary. For instance, if we're * running inside a certain function and there's * breakpoint outside in (after the call site), we * don't want to slow down execution of the function.
label: code-design

24530. start byte offset of token in lexer input

24531. * Note: lj.value1 is 'value', lj.value2 is 'resume'. * This differs from YIELD.

24532. curr_token = get/set name

24533. Initial '|' has been checked and eaten by caller.

24534. Thread may have changed, e.g. YIELD converted to THROW.

24535. Lookup to encode one byte directly into 2 characters: * * def genhextab(bswap): * for i in xrange(256): * t = chr(i).encode('hex') * if bswap: * t = t[1] + t[0] * print('0x' + t.encode('hex') + 'U') * print('big endian'); genhextab(False) * print('little endian'); genhextab(True)

24536. * 64-bit arithmetic * * There are some platforms/compilers where 64-bit types are available * but don't work correctly. Test for known cases.

24537. be paranoid for 32-bit time values (even avoiding negative ones)

24538. * Node.js Buffer.byteLength()

24539. emit instruction; could return PC but that's not needed in the majority * of cases.

24540. Fast write calls which assume you control the spare beforehand. * Multibyte write variants exist and use a temporary write pointer * because byte writes alias with anything: with a stored pointer * explicit pointer load/stores get generated (e.g. gcc -Os).

24541. All the flags fit in 16 bits, so will fit into duk_bool_t.

24542. flags: "g"

24543. For cross-checking during development: ensure dispatch count * matches cumulative interrupt counter init value sums.

24544. valstack should not need changes

24545. end of _Varmap

24546. **comment:** Certain boxed types are required to go through * automatic unboxing. Rely on internal value being * sane (to avoid infinite recursion).
label: code-design

24547. [-0x80000000,0x7fffffff]

24548. DUK_HOBJECT_FLAG_EXOTIC_STRINGOBJ varies

24549. 'Arguments'

24550. [key getter this] -> [key retval]

24551. Retry allocation after mark-and-sweep for this * many times. A single mark-and-sweep round is * not guaranteed to free all unreferenced memory * because of finalization (in fact, ANY number of * rounds is strictly not enough).

24552. only functions can have arguments

24553. E5 Section 9.3.1

24554. * GETVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is GetValue'd] * 8.7.1 GetValue (V) * 8.12.1 [[GetOwnProperty]] (P) * 8.12.2 [[GetProperty]] (P) * 8.12.3 [[Get]] (P) * * If 'throw' is true, always leaves two values on top of stack: [val this]. * * If 'throw' is false, returns 0 if identifier cannot be resolved, and the * stack will be unaffected in this case. If identifier is resolved, returns * 1 and leaves [val this] on top of stack. * * Note: the 'strict' flag of a reference returned by GetIdentifierReference * is ignored by GetValue. Hence we don't take a 'strict' parameter. * * The 'throw' flag is needed for implementing 'typeof' for an unreferenced * identifier. An unreferenced identifier in other contexts generates a * ReferenceError.

24555. **comment:** XXX: top of stack must contain value, which helper doesn't touch, * rework to use tv_val directly?
label: code-design

24556. * Executor interrupt handling * * The handler is called whenever the interrupt countdown reaches zero * (or below). The handler must perform whatever checks are activated, * e.g. check for cumulative step count to impose an execution step * limit or check for breakpoints or other debugger interaction. * * When the actions are done, the handler must reinvoke the interrupt * init and counter values. The 'init' value must indicate how many * bytecode instructions are executed before the next interrupt. The * counter must interface with the bytecode executor loop. Concretely, * the new init value is normally one higher than the new counter value. * For instance, to execute exactly one bytecode instruction the init * value is set to 1 and the counter to 0. If an error is thrown by the * interrupt handler, the counters are set to the same value (e.g. both * to 0 to cause an interrupt when the next bytecode instruction is about * to be executed after error handling). * * Maintaining the init/counter value properly is important for accurate * behavior. For instance, executor step limit needs a cumulative step * count which is simply computed as a sum of 'init' values. This must * work accurately even when single stepping.

24557. 0xd0...0xdf

24558. tv1 is -below- valstack_bottom
24559. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT32
24560. [obj Properties]
24561. * First attempt
24562. not found
24563. Loop check.
24564. XXX: just assert non-NULL values here and make caller arguments * do the defaulting to the default implementations (smaller code)?
24565. reserved words: additional future reserved words in strict mode
24566. * String table resize check. * * Note: this may silently (and safely) fail if GC is caused by an * allocation call in stringtable resize_hash(). Resize_hash() * will prevent a recursive call to itself by setting the * DUK_MS_FLAG_NO_STRINGTABLE_RESIZE in heap->mark_and_sweep_base_flags.
24567. Sync and NULL early.
24568. 12 to 21
24569. **comment:** XXX: we should never shrink here; when we error out later, we'd * need to potentially grow the value stack in error unwind which could * cause another error.
label: code-design
24570. -> pushes fixed buffer
24571. **comment:** XXX: improve check; check against nregs, not against top
label: code-design
24572. byte sizes will match
24573. max # of values initialized in one MPUTARR set
24574. not reached
24575. -> [... this]
24576. * Local defines and forward declarations.
24577. **comment:** prev value must be unused, no decref
label: code-design
24578. Calling duk_debugger_cooperate() while Duktape is being * called into is not supported. This is not a 100% check * but prevents any damage in most cases.
24579. Note: careful with indices like '-x'; if 'x' is zero, it refers to bottom
24580. x >= y --> not (x < y)
24581. yes, must set array length explicitly
24582. * Post resize assertions.
24583. Note: for dstlen=0, dst may be NULL
24584. !DUK_USE_TRACEBACKS
24585. -> [func funcname env funcname]
24586. Constants for duk_lexer_ctx.buf.
24587. Difference is in 100ns units, convert to milliseconds w/o fractions
24588. * "IdentifierPart" production check.
24589. prev exists and is not a letter
24590. Note: no recursion issue, we can trust 'map' to behave
24591. if fails, e_size will be zero = not an issue, except performance-wise
24592. **comment:** XXX: use a NULL error handler for the finalizer call?
label: code-design
24593. **comment:** workaround: max nbits = 24 now
label: code-design
24594. DUK_USE_HEX_FASTPATH
24595. DUK_UTIL_H_INCLUDED
24596. 'Infinity'
24597. We know _Varmap only has own properties so walk property * table directly. We also know _Varmap is dense and all * values are numbers; assert for these. GC and finalizers * shouldn't affect _Varmap so side effects should be fine.
24598. debugger state, only relevant when attached
24599. highest value of temp_reg (temp_max - 1 is highest used reg)
24600. * Init compiler and lexer contexts
24601. Input values are signed 48-bit so we can detect overflow * reliably from high bits or just a comparison.
24602. pre/post opcode values have constraints,
24603. allocated heap objects
24604. re-read to avoid spill / fetch
24605. LRU: move our entry to first
24606. argument/function declaration shadows 'arguments'
24607. Proxy target
24608. unconditional block
24609. **comment:** XXX: Using a string return value forces a string intern which is * not always necessary. As a rough performance measure, hex encode * time for tests/perf/test-hex-encode.js dropped from ~35s to ~15s * without string coercion. Change to returning a buffer and let the * caller coerce to string if necessary?
label: code-design
24610. * Node.js Buffer.prototype.copy()
24611. DUK_VERBOSE_ERRORS
24612. skip '"'
24613. XXX: additional conditions when to close variables? we don't want to do it * unless the environment may have "escaped" (referenced in a function closure). * With delayed environments, the existence is probably good enough of a check.
24614. When debugger is enabled, we need to recheck the activation * status after returning. This is now handled by call handling * and heap->dbg_force_restart.
24615. 'byteLength'
24616. * Parsing implementation. * * JSON lexer is now separate from duk_lexer.c because there are numerous * small differences making it difficult to share the lexer. * * The parser here works with raw bytes directly; this works because all * JSON delimiters are ASCII characters. Invalid xUTF-8 encoded values * inside strings will be passed on without normalization; this is not a * compliance concern because compliant inputs will always be valid * CESU-8 encodings.
24617. * Detect iteration statements; if encountered, establish an * empty label.
24618. skip ')'
24619. * Poppers
24620. DUK_HOBJECT_H_INCLUDED
24621. always at least the header
24622. A -> target register * B -> bytecode (also contains flags) * C -> escaped source
24623. Fast path, decode as is.
24624. * "/" followed by something in regexp mode. See E5 Section 7.8.5. * * RegExp parsing is a bit complex. First, the regexp body is delimited * by forward slashes, but the body may also contain forward slashes as * part of an escape sequence or inside a character class (delimited by * square brackets). A mini state machine is used to implement these. * * Further, an early (parse time) error must be thrown if the regexp * would cause a run-time error when used in the expression new RegExp(...). * Parsing here simply extracts the (candidate) regexp, and also accepts * invalid regular expressions (which are delimited properly). The caller * (compiler) must perform final validation and regexp compilation. * * RegExp first char may not be '/' (single line comment) or '*' (multi- * line comment). These have already been checked above, so there is no * need below for special handling of the first regexp character as in * the E5 productions. * * About unicode escapes within regexp literals: * * E5 Section 7.8.5 grammar does NOT accept '\uHHHH escapes. * However, Section 6 states that regexps accept the escapes, * see paragraph starting with "In string literals...". * The regexp grammar, which sees the decoded regexp literal * (after lexical parsing) DOES have a '\uHHHH unicode escape. * So, for instance: * * \u1234/* should first be parsed by the lexical grammar as: * * '\u' RegularExpressionBackslashSequence * '1'

RegularExpressionNonTerminator * '2' RegularExpressionNonTerminator * '3' RegularExpressionNonTerminator * '4' RegularExpressionNonTerminator * * and the escape itself is then parsed by the regexp engine. * This is the current implementation. * * Minor spec inconsistency: * * E5 Section 7.8.5
RegularExpressionBackslashSequence is: * * \ RegularExpressionNonTerminator * * while Section A.1 RegularExpressionBackslashSequence is: * * \ NonTerminator * * The latter is not normative and a typo. *

24625. 'break'

24626. in yielder's context

24627. * E5 Section 15.4.5.1, steps 3.k - 3.n. The order at the end combines * the error case 3.l.iii and the success case 3.m-3.n. * * Note: 'length' is always in entries part, so no array abandon issues for * 'writable' update.

24628. * Avoid a GC if GC is already running. This can happen at a late * stage in a GC when we try to e.g. resize the stringtable * or compact objects.

24629. Reject attempt to change virtual properties: not part of the * standard algorithm, applies currently to e.g. virtual index * properties of buffer objects (which are virtual but writable). * (Cannot "force" modification of a virtual property.)

24630. MakeDay

24631. Push 'this' binding, check that it is a Date object; then push the * internal time value. At the end, stack is: [... this timeval]. * Returns the time value. Local time adjustment is done if requested.

24632. Cannot use duk_to_string() on the buffer because it is usually * larger than 'len'. Also, 'buf' is usually a stack buffer.

24633. non-leap year: sunday, monday, ...

24634. activation has tail called one or more times

24635. extended types: custom encoding

24636. reg

24637. -> [... enum key val]

24638. EOF

24639. XXX: optimize value stack operation

24640. * Proxy object handling

24641. * Then decode the builtins init data (see genbuiltins.py) to * init objects

24642. mark as complex (capture handling)

24643. string is a reserved word (non-strict)

24644. s <- b^(1-e) * 2

24645. * Object inspection commands: GetHeapObjInfo, GetObjPropDesc, * GetObjPropDescRange

24646. * Unreachable object about to be swept. Finalize target refcounts * (objects which the unreachable object points to) without doing * refzero processing. Recursive decrefs are also prevented when * refzero processing is disabled. * * Value cannot be a finalizable object, as they have been made * temporarily reachable for this round.

24647. Map DUK_FLX_xxx to byte size.

24648. Note that we currently parse -bytes-, not codepoints. * All non-ASCII extended UTF-8 will encode to bytes >= 0x80, * so they'll simply pass through (valid UTF-8 or not).

24649. * Get/require

24650. key, getter and setter, now reachable through object

24651. **comment:** * Finalize objects in the finalization work list. Finalized * objects are queued back to heap_allocated with FINALIZED set. * * Since finalizers may cause arbitrary side effects, they are * prevented during string table and object property allocation * resizing using the DUK_MS_FLAG_NO_FINALIZERS flag in * heap->mark_and_sweep_base_flags. In this case the objects * remain in the finalization work list after mark-and-sweep * exits and they may be finalized on the next pass. * * Finalization currently happens inside "MARKANDSWEET_RUNNING" * protection (no mark-and-sweep may be triggered by the * finalizers). As a side effect: * * 1) an out-of-memory error inside a finalizer will not * cause a mark-and-sweep and may cause the finalizer * to fail unnecessarily * 2) any temporary objects whose refcount decreases to zero * during finalization will not be put into refzero_list; * they can only be collected by another mark-and-sweep * * This is not optimal, but since the sweep for this phase has * already happened, this is probably good enough for now.

label: code-design

24652. * A Dragon4 number-to-string variant, based on: * * Guy L. Steele Jr., Jon L. White: "How to Print Floating-Point Numbers * Accurately" * * Robert G. Burger, R. Kent Dybvig: "Printing Floating-Point Numbers * Quickly and Accurately" * * The current algorithm is based on Figure 1 of the Burger-Dybvig paper, * i.e. the base implementation without logarithm estimation speedups * (these would increase code footprint considerably). Fixed-format output * does not follow the suggestions in the paper; instead, we generate an * extra digit and round-with-carry. * * The same algorithm is used for number parsing (with b=10 and B=2) * by generating one extra digit and doing rounding manually. * * See doc/number-conversion.rst for limitations.

24653. holder will be set to the target object, not the actual object * where the property was found (see duk_get_identifier_reference()).

24654. 'Array'

24655. * INITSET/INITGET are only used to initialize object literal keys. * The compiler ensures that there cannot be a previous data property * of the same name. It also ensures that setter and getter can only * be initialized once (or not at all).

24656. extended utf-8 not allowed for URIs

24657. * E5 Section 15.7.2.1 requires that the constructed object * must have the original Number.prototype as its internal * prototype. However, since Number.prototype is non-writable * and non-configurable, this doesn't have to be enforced here: * The default object (bound to 'this') is OK, though we have * to change its class. * * Internal value set to ToNumber(arg) or +0; if no arg given, * ToNumber(undefined) = NaN, so special treatment is needed * (above). String internal value is immutable.

24658. bytecode limits

24659. guaranteed to find an empty slot

24660. Because 'day since epoch' can be negative and is used to compute weekday * using a modulo operation, add this multiple of 7 to avoid negative values * when year is below 1970 epoch. Ecmascript time values are restricted to * +/- 100 million days from epoch, so this adder fits nicely into 32 bits. * Round to a multiple of 7 (= floor(100000000 / 7) * 7) and add margin.

24661. Allow trailing garbage (e.g. treat "123foo" as "123")

24662. **comment:** XXX: Are steps 6 and 7 in E5 Section 15.11.4.4 duplicated by * accident or are they actually needed? The first ToString() * could conceivably return 'undefined'.

label: code-design

24663. **comment:** Size sanity check. Should not be necessary because caller is * required to check this, but we don't want to cause a segfault * if the size wraps either in duk_size_t computation or when * storing the size in a 16-bit field.

label: code-design

24664. steps 3.h and 3.i

24665. Note: toJSON() is a generic function which works even if 'this' * is not a Date. The sole argument is ignored.

24666. module (argument)

24667. valstack space that suffices for all local calls, including recursion * of other than Duktape calls (getters etc)

24668. current parent level allows 'in' token

24669. **comment:** XXX: fast-int-to-double

label: code-design

24670. * ArrayBuffer.isView()

24671. Magically bound variable cannot be an accessor. However, * there may be an accessor property (or a plain property) in * place with magic behavior removed. This happens e.g. when * a magic property is redefined with defineProperty(). * Cannot assert for "not accessor" here.

24672. * We could clear the book-keeping variables for the topmost activation, * but don't do so now.

24673. * Debugging enabled

24674. * E5 Section 7.4, allow SourceCharacter (which is any 16-bit * code point).

24675. **comment:** XXX: duk_set_length

label: code-design

24676. **comment:** XXX: which lists should participate? to be finalized?

label: code-design

24677. XXXXXX-- -----

24678. * Bitstream decoder.
24679. * The handling here is a bit tricky. If a previous ']' has been processed, * we have a pending split1 and a pending jump (for a previous match). These * need to be back-patched carefully. See docs for a detailed example.
24680. Step 16: update length; note that the final length may be above 32 bit range * (but we checked above that this isn't the case here)
24681. **comment:** XXX: 'internal error' is a bit of a misnomer
 label: code-design
24682. **comment:** In compatible mode and standard JSON mode, output * something useful for non-BMP characters. This won't * roundtrip but will still be more or less readable and * more useful than an error.
 label: code-design
24683. A debugger forced interrupt check is not needed here, as * problematic safe calls are not caused by side effects.
24684. Real world behavior for map(): trailing non-existent * elements don't invoke the user callback, but are still * counted towards result 'length'.
24685. Ensure argument name is not a reserved word in current * (final) strictness. Formal argument parsing may not * catch reserved names if strictness changes during * parsing. * * We only need to do this in strict mode because non-strict * keyword are always detected in formal argument parsing.
24686. retval (sub-atom char length) unused, tainted as complex above
24687. no pre-checks now, assume a previous yield() has left things in * tip-top shape (longjmp handler will assert for these).
24688. * Peephole optimizer for finished bytecode. * * Does not remove opcodes; currently only straightens out unconditional * jump chains which are generated by several control structures.
24689. should not happen
24690. Avoid doing an actual write callback with length == 0, * because that's reserved for a write flush.
24691. DUK_TOK_REGEXP
24692. * Peephole optimize JUMP chains.
24693. * Execution timeout check
24694. Recursive value reviver, implements the Walk() algorithm. No C recursion * check is done here because the initial parsing step will already ensure * there is a reasonable limit on C recursion depth and hence object depth.
24695. type to represent a reg/const reference during compilation
24696. **comment:** Indent helper. Calling code relies on js_ctx->recursion_depth also being * directly related to indent depth.
 label: code-design
24697. DUK_TOK_PUBLIC
24698. E5 Sections 11.8.4, 11.8.5; x >= y --> not (x < y)
24699. 'get'
24700. **comment:** function name (borrowed reference), ends up in _name
 label: code-design
24701. repl_value
24702. quotes
24703. 'undefined' already on stack top
24704. 'res' is used for "left", and 'tmp' for "right"
24705. def_value
24706. Delete elements required by a smaller length, taking into account * potentially non-configurable elements. Returns non-zero if all * elements could be deleted, and zero if all or some elements could * not be deleted. Also writes final "target length" to 'out_result_len'. * This is the length value that should go into the 'length' property * (must be set by the caller). Never throws an error.
24707. * duk_handle_call_protected() and duk_handle_call_unprotected(): * call into a Duktape/C or an Ecmascript function from any state. * * [func this arg1 ... argN] * * Output stack (thr): * * [retv] (DUK_EXEC_SUCCESS) * [errobj] (DUK_EXEC_ERROR (normal error), protected call) * * Even when executing a protected call an error may be thrown in rare cases * such as an insane num_stack_args argument. If there is no catchpoint for * such errors, the fatal error handler is called. * * The error handling path should be error free, even for out-of-memory * errors, to ensure safe sandboxing. (As of Duktape 1.4.0 this is not * yet the case, see XXX notes below.)
24708. **comment:** XXX: skip null filename?
 label: code-design
24709. avoid multiply
24710. New require() function for module, updated resolution base
24711. **comment:** * Function pointers * * Printing function pointers is non-portable, so we do that by hex printing * bytes from memory.
 label: code-design
24712. "-Infinity", '-' has been eaten
24713. make absolute
24714. 'act' already set above
24715. DUK_BUFOBJ_UINT8CLAMPEDARRAY
24716. **comment:** XXX: turkish / azeri, lowercase rules
 label: code-design
24717. **comment:** Coerce like a string. This makes sense because addition also treats * buffers like strings.
 label: code-design
24718. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
24719. no comma
24720. strict outer context
24721. jump for 'finally' case or end (if no finally)
24722. * Structs
24723. DUK__L0 -> '\ char * DUK__L1 ... DUK__L5 -> more lookup
24724. from next pc
24725. If the value happens to be 0xFFFFFFFF, it's not a valid array index * but will then match DUK__NO_ARRAY_INDEX.
24726. use 'left' as a temp
24727. JSON
24728. binary arithmetic; DUK_OP_ADD, DUK_OP_EQ, other binary ops
24729. Innermost fast path processes 4 valid base-64 characters at a time * but bails out on whitespace, padding chars ('=') and invalid chars. * Once the slow path segment has been processed, we return to the * inner fast path again. This handles e.g. base64 with newlines * reasonably well because the majority of a line is in the fast path.
24730. Key and value indices are either (-2, -1) or (-1, -2). Given idx_key, * idx_val is always (idx_key ^ 0x01).
24731. * Pointer built-ins
24732. Zero length string is not accepted without quotes
24733. [key] -> []
24734. No activation, no variable access. Could also pretend * we're in the global program context and read stuff off * the global object.
24735. current function being compiled (embedded instead of pointer for more compact access)
24736. Note: src_data may be NULL if input is a zero-size * dynamic buffer.
24737. Note: 'this' may be bound to any value, not just an object
24738. **comment:** XXX: Not needed for now, so not implemented. Note that * function pointers may have different size/layout than * a void pointer.
 label: requirement
24739. out of spec, don't care
24740. **comment:** XXX: idx_val would fit into 16 bits, but using duk_small_uint_t * might not generate better code due to casting.
 label: code-design
24741. used by e.g. duk_regexp_executor.c, string built-ins

24742. **comment:** XXX: could also copy from template, but there's no way to have any * other value here now (used code has no access to the template).
label: code-design

24743. Share yield longjmp handler.

24744. curr_token slot2 (matches 'lex' slot2_idx)

24745. Error message doesn't matter: the error is ignored anyway.

24746. Have a calling activation, check for direct eval (otherwise * assume indirect eval).

24747. **comment:** Compact but lots of churn.
label: code-design

24748. Output shuffling: only one output register is realistically possible. * * (Zero would normally be an OK marker value: if the target register * was zero, it would never be shuffled. But with DUK_USE_SHUFFLE_TORTURE * this is no longer true, so use -1 as a marker instead.)

24749. **comment:** actually used
label: code-design

24750. rule match

24751. * Helper structs

24752. terminating conditions

24753. see below

24754. allow automatic semicolon even without lineterm (compatibility)

24755. Outside any activation -> put to global.

24756. +0 = break, +1 = continue

24757. try locale specific formatter; if it refuses to format the * string, fall back to an ISO 8601 formatted value in local * time.

24758. -> [val arr]

24759. * Byte matching for back-references would be OK in case- * sensitive matching. In case-insensitive matching we need * to canonicalize characters, so back-reference matching needs * to be done with codepoints instead. So, we just decode * everything normally here, too. * * Note: back-reference index which is 0 or higher than * NCapturingParens (= number of capturing parens in the * -entire- regexp) is a compile time error. However, a * backreference referring to a valid capture which has * not matched anything always succeeds! See E5 Section * 15.10.2.9, step 5, sub-step 3.

24760. this is needed for regexp mode

24761. jump to next case clause

24762. pop plain buffer, now reachable through h_bufres

24763. * PUTPROP: Ecmascript property write. * * Unlike Ecmascript primitive which returns nothing, returns 1 to indicate * success and 0 to indicate failure (assuming throw is not set). * * This is an extremely tricky function. Some examples: * * * Currently a decref may trigger a GC, which may compact an object's * property allocation. Consequently, any entry indices (e_idx) will * be potentially invalidated by a decref. * * * Exotic behaviors (strings, arrays, arguments object) require, * among other things: * * - Preprocessing before and postprocessing after an actual property * write. For example, array index write requires pre-checking the * array 'length' property for access control, and may require an * array 'length' update after the actual write has succeeded (but * not if it fails). * * - Deletion of multiple entries, as a result of array 'length' write. * * * Input values are taken as pointers which may point to the valstack. * If valstack is resized because of the put (this may happen at least * when the array part is abandoned), the pointers can be invalidated. * (We currently make a copy of all of the input values to avoid issues.)

24764. add AD*2^16

24765. Proxy handler

24766. **comment:** * Tailcall handling * * Although the callstack entry is reused, we need to explicitly unwind * the current activation (or simulate an unwind). In particular, the * current activation must be closed, otherwise something like * test-bug-reduce-judofy.js results. Also catchstack needs be unwound * because there may be non-error-catching label entries in valid tail calls.
label: code-design

24767. **comment:** Should not be required because the code below always sets both high * and low parts, but at least gcc-4.4.5 fails to deduce this correctly * (perhaps because the low part is set (seemingly) conditionally in a * loop), so this is here to avoid the bogus warning.
label: code-design

24768. Impose a maximum string length for now. Restricted artificially to * ensure adding a heap header length won't overflow size_t. The limit * should be synchronized with DUK_HBUFFER_MAX_BYTelen. * * E5.1 makes provisions to support strings longer than 4G characters. * This limit should be eliminated on 64-bit platforms (and increased * closer to maximum support on 32-bit platforms).

24769. relookup exports from module.exports in case it was changed by modSearch

24770. * Indexed read/write helpers (also used from outside this file)

24771. adv = 2 - 1 default OK

24772. **comment:** XXX: the key in 'key in obj' is string coerced before we're called * (which is the required behavior in E5/E5.1/E6) so the key is a string * here already.
label: code-design

24773. **comment:** XXX: side effect handling is quite awkward here
label: code-design

24774. act->func

24775. **comment:** Set stack top within currently allocated range, but don't reallocate. * This is performance critical especially for call handling, so whenever * changing, profile and look at generated code.
label: code-design

24776. **comment:** Neutered checks not necessary here: neutered buffers have * zero 'length' so we'll effectively skip them.
label: code-design

24777. Call setup checks callability.

24778. **comment:** Flags for intermediate value coercions. A flag for using a forced reg * is not needed, the forced_reg argument suffices and generates better * code (it is checked as it is used).
label: code-design

24779. **comment:** Non-ASCII slow path (range-by-range linear comparison), very slow
label: code-design

24780. Contract, either: * - Push value on stack and return 1 * - Don't push anything on stack and return 0

24781. 0: not IdentifierStart or IdentifierPart * 1: IdentifierStart and IdentifierPart * -1: IdentifierPart only

24782. DUK_BUFOBJ_UINT32ARRAY

24783. * Restore 'caller' property for non-strict callee functions.

24784. ($\geq e 0$) AND ($\neq f (\text{expt } b (- p 1))$) * * be <- (expt b e) == b^e * r <- (* f be 2) == 2 * f * b^e [if $b==2 \rightarrow f * b^{(e+1)}$] * s <- 2 * m+ <- be == b^e * m- <- be == b^e * k <- 0 * B <- B * low_ok <- round * high_ok <- round

24785. A -> unused (reserved for flags, for consistency with DUK_OP_CALL) * B -> target register and start reg: constructor, arg1, ..., argN * (for DUK_OP_NEWI, 'b' is indirect) * C -> num args (N)

24786. 0x60...0x6f

24787. ≥ 0

24788. mimic semantics for strings

24789. Parse a single statement. * * Creates a label site (with an empty label) automatically for iteration * statements. Also "peels off" any label statements for explicit labels.

24790. DUK__FLD_8BIT

24791. [key]

24792. normal key/value

24793. mm <- b^e

24794. steps 4.a and 4.b are tricky

24795. The external string struct is defined even when the feature is inactive.

24796. fixed arg count

24797. A -> flags * B -> return value reg/const * C -> currently unused

24798. **comment:** Because there are quite many call sites, pack error code (require at most * 8-bit) into a single argument.

label: code-design

24799. Raw helper to extract internal information / statistics about a value. * The return values are version specific and must not expose anything * that would lead to security issues (e.g. exposing compiled function * 'data' buffer might be an issue). Currently only counts and sizes and * such are given so there should not be a security impact.

24800. -> [... closure template newobj]

24801. XXX: This causes recursion up to inner function depth * which is normally not an issue, e.g. mark-and-sweep uses * a recursion limiter to avoid C stack issues. Avoiding * this would mean some sort of a work list or just refusing * to serialize deep functions.

24802. * Size-optimized pc->line mapping.

24803. return as is

24804. value1 -> resume value, value2 -> resume thread, iserror -> error/normal

24805. function: create binding for func name (function templates only, used for named function expressions)

24806. The low 8 bits map directly to duk_hobject.h DUK_PROPDESC_FLAG_xxx. * The remaining flags are specific to the debugger.

24807. **comment:** XXX: copy from caller?

label: code-design

24808. always for register bindings

24809. **comment:** XXX: optimize: allocate an array part to the necessary size (upwards * estimate) and fill in the values directly into the array part; finally * update 'length'.

label: code-design

24810. heap->dbg_udata: keep

24811. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

24812. IdentifierStart production with Letter, ASCII, and non-BMP excluded

24813. **comment:** * Custom formatter for debug printing, allowing Duktape specific data * structures (such as tagged values and heap objects) to be printed with * a nice format string. Because debug printing should not affect execution * state, formatting here must be independent of execution (see implications * below) and must not allocate memory. * * Custom format tags begin with '%!'. To safely distinguish them from * standard format tags. The following conversions are supported: * * %!T tagged value (duk_tval *) * %!O heap object (duk_heapobj *) * %!I decoded bytecode instruction * %!C bytecode instruction opcode name (arg is long) * * Everything is serialized in a JSON-like manner. The default depth is one * level, internal prototype is not followed, and internal properties are not * serialized. The following modifiers change this behavior: * * @ print pointers * # print binary representations (where applicable) * d deep traversal of own properties (not prototype) * p follow prototype chain (useless without 'd') * i include internal properties (other than prototype) * x hexdump buffers * h heavy formatting * * For instance, the following serializes objects recursively, but does not * follow the prototype chain nor print internal properties: "%!do". * * Notes: * * * Standard snprintf return value semantics seem to vary. This * implementation returns the number of bytes it actually wrote * (excluding the null terminator). If retv == buffer size, * output was truncated (except for corner cases). * * * Output format is intentionally different from Ecmascript * formatting requirements, as formatting here serves debugging * of internals. * * * Depth checking (and updating) is done in each type printer * separately, to allow them to call each other freely. * * * Some pathological structures might take ages to print (e.g. * self recursion with 100 properties pointing to the object * itself). To guard against these, each printer also checks * whether the output buffer is full; if so, early exit. * * * Reference loops are detected using a loop stack.

label: code-design

24814. dummy value used as marker

24815. never here, but fall through

24816. **comment:** * duk_handle_safe_call(): make a "C protected call" within the * current activation. * * The allowed thread states for making a call are the same as for * duk_handle_call_xxx(). * * Error handling is similar to duk_handle_call_xxx(); errors may be thrown * (and result in a fatal error) for insane arguments.

label: code-design

24817. must never longjmp

24818. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside object 'a_size'.

24819. Must set 'length' explicitly when using duk_xdef_prop_xxx() to * set the values.

24820. heapobj size and additional allocation size, followed by * type specific stuff (with varying value count)

24821. stack top contains 'true'

24822. callbacks and udata; dbg_read_cb != NULL is used to indicate attached state

24823. '\xffHandler'

24824. Note: string is a terminal heap object, so no depth check here

24825. 0x80-0x8f

24826. * Internal helper for defining an accessor property, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes. This is called * very rarely, so the implementation first sets a value to undefined * and then changes the entry to an accessor (this is to save code space).

24827. Debug print which visualizes the qsort partitioning process.

24828. XXX: this placeholder is not always correct, but use for now. * It will fail in corner cases; see test-dev-func-cons-args.js.

24829. r <- (2 * f) * b^e

24830. unreferenced with some options

24831. always terminates led()

24832. (reference is valid as long activation exists)

24833. don't report __FILE__ / __LINE__ as fileName/lineNumber

24834. **comment:** Note: don't bail out early, we must read all the ranges from * bytecode. Another option is to skip them efficiently after * breaking out of here. Prefer smallest code.

label: code-design

24835. Pause for all step types: step into, step over, step out. * This is the only place explicitly handling a step out.

24836. awkward; we assume there is space for this

24837. Peek ahead in the stream one byte.

24838. * Breakpoint management

24839. * Compact an object

24840. slot B is both a target and a source (used by extraops like DUK_EXTRAOP_INSTOF

24841. alias for above

24842. handle comma and closing brace

24843. shrink case; leave some spare

24844. strictness is not inherited, intentional

24845. only Array.prototype matches

24846. Parse a function-like expression, assuming that 'comp_ctx->curr_func' is * correctly set up. Assumes that curr_token is just after 'function' (or * 'set'/get' etc).

24847. function: always register bound

24848. **comment:** XXX: fast primitive to set a bunch of values to UNDEFINED

label: code-design

24849. Buffer size needed for duk_bi_date_format_timeval(). * Accurate value is 32 + 1 for NUL termination: * >>> len('+123456-01-23T12:34:56.123+12:34') * 32 * Include additional space to be safe.

24850. We want to compare the slice/view areas of the arguments. * If either slice/view is invalid (underlying buffer is shorter) * ensure equals() is false, but otherwise the only thing that * matters is to be memory safe.

24851. This seems reasonable overall.

24852. Loop prevention

24853. 'throw'

24854. we're running inside the caller's activation, so no change in call/catch stack or valstack bottom

24855. Assume that the native representation never contains a closing * parenthesis.

24856. varargs marker
24857. entry level reached
24858. Current strictness flag: affects API calls.
24859. Because value stack init policy is 'undefined above top', * we don't need to write, just assert.
24860. '*'
24861. call is from executor, so we know we have a jmpbuf
24862. Yield/resume book-keeping.
24863. finally part will catch
24864. Quite lenient, e.g. allow empty as zero, but don't allow trailing * garbage.
24865. IEEE requires that NaNs compare false
24866. Load inner functions to value stack, but don't yet copy to buffer.
24867. [start end]
24868. **comment:** Try to make do with a stack buffer to avoid allocating a temporary buffer. * This works 99% of the time which is quite nice.
label: code-design
24869. This testcase fails when Emscripten-generated code runs on Firefox. * It's not an issue because the failure should only affect packed * duk_tval representation, which is not used with Emscripten.
24870. **comment:** XXX: keys is an internal object with all keys to be processed * in its (gapless) array part. Because nobody can touch the keys * object, we could iterate its array part directly (keeping in mind * that it can be reallocated).
label: code-design
24871. Return value is like write(), number of bytes written. * The return value matters because of code like: * "off += buf.copy(...)".
24872. Exponent without digits (e.g. "1e" or "1e+"). If trailing garbage is * allowed, ignore exponent part as garbage (= parse as "1", i.e. exp 0).
24873. DUK_TOK_THROW
24874. **comment:** Note: not a typo, "object" is returned for a null value
label: documentation
24875. atom_char_length, atom_start_offset, atom_start_offset reflect the * atom matched on the previous loop. If a quantifier is encountered * on this loop, these are needed to handle the quantifier correctly. * new_atom_char_length etc are for the atom parsed on this round; * they're written to atom_char_length etc at the end of the round.
24876. Original idiom used, minimal code size.
24877. **comment:** 'fun' is quite rarely used, so no local for it
label: code-design
24878. **comment:** XXX: fastint
label: code-design
24879. Note: in integer arithmetic, (x / 4) is same as floor(x / 4) for non-negative * values, but is incorrect for negative ones.
24880. normal
24881. Careful with wrapping (left shifting idx would be unsafe).
24882. **comment:** XXX: for negative input offsets, 'offset' will be a large * positive value so the result here is confusing.
label: code-design
24883. no overflow
24884. "+11:22|0"
24885. borrowed: function being executed; for bound function calls, this is the final, real function, NULL for lightfuncs
24886. **comment:** * Augment error (throw time), unless alloc/double error
label: code-design
24887. -> [... ToObject(this) ToUint32(length) arg[i]]
24888. The specification has quite awkward order of coercion and * checks for toPrecision(). The operations below are a bit * reordered, within constraints of observable side effects.
24889. caller
24890. match followed by capture(s)
24891. return input buffer, converted to a Duktape.Buffer object * if called as a constructor (no change if called as a * function).
24892. We could use a switch-case for the class number but it turns out * a small if-else ladder on class masks is better. The if-ladder * should be in order of relevancy.
24893. * New length is smaller than old length, need to delete properties above * the new length. * * If array part exists, this is straightforward: array entries cannot * be non-configurable so this is guaranteed to work. * * If array part does not exist, array-indexed values are scattered * in the entry part, and some may not be configurable (preventing length * from becoming lower than their index + 1). To handle the algorithm * in E5 Section 15.4.5.1, step 1 correctly, we scan the entire property * set twice.
24894. 'ptr' is evaluated both as LHS and RHS.
24895. * Ecmascript [[Class]]
24896. **comment:** Quick reject of too large or too small exponents. This check * would be incorrect for zero (e.g. "0e1000" is zero, not Infinity) * so zero check must be above.
label: code-design
24897. finalize_list will always be processed completely
24898. **comment:** XXX: exposed duk_debug_read_pointer
label: code-design
24899. -> [... key val replacer holder]
24900. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside bufferobject length.
24901. open scope information, for compiled functions only
24902. Handle empty separator case: it will always match, and always * triggers the check in step 13.c.iii initially. Note that we * must skip to either end of string or start of first codepoint, * skipping over any continuation bytes! * * Don't allow an empty string to match at the end of the input.
24903. temp reg handling
24904. * Parse function body
24905. x <- y - z
24906. 'Pointer'
24907. relookup if changed
24908. **comment:** * Augment an error at creation time with _Tracedata/fileName/lineNumber * and allow a user error handler (if defined) to process/replace the error. * The error to be augmented is at the stack top. * * thr: thread containing the error value * thr_callstack: thread which should be used for generating callstack etc. * c_filename: C __FILE__ related to the error * c_line: C __LINE__ related to the error * noblame_fileline: if true, don't fileName/line as error source, otherwise use traceback * (needed because user code filename/line are reported but internal ones * are not) * * XXX: rename noblame_fileline to flags field; combine it to some existing * field (there are only a few call sites so this may not be worth it).
label: code-design
24909. throw_flag
24910. * Else, must be one of: * - ExpressionStatement, possibly a directive (String) * - LabelledStatement (Identifier followed by ':') * * Expressions beginning with 'function' keyword are covered by a case * above (such expressions are not allowed in standard E5 anyway). * Also expressions starting with '{' are interpreted as block * statements. See E5 Section 12.4. * * Directive detection is tricky; see E5 Section 14.1 on directive * prologue. A directive is an expression statement with a single * string literal and an explicit or automatic semicolon. Escape * characters are significant and no parens etc are allowed: * * 'use strict'; // valid 'use strict' directive * 'use\u0020strict'; // valid directive, not a 'use strict' directive * ('use strict'); // not a valid directive * * The expression is determined to consist of a single string literal * based on duk_expr_nud() and duk_expr_led() call counts. The string literal * of a 'use strict' directive is determined to lack any escapes based * num_escapes count from the lexer. Note that other directives may be * allowed to contain escapes, so a directive with escapes does not * terminate a directive prologue. * * We rely on the fact that the expression parser will not emit any * code for a single token expression. However, it will generate an * intermediate value which we will then successfully ignore. * * A similar approach is used for labels.

24911. The act->prev_caller should only be set if the entry for 'caller' * exists (as it is only set in that case, and the property is not * configurable), but handle all the cases anyway.

24912. break/continue without label

24913. **comment:** XXX: the best typing needs to be validated by perf measurement: * e.g. using a small type which is the cast to a larger duk_idx_t * may be slower than declaring the variable as a duk_idx_t in the * first place.
label: code-design

24914. **comment:** XXX: conversion errors should not propagate outwards. * Perhaps values need to be coerced individually?
label: code-design

24915. XXX: here we need to know if 'left' is left-hand-side compatible. * That information is no longer available from current expr parsing * state; it would need to be carried into the 'left' ivalue or by * some other means.

24916. non-global regexp: lastIndex never updated on match

24917. Note: undefined from Section 11.8.5 always results in false * return (see e.g. Section 11.8.3) - hence special treatment here.

24918. If there's no catch block, rc_varname will be 0 and duk_patch_tryCatch() * will replace the LDCONST with a NOP. For any actual constant (including * constant 0) the DUK_CONST_MARKER flag will be set in rc_varname.

24919. exposed because lexer needs these too

24920. empty match (may happen with empty separator) -> bump and continue

24921. As with all inspection code, we rely on the debug client providing * a valid, non-stale pointer: there's no portable way to safely * validate the pointer here.

24922. Set up curr_pc for opcode dispatch.

24923. [... pattern flags escaped_source]

24924. **comment:** The 'magic' field allows an opaque 16-bit field to be accessed by the * Duktape/C function. This allows, for instance, the same native function * to be used for a set of very similar functions, with the 'magic' field * providing the necessary non-argument flags / values to guide the behavior * of the native function. The value is signed on purpose: it is easier to * convert a signed value to unsigned (simply AND with 0xffff) than vice * versa. * * Note: cannot place nargs/magic into the heapdr flags, because * duk_hobject takes almost all flags already (and needs the spare).
label: code-design

24925. Get minimum entry part growth for a certain size.

24926. valstack index of start of args (arg1) (relative to entry valstack_bottom)

24927. [ToObject(this) item1 ... itemN arr item(i) item(i)[j]]

24928. thr->heap->lj.value2 is 'thread', will be wiped out at the end

24929. break/continue with label (label cannot be a reserved word, production is 'Identifier')

24930. property access

24931. Coerce all finite parts with ToInteger(). ToInteger() must not * be called for NaN/Infinity because it will convert e.g. NaN to * zero. If ToInteger() has already been called, this has no side * effects and is idempotent. * * Don't read dparts[DUK_DATE_IDX_WEEKDAY]; it will cause Valgrind * issues if the value is uninitialized.

24932. internal property functions

24933. Report thrown value to client coerced to string

24934. **comment:** ArrayBuffer argument is handled specially above; the rest of the * argument variants are handled by shared code below.
label: code-design

24935. DUK_USE_USER_INITJS

24936. Range limited to [0, 0x7fffffff] range, i.e. range that can be * represented with duk_int32_t. Use this when the method doesn't * handle the full 32-bit unsigned range correctly.

24937. 110x xxxx 10xx xxxx

24938. Sanity limits for stack sizes.

24939. **comment:** Linear scan: more likely because most objects are small. * This is an important fast path. * * XXX: this might be worth inlining for property lookups.
label: code-design

24940. XXX: will need a force flag if garbage collection is triggered * explicitly during paused state.

24941. Array length is larger than 'asize'. This shouldn't * happen in practice. Bail out just in case.

24942. No net refcount change.

24943. * Assert context is valid: non-NULL pointer, fields look sane. * * This is used by public API call entrypoints to catch invalid 'ctx' pointers * as early as possible; invalid 'ctx' pointers cause very odd and difficult to * diagnose behavior so it's worth checking even when the check is not 100%.

24944. * Setup value stack: clamp to 'nargs', fill up to 'nregs'

24945. This should not happen because DUK_TAG_OBJECT case checks * for this already, but check just in case.

24946. result array is already at the top of stack

24947. args go here as a comma expression in parens

24948. [... source? filename? &comp_args] (depends on flags)

24949. no extra padding

24950. Start position (inclusive) and end position (exclusive)

24951. tentative, checked later

24952. slot A is a source (default: target)

24953. indexed by recursion_depth

24954. * Fixed buffer helper useful for debugging, requires no allocation * which is critical for debugging.

24955. Order of unwinding is important

24956. * Final sigma context specific rule. This is a rather tricky * rule and this handling is probably not 100% correct now. * The rule is not locale/language specific so it is supported.

24957. tracks maximum initialized index + 1

24958. **comment:** first register that is a temporary (below: variables)
label: code-design

24959. A -> register of target object * B -> first register of value data (start_index, value1, value2, ..., valueN) * C -> number of key/value pairs (N)

24960. Just skip, leaving zeroes in the result.

24961. [... arr]

24962. Note: successive characters could be joined into string matches * but this is not trivial (consider e.g. '/xyz+/'; see docs for * more discussion).

24963. Coerce an duk_ivalue to a register or constant; result register may * be a temp or a bound register. * * The duk_ivalue argument ('x') is converted into a regconst as a * side effect.

24964. bytes in source

24965. no voluntary gc

24966. A -> result reg * B -> object reg * C -> key reg/const

24967. An object may have FINALIZED here if it was finalized by mark-and-sweep * on a previous run and refcount then decreased to zero. We won't run the * finalizer again here.

24968. **comment:** XXX: allow object to be a const, e.g. in 'foo'.toString()? * On the other hand, DUK_REGCONSTP() is slower and generates * more code.
label: code-design

24969. refcount

24970. regexp res_obj is at index 4

24971. stack[0] = callback fn * stack[1] = initialValue * stack[2] = object (coerced this) * stack[3] = length (not needed, but not popped above) * stack[4] = accumulator

24972. **comment:** Node.js Buffer variable width integer field. We don't really * care about speed here, so aim for shortest algorithm.
label: code-design

24973. The spec algorithm first does "R = ToString(separator)" before checking * whether separator is undefined. Since this is side effect free, we can * skip the ToString() here.

24974. ']'

24975. NULL with zero length represents an empty string; NULL with higher * length is also now treated like an empty string although it is * a bit dubious. This is unlike duk_push_string() which pushes a '*' if the input string is a NULL.

24976. -> [func funcname env funcname func]

24977. true, because v[1] has at least one bit set

24978. bp to use when parsing a top level Expression

24979. Caller will finish the marking process if we hit a recursion limit.

24980. y

24981. **comment:** * E5 Section 7.6: ** IdentifierStart: * UnicodeLetter * \$ * _ * \ UnicodeEscapeSequence ** IdentifierStart production has one multi-character production: ** \ UnicodeEscapeSequence ** The '\' character is -not- matched by this function. Rather, the caller * should decode the escape and then call this function to check whether the * decoded character is acceptable (see discussion in E5 Section 7.6). ** The "UnicodeLetter" alternative of the production allows letters * from various Unicode categories. These can be extracted with the * "src/extract_chars.py" script. ** Because the result has hundreds of Unicode codepoint ranges, matching * for any values >= 0x80 are done using a very slow range-by-range scan * and a packed range format. ** The ASCII portion (codepoints 0x00 ... 0x7f) is fast-patched below because * it matters the most. The ASCII related ranges of IdentifierStart are: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z'] * 0x0024 ['\$'] * 0x005f ['_']

label: code-design

24982. Must be an object, otherwise TypeError (E5.1 Section 8.10.5, step 1).

24983. TypeError if fails

24984. * Entries part

24985. * Delayed env creation check

24986. insert 'undefined' values at idx_rbase to get the * return values to idx_retbase

24987. xxx -> DUK_HBUFFEROBJECT_ELEM_INT8

24988. guarantees entry_callstack_top - 1 >= 0

24989. no code needs to be emitted, the regs already have values

24990. magic: 0=getter call, 1=Object.getPrototypeOf

24991. PC is unsigned. If caller does PC arithmetic and gets a negative result, * it will map to a large PC which is out of bounds and causes a zero to be * returned.

24992. * Debug connection write primitives

24993. **comment:** XXX: join these ops (multiply-accumulate), but only if * code footprint decreases.

label: code-design

24994. module.name for .name, default to last component if * not present.

24995. * Exposed debug macros: debugging enabled

24996. Object.defineProperty() calls [[DefineOwnProperty]] with Throw=true

24997. 0xe0-0xef

24998. compact the prototype

24999. XXX: default priority for infix operators is duk__expr_lbp(tok) -> get it here?

25000. read header

25001. prop index in 'hash part', < 0 if not there

25002. This is based on Rhino EquivalentYear() algorithm: *

<https://github.com/mozilla/rhino/blob/f99cc11d616f0cdda2c42bde72b3484df6182947/src/org.mozilla/javascript/NativeDate.java>

25003. **comment:** Clamping to zero makes the API more robust to calling code * calculation errors.

label: code-design

25004. Surround with parentheses like in JX, ensures NULL pointer * is distinguishable from null value ("null" vs "null").

25005. Attempt to write 'stack', 'fileName', 'lineNumber' works as if * user code called Object.defineProperty() to create an overriding * own property. This allows user code to overwrite .fileName etc * intuitively as e.g. "err.fileName = 'dummy'" as one might expect. * See <https://github.com/svaarala/duktape/issues/387>.

25006. object is a native function (duk_hnativefunction)

25007. [... error func fileName lineNumber]

25008. 21 bits

25009. **comment:** XXX: thread selection for mark-and-sweep is currently a hack. * If we don't have a thread, the entire mark-and-sweep is now * skipped (although we could just skip finalizations).

label: code-design

25010. embed: duk_hstring ptr

25011. enumeration

25012. * Free memory

25013. Maximum value check ensures 'nbytes' won't wrap below.

25014. Use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to an existing one).

25015. currently, even for Array.prototype

25016. unsigned intentionally

25017. DUK_TOK_SNEQ

25018. only outer_lex_env matters, as functions always get a new * variable declaration environment.

25019. No need to assert, buffer size maximum is 0xffff.

25020. default is explicit index read/write copy

25021. not enumerable

25022. * Exposed data

25023. avoid problems if p == h->prototype

25024. DUK_USE_PROVIDE_DEFAULT_ALLOC_FUNCTIONS

25025. Output shuffle needed after main operation

25026. For native calls must be NULL so we don't sync back

25027. **comment:** XXX: very basic optimization -> duk_get_prop_stridx_top

label: code-design

25028. omitted

25029. 'toGMTString'

25030. * Object related * * Note: seal() and freeze() are accessible through EcmaScript bindings, * and are not exposed through the API.

25031. * ISO 8601 subset parser.

25032. Internal keys are prefixed with 0xFF in the stringtable * (which makes them invalid UTF-8 on purpose).

25033. without explicit non-BMP support, assume non-BMP characters * are always accepted as identifier characters.

25034. [res]

25035. dash is already 0

25036. Object flag. There are currently 26 flag bits available. Make sure * this stays in sync with debugger object inspection code.

25037. This may only happen if built-ins are being "torn down". * This behavior is out of specification scope.

25038. Not possible because array object 'length' is present * from its creation and cannot be deleted, and is thus * caught as an existing property above.

25039. Note: strictness is not inherited from the current Duktape/C * context. Otherwise it would not be possible to compile * non-strict code inside a Duktape/C activation (which is * always strict now). See tests/api/test-eval-strictness.c * for discussion.

25040. A -> flags * B -> base register for call (base -> func, base+1 -> this, base+2 -> arg1 ... base+2+N-1 -> argN) * (for DUK_OP_CALLI, 'b' is indirect) * C -> nargs

25041. * Wrapper for jmp_buf. * * This is used because jmp_buf is an array type for backward compatibility. * Wrapping jmp_buf in a struct makes pointer references, sizeof, etc, * behave more intuitively. * * http://en.wikipedia.org/wiki/Setjmp.h#Member_types

25042. * This is a bit tricky to implement portably. The result depends * on the timestamp (specifically, DST depends on the timestamp). * If e.g. UNIX APIs are used, they'll have portability issues with * very small and very large years. * * Current approach: * * - Stay within portable UNIX limits by using equivalent year mapping. * Avoid year 1970 and 2038 as some conversions start to fail, at * least on some platforms. Avoiding 1970 means that there are * currently DST discrepancies for 1970. * * - Create a UTC and local time breakdowns from 't'. Then create * a time_t using gmtime() and localtime() and compute the time *

difference between the two. ** Equivalent year mapping (E5 Section 15.9.1.8): ** If the host environment provides functionality for determining * daylight saving time, the implementation of ECMAScript is free * to map the year in question to an equivalent year (same * leap-year-ness and same starting week day for the year) for which * the host environment provides daylight saving time information. * The only restriction is that all equivalent years should produce * the same result. ** This approach is quite reasonable but not entirely correct, e.g. * the specification also states (E5 Section 15.9.1.8): ** The implementation of ECMAScript should not try to determine * whether the exact time was subject to daylight saving time, but * just whether daylight saving time would have been in effect if * the _current daylight saving time algorithm_ had been used at the * time. This avoids complications such as taking into account the * years that the locale observed daylight saving time year round. ** Since we rely on the platform APIs for conversions between local * time and UTC, we can't guarantee the above. Rather, if the platform * has historical DST rules they will be applied. This seems to be the * general preferred direction in Ecmascript standardization (or at least * implementations) anyway, and even the equivalent year mapping should * be disabled if the platform is known to handle DST properly for the * full Ecmascript range. ** The following has useful discussion and links: ** https://bugzilla.mozilla.org/show_bug.cgi?id=351066

25043. keep val_highest

25044. **comment:** XXX: limit to quoted strings only, to save keys from being cluttered?

label: code-design

25045. DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT

25046. can't get value, may be accessor

25047. Relookup and initialize dispatch loop variables. Debugger check.

25048. * Longjmp state, contains the information needed to perform a longjmp. * Longjmp related values are written to value1, value2, and iserror.

25049. leave result on stack top

25050. Resolved, normalized absolute module ID

25051. marker; not actual tagged value

25052. re-lookup curr char on first round

25053. [... formals arguments map mappedNames]

25054. borrowed: full duk_tval for function being executed; for lightfuncs

25055. Shared handler to minimize parser size. Cause will be * hidden, unfortunately, but we'll have an offset which * is often quite enough.

25056. Skip dvalue.

25057. **comment:** XXX: inefficient; block remove primitive

label: code-design

25058. If duk__value_toplain_raw() allocates a temp, forget it and * restore next temp state.

25059. DUK_HOBJECT_FLAG_CONSTRUCTABLE varies

25060. [... pattern flags]

25061. custom; implies DUK_HOBJECT_IS_THREAD

25062. [thisArg arg1 ... argN]

25063. for array and object literals

25064. validation of the regexp is caller's responsibility

25065. regnum is sane

25066. * To avoid creating a heavy intermediate value for the list of ranges, * only the start token ('[or '[^') is parsed here. The regexp * compiler parses the ranges itself.

25067. **comment:** XXX: Currently function source code is not stored, as it is not * required by the standard. Source code should not be stored by * default (user should enable it explicitly), and the source should * probably be compressed with a trivial text compressor; average * compression of 20-30% is quite easy to achieve even with a trivial * compressor (RLE + backwards lookup). ** Debugging needs source code to be useful: sometimes input code is * not found in files as it may be generated and then eval()'d, given * by dynamic C code, etc. ** Other issues: * * - Need tokenizer indices for start and end to substring * - Always normalize function declaration part? * - If we keep _Formals, only need to store body

label: code-design

25068. Preinc/predic for var-by-name, slow path.

25069. tm_isdst is both an input and an output to mktime(), use 0 to * avoid DST handling in mktime(): * - <https://github.com/svaarala/duktape/issues/406> * - <http://stackoverflow.com/questions/8558919/mktime-and-tm-isdst>

25070. Small variant; roughly 150 bytes smaller than the fast variant.

25071. -> [... closure template env funcname]

25072. * Free the heap. ** Frees heap-related non-heap-tracked allocations such as the * string intern table; then frees the heap allocated objects; * and finally frees the heap structure itself. Reference counts * and GC markers are ignored (and not updated) in this process, * and finalizers won't be called. ** The heap pointer and heap object pointers must not be used * after this call.

25073. rescue no longer supported

25074. res->mark_and_sweep_trigger_counter == 0 -> now causes immediate GC; which is OK

25075. _Pc2line

25076. Sanity checks for string and token defines

25077. number of escapes and line continuations (for directive prologue)

25078. MakeTime

25079. **comment:** XXX: this is possible without resorting to the value stack

label: code-design

25080. ignored

25081. NULL for lightfunc

25082. lookup shorthands (note: assume context variable is named 'lex_ctx')

25083. Other callbacks are optional.

25084. truncated in case pass 3 needed

25085. With constants and inner functions on value stack, we can now * atomically finish the function 'data' buffer, bump refcounts, * etc. ** Here we take advantage of the value stack being just a duk_tval * array: we can just memcpy() the constants as long as we incref * them afterwards.

25086. DUK_USE_NONSTD_FUNC_SOURCE_PROPERTY

25087. Leave 'getter' on stack

25088. Called as a function, pattern has [[Class]] "RegExp" and * flags is undefined -> return object as is.

25089. Q...f

25090. x < y

25091. set exotic behavior only after we're done

25092. **comment:** Default function to write a formatted log line. Writes to stderr, * appending a newline to the log line. ** The argument is a buffer whose visible size contains the log message. * This function should avoid coercing the buffer to a string to avoid * string table traffic.

label: code-design

25093. Current size is about 152 bytes.

25094. **comment:** These base values are never used, but if the compiler doesn't know * that DUK_ERROR() won't return, these are needed to silence warnings. * On the other hand, scan-build will warn about the values not being * used, so add a DUK_UNREF.

label: code-design

25095. hash size relative to entries size: for value X, approx. hash_prime(e_size + e_size / X)

25096. while repl

25097. Note: we don't parse back exponent notation for anything else * than radix 10, so this is not an ambiguous check (e.g. hex * exponent values may have 'e' either as a significand digit * or as an exponent separator). ** If the exponent separator occurs twice, 'e' will be interpreted * as a digit (= 14) and will be rejected as an invalid decimal * digit.

25098. Setup state. Initial ivalue is 'undefined'.

25099. 'object'

25100. If we processed any debug messages breakpoints may have * changed; restart execution to re-check active breakpoints.

25101. 1 1 1 <32 bits>

25102. **comment:** Leading / trailing whitespace is sometimes accepted and * sometimes not. After white space trimming, all valid input * characters are pure ASCII.

label: code-design

25103. patch in range count later
25104. * If tracebacks are disabled, 'fileName' and 'lineNumber' are added * as plain own properties. Since Error.prototype has accessors of * the same name, we need to define own properties directly (cannot * just use e.g. duk_put_prop_stridx). Existing properties are not * overwritten in case they already exist.
25105. 0: toString
comment: XXX: unnecessary string coercion for array indices, * intentional to keep small.
label: code-design
25107. UTF-8 encoded bytes escaped as %xx%xx%xx... -> 3 * nbytes. * Codepoint range is restricted so this is a slightly too large * but doesn't matter.
25108. lowest mantissa for this exponent
25109. * Traceback handling when tracebacks disabled. * * The fileName / lineNumber stubs are now necessary because built-in * data will include the accessor properties in Error.prototype. If those * are removed for builds without tracebacks, these can also be removed. * 'stack' should still be present and produce a ToString() equivalent: * this is useful for user code which prints a stacktrace and expects to * see something useful. A normal stacktrace also begins with a ToString() * of the error so this makes sense.
25110. all we need to know
25111. ASSIGNMENT EXPRESSION
25112. IdentityEscape
25113. Must decrease recursion depth before returning.
25114. note: mixing len into seed improves hashing when skipping
25115. For join(), nargs is 1. For toLocaleString(), nargs is 0 and * setting the top essentially pushes an undefined to the stack, * thus defaulting to a comma separator.
25116. Return with final function pushed on stack top.
25117. Note: 'e' and 'E' are also accepted here.
25118. '%c', passed concretely as int
25119. thread resumed another thread (active but not running)
25120. or a non-catching entry
25121. next char
25122. -> [... key val]
25123. add F
comment: return a specific NaN (although not strictly necessary)
label: code-design
25125. * Variant 1
25126. global regexp: lastIndex updated on match
comment: pre-check how many atom copies we're willing to make (atom_copies not needed below)
label: requirement
25128. Current size (not counting a dynamic buffer's "spare").
25129. +2 = catcher value, catcher lj_type
25130. parsing in "directive prologue", recognize directives
25131. non-strict: non-deletable, writable
25132. DUK_USE_64BIT_OPS
25133. simulate alloc failure on every realloc (except when mark-and-sweep is running)
25134. 7
comment: alloc temp just in case, to update max temp
label: code-design
25136. * An array must have a 'length' property (E5 Section 15.4.5.2). * The special array behavior flag must only be enabled once the * length property has been added. * * The internal property must be a number (and preferably a * fastint if fastint support is enabled).
25137. * Create a hstring and insert into the heap. The created object * is directly garbage collectable with reference count zero. * * The caller must place the interned string into the stringtable * immediately (without chance of a longjmp); otherwise the string * is lost.
25138. Note: this is correct even for default clause statements: * they participate in 'fall-through' behavior even if the * default clause is in the middle.
25139. exposed, used by e.g. duk.bi.date.c
25140. * Parsing of ints and floats
25141. flags: "gi"
25142. For TypedArrays 'undefined' return value is specified * by ES6 (matches V8).
25143. * Return target object
25144. side effects
25145. [... result]
25146. only objects have finalizers
25147. interpret e.g. '0x' and '0xg' as a NaN (= parse error)
comment: * Get and call the finalizer. All of this must be wrapped * in a protected call, because even getting the finalizer * may trigger an error (getter may throw one, for instance).
label: code-design
25149. **comment:** * String/JSON conversions * * Human readable conversions are now basically ISO 8601 with a space * (instead of 'T') as the date/time separator. This is a good baseline * and is platform independent. * * A shared native helper to provide many conversions. Magic value contains * a set of flags. The helper provides: * * toString() * toDateString() * toTimeString() * toLocaleString() * toLocaleDateString() * toLocaleTimeString() * toUTCString() * toISOString() * * Notes: * * - Date.prototype.toGMTString() and Date.prototype.toUTCString() are * required to be the same EcmaScript function object (!), so it is * omitted from here. * * - Date.prototype.toUTCString(): E5.1 specification does not require a * specific format, but result should be human readable. The * specification suggests using ISO 8601 format with a space (instead of 'T') separator if a more human readable format is not available. * * - Date.prototype.toISOString(): unlike other conversion functions, * toISOString() requires a RangeError for invalid date values.
label: code-design
25150. * Thread support.
25151. bytecode start offset of the atom parsed in this loop * (allows quantifiers to copy the atom bytecode)
25152. temp copy, write back for next loop
25153. char loop
25154. basic platform types
25155. we currently assume virtual properties are not configurable (as none of them are)
25156. Stage 1: find highest preventing non-configurable entry (if any). * When forcing, ignore non-configurability.
25157. **comment:** XXX: keep property attributes or tweak them here? * Properties will now be non-configurable even when they're * normally configurable for the global object.
label: code-design
25158. address
25159. * Property already exists. Steps 5-6 detect whether any changes need * to be made.
25160. may be NULL
25161. * Parse EcmaScript source InputElementDiv or InputElementRegExp * (E5 Section 7), skipping whitespace, comments, and line terminators. * * Possible results are: * (1) a token * (2) a line terminator (skipped) * (3) a comment (skipped) * (4) EOF * * White space is automatically skipped from the current position (but * not after the input element). If input has already ended, returns * DUK_TOK_EOF indefinitely. If a parse error occurs, uses an DUK_ERROR0 * macro call (and hence a longjmp through current heap longjmp context). * Comments and line terminator tokens are automatically skipped. * * The input element being matched is determined by regexp_mode; if set, * parses a InputElementRegExp, otherwise a InputElementDiv. The * difference between these are handling of productions starting with a * forward slash. * * If strict_mode is set, recognizes additional future reserved words * specific to strict mode, and refuses to parse octal literals. * * The matching strategy below is to (currently) use a six character * lookup window to quickly determine which production is the -longest- * matching one, and then parse that. The top-level if-else clauses * match the first character, and the code blocks for each clause * handle -all- alternatives for that first character. EcmaScript

* specification uses the "longest match wins" semantics, so the order * of the if-clauses matters. ** Misc notes: *** Ecmascript numeric literals do not accept a sign character. * Consequently e.g. "-1.0" is parsed as two tokens: a negative * sign and a positive numeric literal. The compiler performs * the negation during compilation, so this has no adverse impact. *** There is no token for "undefined": it is just a value available * from the global object (or simply established by doing a reference * to an undefined value). *** Some contexts want Identifier tokens, which are IdentifierNames * excluding reserved words, while some contexts want IdentifierNames * directly. In the latter case e.g. "while" is interpreted as an * identifier name, not a DUK_TOK_WHILE token. The solution here is * to provide both token types: DUK_TOK_WHILE goes to 't' while * DUK_TOK_IDENTIFIER goes to 't_nores', and 'slot1' always contains * the identifier / keyword name. *** Directive prologue needs to identify string literals such as * "use strict" and 'use strict', which are sensitive to line * continuations and escape sequences. For instance, "use\u0020strict" * is a valid directive but is distinct from "use strict". The solution * here is to decode escapes while tokenizing, but to keep track of the * number of escapes. Directive detection can then check that the * number of escapes is zero. *** Multi-line comments with one or more internal LineTerminator are * treated like a line terminator to comply with automatic semicolon * insertion.

25162. IEEE double is approximately 16 decimal digits; print a couple extra

25163. * Resizing and hash behavior

25164. String sanitizer which escapes ASCII control characters and a few other * ASCII characters, passes Unicode as is, and replaces invalid UTF-8 with * question marks. No errors are thrown for any input string, except in out * of memory situations.

25165. **comment:** Try to optimize X <op>= Y for reg-bound * variables. Detect side-effect free RHS * narrowly by seeing whether it emits code. * If not, rewind the code emitter and overwrite * the unnecessary temp reg load.

label: code-design

25166. [obj key undefined]

25167. **comment:** XXX: can be optimized for smaller footprint esp. on 32-bit environments

label: code-design

25168. * isArray()

25169. DUK_USE_HOBJECT_HASH_PART

25170. trailing elisions?

25171. * Simple commands

25172. XXX: The TypeError is currently not applied to bound * functions because the 'strict' flag is not copied by * bind(). This may or may not be correct, the specification * only refers to the value being a "strict mode Function * object" which is ambiguous.

25173. [... global val] -> [... global]

25174. length in codepoints (must be E5 compatible)

25175. The smallest fastint is no longer 48-bit when * negated. Positive zero becomes negative zero * (cannot be represented) when negated.

25176. Decode helper. Return zero on error.

25177. XXX: specify array size, as we know it

25178. **comment:** Numbers are normalized to big (network) endian. We can * (but are not required) to use integer dvalues when there's * no loss of precision. ** XXX: share check with other code; this check is slow but * reliable and doesn't require careful exponent/mantissa * mask tricks as in the fastint downgrade code.

label: code-design

25179. * Use static helpers which can work with math.h functions matching * the following signatures. This is not portable if any of these math * functions is actually a macro. ** Typing here is intentionally 'double' wherever values interact with * the standard library APIs.

25180. 1111 0xxx; 4 bytes

25181. start array index of current MPUTARR set

25182. cp == -1 (EOF) never matches and causes return value 0

25183. **comment:** XXX: the handling of character range detection is a bit convoluted. * Try to simplify and make smaller.

label: code-design

25184. DUK_OP_CALL flags in A

25185. * Shared helper for non-bound func lookup. * Returns duk_hobject * to the final non-bound function (NULL for lightfunc).

25186. Must be >= 0 and multiple of element size.

25187. A stubbed built-in is useful for e.g. compilation torture testing with BCC.

25188. Decode a plain string consisting entirely of identifier characters. * Used to parse plain keys (e.g. "foo: 123").

25189. note that we can't reliably pop anything here

25190. [... put_value]

25191. 5

25192. ref.value and ref.this_binding invalidated here

25193. nret

25194. object is a compiled function (duk_hcompiledfunction)

25195. Push a new closure on the stack. ** Note: if fun_temp has NEWENV, i.e. a new lexical and variable declaration * is created when the function is called, only outer_lex_env matters * (outer_var_env is ignored and may or may not be same as outer_lex_env).

25196. Need a short reg/const, does not have to be a mutable temp.

25197. delete Duktape.modLoaded[resolved_id]

25198. keep current valstack_top

25199. Disabled until fixed, see above.

25200. 'message'

25201. Just transfer the refcount from act->prev_caller to tv_caller, * so no need for a refcount update. This is the expected case.

25202. Suggested step-by-step method from documentation of RtlTimeToSecondsSince1970: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928(v=vs.85).aspx)

25203. * Migrate array to start of entries if requested. ** Note: from an enumeration perspective the order of entry keys matters. * Array keys should appear wherever they appeared before the array abandon * operation.

25204. 0x70...0x7f

25205. Maximum iteration count for computing UTC-to-local time offset when * creating an Ecmascript time value from local parts.

25206. **comment:** XXX: depend on available temps?

label: code-design

25207. default case

25208. %u; only 16 bits are guaranteed

25209. MultiplicativeExpression

25210. Trim white space (= allow leading and trailing whitespace)

25211. * Call user provided module search function and build the wrapped * module source code (if necessary). The module search function * can be used to implement pure Ecmascript, pure C, and mixed * Ecmascript/C modules. ** The module search function can operate on the exports table directly * (e.g. DLL code can register values to it). It can also return a * string which is interpreted as module source code (if a non-string * is returned the module is assumed to be a pure C one). If a module * cannot be found, an error must be thrown by the user callback. ** Because Duktape.modLoaded[] already contains the module being * loaded, circular references for C modules should also work * (although expected to be quite rare).

25212. does not modify tv_x

25213. 26: setSeconds

25214. patch pending jump and split

25215. traceback depth doesn't take into account the filename/line * special handling above (intentional)

25216. DUK_USE_ERRTHROW || DUK_USE_ERRCREATE

25217. only needed by debugger for now

25218. Compute day number of the first day of a given year.

25219. ignore_loop

25220. * Begin

25221. Output #1: resolved absolute name

25222. and even number

25223. * DELPROP: Ecmascript property deletion.
25224. normal: implicit leading 1-bit
25225. may throw an error
25226. * Encoding constants, must match genbuiltins.py
25227. 'RegExp'
25228. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize?
 label: code-design
25229. <<
25230. **comment:** XXX: double evaluation of DUK_HCOMPILEDFUNCTION_GET_DATA()
 label: code-design
25231. ensure full memcmp() fits in while
25232. unreachable
25233. push operation normalizes NaNs
25234. getter/setter
25235. x == 0x00 (EOF) causes syntax_error
25236. XXX: range assert
25237. **comment:** XXX: move masks to js_ctx? they don't change during one * fast path invocation.
 label: code-design
25238. flags: "i"
25239. * Init stringtable: probe variant
25240. Catches both doubles and cases where only one argument is a fastint
25241. key coercion (unless already coerced above)
25242. **comment:** XXX: There's quite a bit of overlap with buffer creation handling in * duk_bi_buffer.c. Look for overlap and refactor.
 label: code-design
25243. Limit checks for bytecode byte size and line number.
25244. The necessary #includes are in place in duk_config.h.
25245. **comment:** XXX: this could be optimized
 label: code-design
25246. duk_handle_return() is guaranteed never to throw, except * for potential out-of-memory situations which will then * propagate out of the executor longjmp handler.
25247. **comment:** * XXX: if duk_handle_call() took values through indices, this could be * made much more sensible. However, duk_handle_call() needs to fudge * the 'this' and 'func' values to handle bound function chains, which * is now done "in-place", so this is not a trivial change.
 label: code-design
25248. Number serialization has a significant impact relative to * other fast path code, so careful fast path for fastints.
25249. * Compiler intermediate values * * Intermediate values describe either plain values (e.g. strings or * numbers) or binary operations which have not yet been coerced into * either a left-hand-side or right-hand-side role (e.g. object property).
25250. **comment:** Silence a few global unused warnings here.
 label: code-design
25251. -> [... arr num]
25252. tzoffset seconds are dropped; 16 bits suffice for * time offset in minutes
25253. thread has yielded
25254. Lightfuncs are always strict.
25255. '\xffArgs'
25256. **comment:** XXX: for array instances we could take a shortcut here and assume * Array.prototype doesn't contain an array index property.
 label: code-design
25257. PropertyName -> IdentifierName | StringLiteral | NumericLiteral
25258. -1 = top callstack entry, callstack[callstack_top - 1] * -callstack_top = bottom callstack entry, callstack[0]
25259. Further state-dependent pre-checks
25260. E5 Section 8.12.8
25261. is_decl
25262. at least one activation, ours
25263. s <- b^(-e) * 2
25264. use p_src_base from now on
25265. 'aint' result as complex
25266. no issues with memcmp() zero size, even if broken
25267. * Disjunction struct: result of parsing a disjunction
25268. * Emit initializers in sets of maximum max_init_pairs keys. * Setter/getter is handled separately and terminates the * current set of initializer values. Corner cases such as * single value initializers do not have special handling now.
25269. * E5 Section 7.3: CR LF is detected as a single line terminator for * line numbers. Here we also detect it as a single line terminator * token.
25270. We don't need to sync back thr->ptr_curr_pc here because * the bytecode executor always has a setjmp catchpoint which * does that before errors propagate to here.
25271. Previous entry was inside visited[], nothing to do.
25272. **comment:** XXX: optimize temp reg use
 label: code-design
25273.toFixed()
25274. Detect zero special case.
25275. Number and (minimum) size of bigints in the nc_ctx structure.
25276. We come here for actual aborts (like encountering . toJSON()) * but also for recursion/loop errors. Bufwriter size can be * kept because we'll probably need at least as much as we've * allocated so far.
25277. * Preliminaries: trim, sign, Infinity check * * We rely on the interned string having a NUL terminator, which will * cause a parse failure wherever it is encountered. As a result, we * don't need separate pointer checks. * * There is no special parsing for 'NaN' in the specification although * 'Infinity' (with an optional sign) is allowed in some contexts. * Some contexts allow plus/minus sign, while others only allow the * minus sign (like JSON.parse()). * * Automatic hex number detection (leading '0x' or '0X') and octal * number detection (leading '0' followed by at least one octal digit) * is done here too.
25278. duk_unicode_idp_m_ids_noabmp[]
25279. '(?)'
25280. * Process replacer/proplist (2nd argument to JSON.stringify)
25281. may be a string constant
25282. return total chars written excluding terminator
25283. [(builtin objects) name]
25284. new_free / size <= 1 / DIV <=> new_free <= size / DIV
25285. This shouldn't happen; call sites should avoid looking up * _Finalizer "through" a Proxy, but ignore if we come here * with a Proxy to avoid finalizer re-entry.
25286. [... holder name val enum obj_key]
25287. * Create a new function object based on a "template function" which contains * compiled bytecode, constants, etc, but lacks a lexical environment. * * Ecmascript requires that each created closure is a separate object, with * its own set of editable properties. However, structured property values * (such as the formal arguments list and the variable map) are shared. * Also the bytecode, constants, and inner functions are shared. * * See E5 Section 13.2 for detailed requirements on the function objects; * there are no similar requirements for function "templates" which are an * implementation dependent internal feature. Also see function-objects.rst * for a discussion on the function instance properties provided by this * implementation. * * Notes: * * * Order of internal properties should match frequency of use, since the * properties will be linearly scanned on lookup (functions usually don't * have enough properties to warrant a hash part). * * * The

created closure is independent of its template; they do share the * same 'data' buffer object, but the template object itself can be freed * even if the closure object remains reachable.

25288. E5 Section 11.13.1 (and others) step 4 never matches for prop writes -> no check

25289. exit() afterwards to satisfy "noreturn"

25290. string compare is the default (a bit oddly)

25291. %xx%xx...%xx', p points to char after first %'

25292. **comment:** XXX: need a `toplaint_ignore()` which will only coerce a value to a temp * register if it might have a side effect. Side-effect free values do not * need to be coerced.

label: code-design

25293. $2^4 \rightarrow 1/16 = 6.25\%$ spare

25294. * `reduce()`, `reduceRight()`

25295. * Macros for accessing size fields

25296. * Resolve module identifier into canonical absolute form.

25297. leap year: sunday, monday, ...

25298. even after a detach and possible reattach

25299. `tval`

25300. **comment:** integer mixed endian not really used now

label: code-design

25301. only advance if not tainted

25302. tail insert: don't disturb head in case refzero is running

25303. `indexOf`: NaN should cause pos to be zero. * `lastIndexOf`: NaN should cause pos to be +Infinity * (and later be clamped to len).

25304. 0.000...

25305. leave stack unbalanced on purpose

25306. [...] func this <bound args> arg1 ... argN]

25307. Bitfield for each DUK_HBUFFEROBJECT_ELEM_xxx indicating which element types * are compatible with a blind byte copy for the TypedArray set() method (also * used for TypedArray constructor). Array index is target buffer elem type, * bitfield indicates compatible source types. The types must have same byte * size and they must be coercion compatible.

25308. **comment:** Run fake finalizer. Avoid creating unnecessary garbage.

label: code-design

25309. * Table for hex encoding bytes

25310. [...] -> [...] func]

25311. build result as: $(r \ll 32) + s$: start with (BD + E + F)

25312. We only get here when doing non-standard JSON encoding

25313. Without heap pointer compression `duk_hbuffer_dynamic` and `duk_hbuffer_external` * have the same layout so checking for fixed vs. dynamic (or external) is enough.

25314. [target] -> [enum]

25315. Handle both full and partial slice (as long as covered).

25316. envrec: (declarative) record is closed

25317. **comment:** Log frontend shared helper, magic value indicates log level. Provides * frontend functions: `trace()`, `debug()`, `info()`, `warn()`, `error()`, `fatal()`. * This needs to have small footprint, reasonable performance, minimal * memory churn, etc.

label: code-design

25318. XXX: bump preventcount by one for the duration of this call?

25319. for

25320. 15: `getUTCDay`

25321. `need_bytes` may be zero

25322. register binding lookup is based on varmap (even in first pass)

25323. prototype: the only internal property lifted outside 'e' as it is so central

25324. 3 entries actually needed below

25325. end of input and last char has been processed

25326. [start length str]

25327. array

25328. fast paths for space and tab

25329. Significand ('f') padding.

25330. **comment:** not needed

label: requirement

25331. Regardless of whether property is found in entry or array part, * it may have arguments exotic behavior (array indices may reside * in entry part for abandoned / non-existent array parts).

25332. add a new pending match jump for latest finished alternative

25333. exclusive

25334. `DUK_USE_BUILTIN_INITJS`

25335. * If selftests enabled, run them as early as possible

25336. `ivalue/ispec` helpers

25337. * Determine the effective 'this' binding and coerce the current value * on the valstack to the effective one (in-place, at `idx_this`). * * The current this value in the valstack (at `idx_this`) represents either: * - the caller's requested 'this' binding; or * - a 'this' binding accumulated from the bound function chain * * The final 'this' binding for the target function may still be * different, and is determined as described in E5 Section 10.4.3. * * For global and eval code (E5 Sections 10.4.1 and 10.4.2), we assume * that the caller has provided the correct 'this' binding explicitly * when calling, i.e.: * * - global code: `this=global object` * - direct eval: `this=copy from eval()` caller's this binding * - other eval: `this=global object` * * Note: this function may cause a recursive function call with arbitrary * side effects, because `ToObject()` may be called.

25338. `idx_this = idx_func + 1`

25339. **comment:** XXX: for real world code, could just ignore array inheritance * and only look at array own properties.

label: code-design

25340. not needed, as we exit right away

25341. XXX: return type should probably be `duk_size_t`, or explicit checks are needed for * maximum size.

25342. relookup, may have changed

25343. leading digit + fractions

25344. Select appropriate escape format automatically, and set 'tmp' to a * value encoding both the escape format character and the nibble count: * * (`nibble_count << 16`) | (`escape_char1`) | (`escape_char2`)

25345. `tc1 = true`, `tc2 = true`

25346. See comments below on `MakeTime` why these are volatile.

25347. -> [val obj val]

25348. Store lexer position, restoring if quantifier is invalid.

25349. no need to compact since we already did that in `duk_abandon_array_checked()` * (regardless of whether an array part existed or not).

25350. marker; not actual tagged type

25351. **comment:** XXX: expensive check (also shared elsewhere - so add a shared internal API call?)

label: code-design

25352. For NaN/inf, the return value doesn't matter.

25353. curr char

25354. [regexp string]

25355. **comment:** NULL not needed here
label: code-design

25356. duk_pcall_prop() may itself throw an error, but we're content * in catching the obvious errors (like toLogString() throwing an * error).

25357. overflow, more than 32 bits -> not an array index

25358. special result value handling

25359. [...] Logger clog res]

25360. [... put_value varname]

25361. intermediate join to avoid valstack overflow

25362. ptr before mark-and-sweep

25363. end-of-input breaks

25364. reserve a jumps slot after instr before target spilling, used for NEXTENUM

25365. a label site has been emitted by duk_parse_stmt() automatically * (it will also emit the ENDLABEL).

25366. * Variable access

25367. **comment:** function: function must not be tail called
label: code-design

25368. third arg: absolute index (to entire valstack) of idx_bottom of new activation

25369. **comment:** * HASPROP variant used internally. ** This primitive must never throw an error, callers rely on this. * In particular, don't throw an error for prototype loops; instead, * pretend like the property doesn't exist if a prototype sanity limit * is reached. ** Does not implement proxy behavior: if applied to a proxy object, * returns key existence on the proxy object itself.
label: code-design

25370. 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.

25371. !DUK_USE_HSTRING_CLEN

25372. DUK_ERR_EVAL: no macros needed

25373. DUK_USE_PREFER_SIZE

25374. Inref copies, keep originals.

25375. object is a buffer object (duk_hbufferobject) (always exotic)

25376. recheck that the property still exists

25377. jump from true part to end

25378. **comment:** XXX: fastint fast path would be very useful here
label: code-design

25379. XXX: happens e.g. when evaluating: String(Buffer.prototype).

25380. 'name'

25381. eat identifier

25382. * Init stringtable: fixed variant

25383. Copy values through direct validated reads and writes.

25384. break matches always

25385. * "length" maps to number of formals (E5 Section 13.2) for function * declarations/expressions (non-bound functions). Note that 'nargs' * is NOT necessarily equal to the number of arguments.

25386. -1 for opcode

25387. * Regexp instance check, bytecode check, input coercion. ** See E5 Section 15.10.6.

25388. **comment:** hash part size or 0 if unused
label: code-design

25389. **comment:** XXX: this duplicates functionality in duk_regexp.c where a similar loop is * required anyway. We could use that BUT we need to update the regexp compiler * 'nranges' too. Work this out a bit more cleanly to save space.
label: code-design

25390. helper to insert a (non-string) heap object into heap allocated list

25391. * Internal API calls which have (stack and other) semantics similar * to the public API.

25392. retval is directly usable

25393. **comment:** * Error throwing helpers ** The goal is to provide verbose and configurable error messages. Call * sites should be clean in source code and compile to a small footprint. * Small footprint is also useful for performance because small cold paths * reduce code cache pressure. Adding macros here only makes sense if there * are enough call sites to get concrete benefits.
label: code-design

25394. digit count

25395. * Hobject property set/get functionality. ** This is very central functionality for size, performance, and compliance. * It is also rather intricate; see hobject-algorithms.rst for discussion on * the algorithms and memory-management.rst for discussion on refcounts and * side effect issues. ** Notes: * * - It might be tempting to assert "refcount nonzero" for objects * being operated on, but that's not always correct: objects with * a zero refcount may be operated on by the refcount implementation * (finalization) for instance. Hence, no refcount assertions are made. * * - Many operations (memory allocation, identifier operations, etc) * may cause arbitrary side effects (e.g. through GC and finalization). * These side effects may invalidate duk_tval pointers which point to * areas subject to reallocation (like value stack). Heap objects * themselves have stable pointers. Holding heap object pointers or * duk_tval copies is not problematic with respect to side effects; * care must be taken when holding and using argument duk_tval pointers. * * - If a finalizer is executed, it may operate on the same object * we're currently dealing with. For instance, the finalizer might * delete a certain property which has already been looked up and * confirmed to exist. Ideally finalizers would be disabled if GC * happens during property access. At the moment property table realloc * disables finalizers, and all DECREFS may cause arbitrary changes so * handle DECREF carefully. * * - The order of operations for a DECREF matters. When DECREF is executed, * the entire object graph must be consistent; note that a refzero may * lead to a mark-and-sweep through a refcount finalizer.

25396. **comment:** * Compact the closure, in most cases no properties will be added later. * Also, without this the closures end up having unused property slots * (e.g. in Duktape 0.9.0, 8 slots would be allocated and only 7 used). * A better future solution would be to allocate the closure directly * to correct size (and setup the properties directly without going * through the API).
label: code-design

25397. 2^3 -> 1/8 = 12.5% min growth

25398. * Reference count updates ** Note: careful manipulation of refcounts. The top is * not updated yet, so all the activations are reachable * for mark-and-sweep (which may be triggered by decref). * However, the pointers are NULL so this is not an issue.

25399. 'hex'

25400. borrowed label name

25401. * String table algorithm: closed hashing with a probe sequence ** This is the default algorithm and works fine for environments with * minimal memory constraints.

25402. Longjmp state is kept clean in success path

25403. Longjmp callers are required to sync-and-null thr->ptr_curr_pc * before longjmp.

25404. Return invalid index; if caller uses this without checking * in another API call, the index won't map to a valid stack * entry.

25405. **comment:** XXX: faster internal way to get this
label: code-design

25406. old value is number: no refcount

25407. Using classification has smaller footprint than direct comparison.

25408. DUK_USE_ROM_GLOBAL_CLONE || DUK_USE_ROM_GLOBAL_INHERIT

25409. leading byte of match string

25410. **comment:** * Formal argument list ** We don't check for prohibited names or for duplicate argument * names here, because we don't yet know whether the function will * be strict. Function body parsing handles this retroactively.
label: code-design

25411. canonicalized by compiler

25412. [source template closure this]
25413. initial index
25414. 5 heap flags
25415. DUK_TOK_FUNCTION
25416. exponent for 'f'
25417. * Duktape includes (other than duk_features.h) ** The header files expect to be included in an order which satisfies header * dependencies correctly (the headers themselves don't include any other * includes). Forward declarations are used to break circular struct/typedef * dependencies.
25418. always 1 arg
25419. helpers
25420. ignore result
25421. DUK_JMPBUF_H_INCLUDED
25422. 'static'
25423. * sort() ** Currently qsort with random pivot. This is now really, really slow, * because there is no fast path for array parts. ** Signed indices are used because qsort() leaves and degenerate cases * may use a negative offset.
25424. extended types: compatible encoding
25425. **comment:** not allowed in strict mode, regardless of whether resolves; * in non-strict mode DELVAR handles both non-resolving and * resolving cases (the specification description is a bit confusing).
label: code-design
25426. valstack limit caller has check, prevents wrapping
25427. Note that byte order doesn't affect this test: all bytes in * 'test' will be 0xFF for two's complement.
25428. Note: number has no explicit tag (in 8-byte representation)
25429. * Calendar helpers ** Some helpers are used for getters and can operate on normalized values * which can be represented with 32-bit signed integers. Other helpers are * needed by setters and operate on un-normalized double values, must watch * out for non-finite numbers etc.
25430. Unlike year, the other parts fit into 16 bits so %d format * is portable.
25431. reg/const for case value
25432. Equivalent year mapping, used to avoid DST trouble when platform * may fail to provide reasonable DST answers for dates outside the * ordinary range (e.g. 1970-2038). An equivalent year has the same * leap-year-ness as the original year and begins on the same weekday * (Jan 1). ** The year 2038 is avoided because there seem to be problems with it * on some platforms. The year 1970 is also avoided as there were * practical problems with it; an equivalent year is used for it too, * which breaks some DST computations for 1970 right now, see e.g. * test-bi-date-tzoffset-brute-fi.js.
25433. signed integer encoding needed to work with UTF-8
25434. Here val would be potentially invalid if we didn't make * a value copy at the caller.
25435. '&'
25436. jump is inserted here (variant 3)
25437. * Closing environment records. ** The environment record MUST be closed with the thread where its activation * is. In other words (if 'env' is open): ** - 'thr' must match _env.thread * - 'func' must match _env.callee * - 'regbase' must match _env.regbase ** These are not looked up from the env to minimize code size. * * XXX: should access the own properties directly instead of using the API
25438. resume: [... initial_func undefined(= this) resume_value]
25439. ignore relimit, not constructor
25440. '\xffCallee'
25441. is normalized
25442. Variant for writing duk_tvals so that any heap allocated values are * written out as tagged heap pointers.
25443. Coerce like boolean
25444. mark-and-sweep marking reached a recursion limit and must use multi-pass marking
25445. [(builtin objects)]
25446. **comment:** XXX: tv_func is not actually needed
label: requirement
25447. * Source filename (or equivalent), for identifying thrown errors.
25448. '_proto'_
25449. **comment:** XXX: this is probably a useful shared helper: for a * duk_hbufferobject, get a validated buffer pointer/length.
label: code-design
25450. parent env is the prototype
25451. get varenv for varname (callee's declarative lexical environment)
25452. Use a new environment and there's an 'arguments' object. * We need to initialize it right now.
25453. The timevalue must be in valid Ecmascript range, but since a local * time offset can be applied, we need to allow a +/- 24h leeway to * the value. In other words, although the UTC time is within the * Ecmascript range, the local part values can be just outside of it.
25454. Note: can be safely scanned as bytes (undecoded)
25455. ======
25456. No activation matches, use undefined for both .fileName and * .lineNumber (matches what we do with a _Tracedata based * no-match lookup).
25457. also goes into flags
25458. UnaryExpression
25459. **comment:** XXX: return something more useful, so that caller can throw?
label: code-design
25460. Tailcalls are handled by back-patching the TAILCALL flag to the * already emitted instruction later (in return statement parser). * Since A and C have a special meaning here, they cannot be "shuffled".
25461. all virtual properties are non-configurable and non-writable
25462. [obj trap_result res_arr]
25463. **comment:** * The string conversion here incorporates all the necessary Ecmascript * semantics without attempting to be generic. nc_ctx->digits contains * nc_ctx->count digits (>= 1), with the topmost digit's 'position' * indicated by nc_ctx->k as follows: ** digits="123" count=3 k=0 --> 0.123 * digits="123" count=3 k=1 --> 1.23 * digits="123" count=3 k=5 --> 12300 * digits="123" count=3 k=-1 --> 0.0123 ** Note that the identifier names used for format selection are different * in Burger-Dybvig paper and Ecmascript specification (quite confusingly * so, because e.g. 'k' has a totally different meaning in each). See * documentation for discussion. ** Ecmascript doesn't specify any specific behavior for format selection * (e.g. when to use exponent notation) for non-base-10 numbers. ** The bigint space in the context is reused for string output, as there * is more than enough space for that (>1kB at the moment), and we avoid * allocating even more stack.
label: code-design
25464. depth check is done when printing an actual type
25465. Preinc/predec for object properties.
25466. * duk_re_range_callback for generating character class ranges. ** When ignoreCase is false, the range is simply emitted as is. * We don't, for instance, eliminate duplicates or overlapping * ranges in a character class. ** When ignoreCase is true, the range needs to be normalized through * canonicalization. Unfortunately a canonicalized version of a * continuous range is not necessarily continuous (e.g. [x-{ } is * continuous but [X-{ } is not). The current algorithm creates the * canonicalized range(s) space efficiently at the cost of compile * time execution time (see doc/regexp.rst for discussion). ** Note that the ctx->nranges is a context-wide temporary value * (this is OK because there cannot be multiple character classes * being parsed simultaneously).
25467. init function state: init valstack allocations
25468. **comment:** Currently about 7*152 = 1064 bytes. The space for these * duk_bignints is used also as a temporary buffer for generating * the final string. This is a bit awkward; a union would be * more correct.
label: code-design
25469. Check statement type based on the first token type. ** Note: expression parsing helpers expect 'curr_tok' to * contain the first token of the expression upon entry.
25470. **comment:** * The 'in' operator requires an object as its right hand side, * throwing a TypeError unconditionally if this is not the case. ** However, lightfuncs need to behave like fully fledged objects * here to be maximally transparent, so we need to handle them * here.

label: code-design

25471. * Number-to-string conversion. The semantics of these is very tightly * bound with the EcmaScript semantics required for call sites.

25472. **comment:** SCANBUILD: with suitable dmin/dmax limits 'd' is unused**label:** code-design

25473. 0xff => 255 - 256 = -1; 0x80 => 128 - 256 = -128

25474. DUK_USE_PARANOID_ERRORS

25475. Does not assume that jump_pc contains a DUK_OP_JUMP previously; this is intentional * to allow e.g. an INVALID opcode be overwritten with a JUMP (label management uses this).

25476. A 32-bit unsigned integer formats to at most 32 digits (the * worst case happens with radix == 2). Output the digits backwards, * and use a memmove() to get them in the right place.

25477. Without ROM objects "needs refcount update" == is heap allocated.

25478. **comment:** Note: 'act' is dangerous here because it may get invalidated at many * points, so we re-lookup it multiple times.**label:** code-design25479. **comment:** XXX: Add a flag to reject an attempt to re-attach? Otherwise * the detached callback may immediately reattach.**label:** code-design

25480. side effects -> don't use tv_x, tv_y after

25481. [... enum]

25482. if gap is empty, behave as if not given at all

25483. When JX/JC not in use, the type mask above will avoid this case if needed.

25484. DUK_TOK_LT

25485. no value

25486. * HASVAR: check identifier binding from a given environment record * without traversing its parents. * * This primitive is not exposed to user code as such, but is used * internally for e.g. declaration binding instantiation. * * See E5 Sections: * 10.2.1.1 HasBinding(N) * 10.2.1.2.1 HasBinding(N) * * Note: strictness has no bearing on this check. Hence we don't take * a 'strict' parameter.

25487. hash probe sequence

25488. * Make the C call

25489. DUK_USE_NONSTD_FUNC_STMT

25490. has access to 'this' binding

25491. **comment:** Create a temporary enumerator to get the (non-duplicated) key list; * the enumerator state is initialized without being needed, but that * has little impact.**label:** code-design25492. **comment:** XXX: optimize for string inputs: no need to coerce to a buffer * which makes a copy of the input.**label:** code-design

25493. initial size guess

25494. Buffer writes are often integers.

25495. env[funcname] = closure

25496. Safe to call multiple times.

25497. Default variants. Selection depends on speed/size preference. * Concretely: with gcc 4.8.1 -Os x64 the difference in final binary * is about +1kB for _FAST variants.

25498. formatted result limited

25499. [] -> [res]

25500. may be NaN

25501. 'undefined', artifact of lookup

25502. assume array part is comprehensive (contains all array indexed elements * or none of them); hence no need to check the entries part here.

25503. lj value1: value

25504. pushes function template

25505. Identifier, i.e. don't allow reserved words

25506. **comment:** Parse an inner function, adding the function template to the current function's * function table. Return a function number to be used by the outer function. * * Avoiding O(depth^2) inner function parsing is handled here. On the first pass, * compile and register the function normally into the 'funcs' array, also recording * a lexer point (offset/line) to the closing brace of the function. On the second * pass, skip the function and return the same 'fnum' as on the first pass by using * a running counter. * * An unfortunate side effect of this is that when parsing the inner function, almost * nothing is known of the outer function, i.e. the inner function's scope. We don't * need that information at the moment, but it would allow some optimizations if it * were used.**label:** code-design

25507. [arg toLogString]

25508. out_token->lineterm set by caller

25509. note: any entries above the catchstack top are garbage and not zeroed

25510. Error code also packs a tracedata related flag.

25511. enumerator must have no keys deleted

25512. Stack size increases or stays the same.

25513. known to be number; in fact an integer

25514. avail_bytes += need_bytes

25515. **comment:** XXX: this illustrates that a C catchpoint implemented using duk_safe_call() * is a bit heavy at the moment. The wrapper compiles to ~180 bytes on x64. * Alternatives would be nice.**label:** code-design

25516. temp variable to pass constants and flags to shared code

25517. char length of the atom parsed in this loop

25518. -> [... fn x y]

25519. **comment:** ToBoolean() does not require any operations with side effects so * we can do it efficiently. For footprint it would be better to use * duk_js_tobool()

and then push+replace to the result slot.

label: code-design

25520. -> [...]

25521. topmost activation idx_retval is considered garbage, no need to init

25522. * Delayed activation environment record initialization (for functions * with NEWENV). * * The non-delayed initialization is handled by duk_handle_call().

25523. ref.value, ref.this.binding invalidated here by getprop call

25524. * Field accessor macros

25525. [... buf]

25526. idx_base and idx_base+1 get completion value and type

25527. integer number in range

25528. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor). * * Using duk_put_prop() works incorrectly with '__proto__' * if the own property with that name has been deleted. This * does not happen normally, but a clever reviver can trigger * that, see complex reviver case in: test-bug-json-parse-__proto__.js.**label:** code-design

25529. Force interrupt right away if we're paused or in "checked mode". * Step out is handled by callstack unwind.

25530. check stack before interning (avoid hanging temp)

25531. The extra (+4) is tight.

25532. Send version identification and flush right afterwards. Note that * we must write raw, unframed bytes here.

25533. DUK_TOK_STATIC

25534. No pointer compression because pointer is potentially outside of * Duktape heap.

25535. 'null'
25536. pc2line
25537. always normalized
25538. JUMP L1 omitted
25539. **comment:** writable, not configurable
 label: code-design
25540. * Init/assert flags, copying them where appropriate. Some flags * (like NEWENV) are processed separately below.
25541. big endian
25542. entry_top + 3
25543. must match bytecode defines now; build autogenerate?
25544. Although NAMEBINDING is not directly needed for using * function instances, it's needed by bytecode dump/load * so copy it too.
25545. ''
25546. * Automatically generated by genbuiltins.py, do not edit!
25547. * Debug connection read primitives
25548. Note: when parsing a formal list in non-strict context, e.g. * "implements" is parsed as an identifier. When the function is * later detected to be strict, the argument list must be rechecked * against a larger set of reserved words (that of strict mode). * This is handled by duk_parse_func_body(). Here we recognize * whatever tokens are considered reserved in current strictness * (which is not always enough).
25549. **comment:** XXX: optimize with more direct internal access
 label: code-design
25550. __proto__ setter returns 'undefined' on success unlike the * setPrototypeOf() call which returns the target object.
25551. 0x00-0x0f
25552. * String comparison (E5 Section 11.8.5, step 4), which * needs to compare codepoint by codepoint. * * However, UTF-8 allows us to use strcmp directly: the shared * prefix will be encoded identically (UTF-8 has unique encoding) * and the first differing character can be compared with a simple * unsigned byte comparison (which strcmp does). * * This will not work properly for non-xutf-8 strings, but this * is not an issue for compliance.
25553. DUK_TOK_BOR_EQ
25554. Argument must be a string, e.g. a buffer is not allowed.
25555. Note: val is a stable duk_tval pointer. The caller makes * a value copy into its stack frame, so 'tv_val' is not subject * to side effects here.
25556. number of formal arguments
25557. register, constant, or value
25558. Note: intentionally allow leading zeroes here, as the * actual parser will check for them.
25559. Impose a maximum buffer length for now. Restricted artificially to * ensure resize computations or adding a heap header length won't * overflow size_t and that a signed duk_int_t can hold a buffer * length. The limit should be synchronized with DUK_HSTRING_MAX_BYTELEN.
25560. h_code: held in bw_code
25561. * Forward declarations.
25562. heap pointer comparison suffices
25563. Note: 'count' is currently not adjusted by rounding (i.e. the * digits are not "chopped off". That shouldn't matter because * the digit position (absolute or relative) is passed on to the * convert-and-push function.
25564. **comment:** The variable environment for magic variable bindings needs to be * given by the caller and recorded in the arguments object. * * See E5 Section 10.6, the creation of setters/getters. * * The variable environment also provides access to the callee, so * an explicit (internal) callee property is not needed.
 label: code-design
25565. All components default to 0 except day-of-month which defaults * to 1. However, because our internal day-of-month is zero-based, * it also defaults to zero here.
25566. Backtrack.
25567. **comment:** XXX: code duplication
 label: code-design
25568. arg count
25569. **comment:** Switch to caller's setjmp() catcher so that if an error occurs * during error handling, it is always propagated outwards instead * of causing an infinite loop in our own handler.
 label: code-design
25570. [... number] -> [... string]
25571. 'data' (and everything in it) is reachable through h_res now
25572. Fast path.
25573. called as a normal function: return new Date().toString()
25574. -----XX
25575. must be computed after realloc
25576. 'join'
25577. Get the value represented by an duk_ispec to a register or constant. * The caller can control the result by indicating whether or not: * * (1) a constant is allowed (sometimes the caller needs the result to * be in a register) * * (2) a temporary register is required (usually when caller requires * the register to be safely mutable; normally either a bound * register or a temporary register are both OK) * * (3) a forced register target needs to be used * * Bytecode may be emitted to generate the necessary value. The return * value is either a register or a constant.
25578. **comment:** XXX: FP_ZERO check can be removed, the else clause handles it * correctly (preserving sign).
 label: code-design
25579. **comment:** XXX: the string shouldn't appear twice, but we now loop to the * end anyway; if fixed, add a looping assertion to ensure there * is no duplicate.
 label: code-design
25580. re-lookup first char on first loop
25581. [... buf loop (proplist)]
25582. DUK_BUFOBJ_INT16ARRAY
25583. No net refcount changes.
25584. DUK_USE_DATE_FMT_STRFTIME
25585. eat the right paren
25586. **comment:** A straightforward 64-byte lookup would be faster * and cleaner, but this is shorter.
 label: code-design
25587. For this to work, DATEMSK must be set, so this is not very * convenient for an embeddable interpreter.
25588. Fixed buffer; data follows struct, with proper alignment guaranteed by * struct size.
25589. **comment:** * Call handling. * * Main functions are: * * - duk_handle_call_unprotected(): unprotected call to Ecmascript or * Duktape/C function * - duk_handle_call_protected(): protected call to Ecmascript or * Duktape/C function * - duk_handle_safe_call(): make a protected C call within current * activation * - duk_handle_ecma_call_setup(): Ecmascript-to-Ecmascript calls * (not always possible), including tail calls and coroutine resume * * See 'execution.rst'. * * Note: setjmp() and local variables have a nasty interaction, * see execution.rst; non-volatile locals modified after setjmp() * call are not guaranteed to keep their value.
 label: code-design
25590. -1 if disjunction is complex, char length if simple
25591. noblame_fileline
25592. Backtrack to previous slash or start of buffer.
25593. func limits
25594. This is a rare property helper; it sets the global thrower (E5 Section 13.2.3) * setter/getter into an object property. This is needed by the 'arguments' * object creation code, function instance creation code, and Function.prototype.bind().
25595. For anything other than an Error instance, we calculate the error * location directly from the current activation.
25596. **comment:** Workaround for some exotic platforms where NAN is missing * and the expression (0.0 / 0.0) does NOT result in a NaN. * Such platforms use the global 'duk_computed_nan' which must * be initialized at runtime. Use 'volatile' to ensure that * the compiler will actually do the computation and not try * to do

constant folding which might result in the original * problem.

label: code-design

25597. A -> target register (A, A+1) for call setup * (for DUK_OP_CSREGI, 'a' is indirect) * B -> register containing target function (not type checked here)

25598. Return the Buffer to allow chaining: b.fill(0x11).fill(0x22, 3, 5).toString()

25599. **comment:** code_idx: not needed

label: requirement

25600. keep func->h_funcs; inner functions are not reparsed to avoid O(depth^2) parsing

25601. load factor min 25%

25602. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx

25603. Note: peek cannot currently trigger a detach * so the dbg_detaching == 0 assert outside the * loop is correct.

25604. Assume interrupt init/counter are properly initialized here.

25605. * Parse a case or default clause.

25606. **comment:** XXX: when this error is caused by a nonexistent * file given to duk_peval_file() or similar, the * error message is not the best possible.

label: code-design

25607. impose a reasonable exponent limit, so that exp * doesn't need to get tracked using a bigint.

25608. marker value; in E5 2^32-1 is not a valid array index (2^32-2 is highest valid)

25609. maxval

25610. valstack space allocated especially for proxy lookup which does a * recursive property lookup

25611. Accept ASCII strings which conform to identifier requirements * as being emitted without key quotes. Since we only accept ASCII * there's no need for actual decoding: 'p' is intentionally signed * so that bytes >= 0x80 extend to negative values and are rejected * as invalid identifier codepoints.

25612. The Str(key, holder) operation. * * Stack policy: [... key] -> [...]

25613. * Bitstream encoder

25614. -> [sep ToObject(this) len str sep]

25615. Decrement PC if that was requested, this requires a PC sync.

25616. Array properties have exotic behavior but they are concrete, * so no special handling here. * * Arguments exotic behavior (E5 Section 10.6, [[GetOwnProperty]] * is only relevant as a post-check implemented below; hence no * check here.

25617. XXX: request a "last statement is terminal" from duk_parse_stmt() and duk_parse_stmts(); * we could avoid the last RETURN if we could ensure there is no way to get here * (directly or via a jump)

25618. [... key_obj key key flags]

25619. no side effects

25620. INACTIVE: no activation, single function value on valstack

25621. Equality may be OK but >length not. Checking * this explicitly avoids some overflow cases * below.

25622. this cannot initiate a detach

25623. done so that duk_mark_heapHdr() works correctly

25624. if highest bit of a register number is set, it refers to a constant instead

25625. **comment:** Slow variants, call to a helper to reduce code size. * Can be used explicitly when size is always more important than speed.

label: code-design

25626. DUK_USE_REFERENCE_COUNTING

25627. Caller ensures space for at least DUK_JSON_MAX_ESC_LEN.

25628. byte offset to buf

25629. * The catch variable must be updated to reflect the new allocated * register for the duration of the catch clause. We need to store * and restore the original value for the varmap entry (if any).

25630. **comment:** unused now

label: code-design

25631. **comment:** stack[0] = callback * stack[1] = thisArg * stack[2] = object * stack[3] = ToUInt32(length) (unused, but avoid unnecessary pop) * stack[4] = result array (or undefined)

label: code-design

25632. still in prologue

25633. heap->dbg_processing: keep on purpose to avoid debugger re-entry in detaching state

25634. fixed-format termination conditions

25635. DUK_TOK_ELSE

25636. * Add line number to a compiler error.

25637. array of function templates: [func1, offset1, line1, func2, offset2, line2] * offset/line points to closing brace to allow skipping on pass 2

25638. Ensure copy is covered by underlying buffers.

25639. longjmp state

25640. * Note: * * - Intentionally attempt (empty) match at char_offset == k_input->clen * * - Negative char_offsets have been eliminated and char_offset is duk_uint32_t * -> no need or use for a negative check

25641. Expression, terminates at a ')

25642. can't happen when keeping current stack size

25643. Special handling for CALL/NEW/MPUTOBJ/MPUTARR shuffling. * For each, slot B identifies the first register of a range * of registers, so normal shuffling won't work. Instead, * an indirect version of the opcode is used.

25644. DUK_TOK_FOR

25645. Miscellaneous.

25646. -> [... retval]

25647. If a setter/getter is missing (undefined), the descriptor must * still have the property present with the value 'undefined'.

25648. -> [timeval this]

25649. **comment:** Clean up stack

label: code-design

25650. **comment:** dump all allocated entries, unused entries print as 'unused', * note that these may extend beyond current 'length' and look * a bit funny.

label: code-design

25651. unwind to 'yield' caller

25652. How much stack to require on entry to object/array decode

25653. (e.g. k=3, digits=2 -> "12X")

25654. Note: changing from writable to non-writable is OK

25655. unwinds valstack, updating refcounts

25656. For JX, expressing the whole unsigned 32-bit range matters.

25657. r <- 2 * f

25658. currently active breakpoints: NULL term, borrowed pointers

25659. conversions, coercions, comparison, etc

25660. [key val] -> [key]

25661. use stack allocated buffer to ensure reachability in errors (e.g. intern error)

25662. **comment:** * Manually optimized double-to-fastint downgrade check. * * This check has a large impact on performance, especially for fastint * slow paths, so must be changed carefully. The code should probably be * optimized for the case where the result does not fit into a fastint, * to minimize the penalty for "slow path code" dealing with fractions etc. * * At least on one tested soft float ARM platform double-to-int64 coercion * is very slow (and sometimes produces incorrect results, see self tests). * This algorithm combines a fastint compatibility check and extracting the * integer value from an IEEE double for setting the tagged fastint. For * other platforms a more naive approach might be better. * * See doc/fastint.rst for details.

label: code-design

25663. * Misc support functions

25664. ch is a literal character here or -1 if parsed entity was * an escape such as "'s".
25665. Code emission flags, passed in the 'opcode' field. Opcode + flags * fit into 16 bits for now, so use duk_small_uint.t.
25666. array of formal argument names (-> _Formals)
25667. [...] func this arg1 ... argN]
25668. * Simple atom ** If atom_char_length is zero, we'll have unbounded execution time for e.g. */()*/x/.exec('x'). We can't just skip the match because it might have some * side effects (for instance, if we allowed captures in simple atoms, the * capture needs to happen). The simple solution below is to force the * quantifier to match at most once, since the additional matches have no effect. ** With a simple atom there can be no capture groups, so no captures need * to be reset.
25669. key encountered as a getter
25670. **comment:** XXX: typing
 label: code-design
25671. catches EOF (NUL)
25672. 'act' is no longer accessed, scanbuild fix
25673. **comment:** XXX: Array size is known before and (2 * re_ctx.nsaved) but not taken * advantage of now. The array is not compacted either, as regexp match * objects are usually short lived.
 label: code-design
25674. Ecmascript activation + Duktape.Thread.yield() activation
25675. **comment:** Presence of 'fun' is config based, there's a marginal performance * difference and the best option is architecture dependent.
 label: code-design
25676. [...] RegExp]
25677. DUK_TOK_DECREMENT
25678. [...] errhandler undefined errval]
25679. **comment:** The target for this LDCONST may need output shuffling, but we assume * that 'pc_ldconst' will be the LDCONST that we can patch later. This * should be the case because there's no input shuffling. (If there's * no catch clause, this LDCONST will be replaced with a NOP.)
 label: code-design
25680. no wrap assuming h_bufobj->length is valid
25681. Nominal size check.
25682. register bound variables are non-configurable -> always false
25683. -> [obj trap handler target]
25684. reasonable output estimate
25685. Note: DST adjustment is determined using UTC time. * If 'd' is NaN, tzoffset will be 0.
25686. * Character and charcode access
25687. nargs == 2 so we can pass a callstack level to eval().
25688. First pass parse is very lenient (e.g. allows '1.2.3') and extracts a * string for strict number parsing.
25689. normal valued properties
25690. '\xffLexenv'
25691. Module loading failed. Node.js will forget the module * registration so that another require() will try to load * the module again. Mimic that behavior.
25692. **comment:** Note: for an accessor without getter, falling through to * check for "caller" exotic behavior is unnecessary as * "undefined" will never activate the behavior. But it does * no harm, so we'll do it anyway.
 label: code-design
25693. e.g. 12.3 -> digits="123" k=2 -> 1.23e1
25694. **comment:** XXX: there is room to use a shared helper here, many built-ins * check the 'this' type, and if it's an object, check its class, * then get its internal value, etc.
 label: code-design
25695. variable map for pass 2 (identifier -> register number or null (unmapped))
25696. buffer pointer is to an externally allocated buffer
25697. 'Uint32Array'
25698. 'with'
25699. **comment:** * (Re)initialize the temporary byte buffer. May be called extra times * with little impact.
 label: code-design
25700. callstack index of related activation
25701. non-standard
25702. may be safe, or non-safe depending on flags
25703. activation prevents yield (native call or "new")
25704. **comment:** XXX: expensive, but numconv now expects to see a string
 label: code-design
25705. DUK_FLD_VARINT; not relevant here
25706. **comment:** * Note: use indirect realloc variant just in case mark-and-sweep * (finalizers) might resize this same buffer during garbage * collection.
 label: code-design
25707. eat 'try'
25708. 1110 xxxx 10xx xxxx 10xx xxxx
25709. Note: it's nice if size is 2^N (at least for 32-bit platforms).
25710. Quite heavy assert: check valstack policy. Improper * shuffle instructions can write beyond valstack_top/end * so this check catches them in the act.
25711. Expects 'this' at top of stack on entry.
25712. The token in the switch has already been eaten here
25713. currently all labels accept a break, so no explicit check for it now
25714. [arg result]
25715. got lineterm preceding non-whitespace, non-lineterm token
25716. intentional overlap with earlier memzero
25717. Allow exponent
25718. '\xffSource'
25719. read-only object, in code section
25720. **comment:** Slice and accessor information. ** Because the underlying buffer may be dynamic, these may be * invalidated by the buffer being modified so that both offset * and length should be validated before every access. Behavior * when the underlying buffer has changed doesn't need to be clean: * virtual 'length' doesn't need to be affected, reads can return * zero/NaN, and writes can be ignored. ** Note that a data pointer cannot be precomputed because 'buf' may * be dynamic and its pointer unstable.
 label: code-design
25721. Shared object part.
25722. **comment:** XXX: there are language sensitive rules for the ASCII range. * If/when language/locale support is implemented, they need to * be implemented here for the fast path. There are no context * sensitive rules for ASCII range.
 label: code-design
25723. DUK_OP_RETURN flags in A
25724. DUK_TOK_ADD
25725. allocate to reg only (not const)
25726. **comment:** not strictly necessary, but avoids "uninitialized variable" warnings
 label: code-design
25727. * Augment an error at throw time; allow a user error handler (if defined) * to process/replace the error. The error to be augmented is at the * stack top.
25728. **comment:** This macro works when a regconst field is 9 bits, [0,0x1ff]. Adding * DUK_LIKELY/DUK_UNLIKELY increases code footprint and doesn't seem to * improve performance on x64 (and actually harms performance in some tests).

label: code-design

25729. requested number of output digits; 0 = free-format
25730. * Shared readfield and writefield methods * * The readfield/writefield methods need support for endianness and field * types. All offsets are byte based so no offset shifting is needed.
25731. Gap in array; check for inherited property, * bail out if one exists. This should be enough * to support gappy arrays for all practical code.
25732. **comment:** XXX: could improve bufwriter handling to write multiple codepoints * with one ensure call but the relative benefit would be quite small.
label: code-design
25733. **comment:** * Delete references to given hstring from the heap string cache. * * String cache references are 'weak': they are not counted towards * reference counts, nor serve as roots for mark-and-sweep. When an * object is about to be freed, such references need to be removed.
label: code-design
25734. DUK_TOK_EOF
25735. **comment:** XXX: signalling the need to shrink check (only if unwound)
label: code-design
25736. XXX: duk_dup_unvalidated(ctx, -2) etc.
25737. fill new entries with -unused- (required, gc reachable)
25738. -> [O toString O]
25739. accept anything, expect first value (EOF will be * caught by key parsing below.
25740. unknown, ignore
25741. * Note: fmax() does not match the E5 semantics. E5 requires * that if -any- input to Math.max() is a NaN, the result is a * NaN. fmax() will return a NaN only if both- inputs are NaN. * Same applies to fmin(). * * Note: every input value must be coerced with ToNumber(), even * if we know the result will be a NaN anyway: ToNumber() may have * side effects for which even order of evaluation matters.
25742. [... v1 v2 name filename]
25743. !DUK_USE_PANIC_HANDLER
25744. **comment:** XXX: inlined DECREF macro would be nice here: no NULL check, * refzero queueing but no refzero algorithm run (= no pointer * instability), inline code.
label: code-design
25745. expected ival
25746. **comment:** A 16-bit listlen makes sense with 16-bit heap pointers: there * won't be space for 64k strings anyway.
label: code-design
25747. * Finalize refcounts for heap elements just about to be freed. * This must be done for all objects before freeing to avoid any * stale pointer dereferences. * * Note that this must deduce the set of objects to be freed * identically to duk_sweep_heap().
25748. no sce, or sce scan not best
25749. caller must have decref'd values above new_a_size (if that is necessary)
25750. [... buf loop (proplist) (gap)]
25751. **comment:** XXX: duk_hobject_hasprop() would be correct for * non-Proxy objects too, but it is about ~20-25% * slower at present so separate code paths for * Proxy and non-Proxy now.
label: code-design
25752. Coercion may be needed, the helper handles that by pushing the * tagged values to the stack.
25753. checked by caller
25754. 'Int16Array'
25755. res->strs16[] is zeroed and zero decodes to NULL, so no NULL inits.
25756. entry part must not contain any configurable properties, or * writable properties (if is_frozen).
25757. thread; minimizes argument passing
25758. step 4.d
25759. **comment:** * The Number constructor uses ToNumber(arg) for number coercion * (coercing an undefined argument to NaN). However, if the * argument is not given at all, +0 must be used instead. To do * this, a vararg function is used.
label: code-design
25760. one operator (= assign)
25761. no need to init reg, it will be undefined on entry
25762. string objects must not created without internal value
25763. eat backslash on entry
25764. * Cached module check. * * If module has been loaded or its loading has already begun without * finishing, return the same cached value ('exports'). The value is * registered when module load starts so that circular references can * be supported to some extent.
25765. * Setup a preliminary activation and figure out nargs/nregs. * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
25766. **comment:** XXX: Rework to use an always-inline function?
label: code-design
25767. **comment:** * Common heap header * * All heap objects share the same flags and refcount fields. Objects other * than strings also need to have a single or double linked list pointers * for insertion into the "heap allocated" list. Strings are held in the * heap-wide string table so they don't need link pointers. * * Technically, 'h_refcount' must be wide enough to guarantee that it cannot * wrap (otherwise objects might be freed incorrectly after wrapping). This * means essentially that the refcount field must be as wide as data pointers. * On 64-bit platforms this means that the refcount needs to be 64 bits even * if an 'int' is 32 bits. This is a bit unfortunate, and compromising on * this might be reasonable in the future. * * Heap header size on 32-bit platforms: 8 bytes without reference counting, * 16 bytes with reference counting.
label: code-design
25768. We're a varargs function because we need to detect whether * initialValue was given or not.
25769. cannot be strict (never mapped variables)
25770. Avoid recursive re-entry; enter when we're attached or * detaching (to finish off the pending detach).
25771. DUK_DEBUGGER_H_INCLUDED
25772. Maximum length of CommonJS module identifier to resolve. Length includes * both current module ID, requested (possibly relative) module ID, and a * slash in between.
25773. **comment:** * Property count check. This is the only point where we ensure that * we don't get more (allocated) property space that we can handle. * There aren't hard limits as such, but some algorithms fail (e.g. * finding next higher prime, selecting hash part size) if we get too * close to the 4G property limit. * * Since this works based on allocation size (not actually used size), * the limit is a bit approximate but good enough in practice.
label: code-design
25774. This loop intentionally does not ensure characters are valid * ([0-9a-fA-F]) because the hex decode call below will do that.
25775. 0 if no used entries
25776. * Basic double / byte union memory layout.
25777. copy values (no overlap even if to_ctx == from_ctx; that's not * allowed now anyway)
25778. Use match charlen instead of bytelen, just in case the input and * match codepoint encodings would have different lengths.
25779. **comment:** Count actually used array part entries and array minimum size. * NOTE: 'out_min_size' can be computed much faster by starting from the * end and breaking out early when finding first used entry, but this is * not needed now.
label: code-design
25780. if new_p == NULL, all of these pointers are NULL
25781. -> [... obj retrval/error]
25782. [... holder key] -> [... holder]
25783. 'this' binding is just before current activation's bottom
25784. **comment:** * Error handling macros, assertion macro, error codes. * * There are three level of 'errors': * * 1. Ordinary errors, relative to a thread, cause a longjmp, catchable. * 2. Fatal errors, relative to a heap, cause fatal handler to be called. * 3. Panic errors, unrelated to a heap and cause a process exit. * * Panics are used by the default fatal error handler and by debug code * such as assertions. By providing a proper fatal error handler, user * code can avoid panics in non-debug builds.

label: code-design

25785. Arguments can be: [source? filename? &comp_args] so that * nargs is 1 to 3. Call site encodes the correct nargs count * directly into flags.
25786. a
25787. Allow leading zeroes (e.g. "0123" -> "123")
25788. * Default clause.
25789. d <- (quotient (* r B) s) (in range 0...B-1)
25790. **comment:** XXX: There used to be a shared log buffer here, but it was removed * when dynamic buffer spare was removed. The problem with using * bufwriter is that, without the spare, the buffer gets passed on * as an argument to the raw() call so it'd need to be resized * (reallocated) anyway. If raw() call convention is changed, this * could be made more efficient.
label: code-design
25791. Commands initiated by debug client.
25792. next loop requires a comma
25793. 0x40...0x4f
25794. 'string'
25795. match == search string, by definition
25796. **comment:** Shared handling for encode/decode argument. Fast path handling for * buffer and string values because they're the most common. In particular, * avoid creating a temporary string or buffer when possible.
label: code-design
25797. 32-bit x 11-bit = 43-bit, fits accurately into a double
25798. Opcode only emitted by compiler when debugger * support is enabled. Ignore it silently without * debugger support, in case it has been loaded * from precompiled bytecode.
25799. eat closing brace
25800. Behavior for constructor and non-constructor call is * the same except for augmenting the created error. When * called as a constructor, the caller (duk_new()) will handle * augmentation; when called as normal function, we need to do * it here.
25801. Encode into a single opcode (18 bits can encode 1-2 bytes + length indicator)
25802. defined(DUK_USE_DATE_NOW_WINDOWS) || defined(DUK_USE_DATE_TZO_WINDOWS)
25803. without finally, the second jump slot is used to jump to end of stmt
25804. slot B is a target (default: source)
25805. fastint constants etc
25806. * Unicode codepoints above U+FFFF are encoded as surrogate * pairs here. This ensures that all CESU-8 codepoints are * 16-bit values as expected in Ecmascript. The surrogate * pairs always get a 3-byte encoding (each) in CESU-8. * See: http://en.wikipedia.org/wiki/Surrogate_pair * * 20-bit codepoint, 10 bits (A and B) per surrogate pair: * * x = 0b00000000 0000AAAA AAAAAAAB BBBB BBBB * sp1 = 0b110110AA AAAAAAAA (0xd800 + ((x >> 10) & 0x3ff)) * sp2 = 0b110111BB BBBB BBBB (0xdc00 + (x & 0x3ff)) * * Encoded into CESU-8: * * sp1 -> 0b1101101 (0xe0 + ((sp1 >> 12) & 0x0f)) * -> 0b1010AAAA (0x80 + ((sp1 >> 6) & 0x3f)) * -> 0b10AAAAAA (0x80 + (sp1 & 0x3f)) * sp2 -> 0b1101101 (0xe0 + ((sp2 >> 12) & 0x0f)) * -> 0b1011BBBB (0x80 + ((sp2 >> 6) & 0x3f)) * -> 0b10BBBBBB (0x80 + (sp2 & 0x3f)) * * Note that 0x10000 must be subtracted first. The code below * avoids the sp1, sp2 temporaries which saves around 20 bytes * of code.
25807. Duktape.modLoaded
25808. We could do this operation without caller updating bw_ctx->ptr, * but by writing it back here we can share code better.
25809. Note: if encoding ends by hitting end of input, we don't check that * the encoding is valid, we just assume it is.
25810. **comment:** XXX: call method tail call?
label: code-design
25811. w.../
25812. always provideThis=true
25813. Note: only numbered indices are relevant, so arr_idx fast reject * is good (this is valid unless there are more than 4***32-1 arguments).
25814. * Thread state and calling context checks
25815. token type (with reserved word identification)
25816. **comment:** XXX: there's no explicit recursion bound here now. For the average * qsort recursion depth O(log n) that's not really necessary: e.g. for * 2**32 recursion depth would be about 32 which is OK. However, qsort * worst case recursion depth is O(n) which may be a problem.
label: code-design
25817. verified at beginning
25818. **comment:** Fall back to the initial (original) Object.toString(). We don't * currently have pointers to the built-in functions, only the top * level global objects (like "Array") so this is now done in a bit * of a hacky manner. It would be cleaner to push the (original) * function and use duk_call_method().
label: code-design
25819. XXX: duk_push_uint_string()
25820. Built-in providers
25821. **comment:** XXX: could add flags for "is valid CESU-8" (Ecmascript compatible strings), * "is valid UTF-8", "is valid extended UTF-8" (internal strings are not, * regexp bytecode is), and "contains non-BMP characters". These are not * needed right now.
label: code-design
25822. pop final_cons
25823. **comment:** When ptr == NULL, the format argument is unused.
label: code-design
25824. To convert a heap stridx to a token number, subtract * DUK_STRIDX_START_RESERVED and add DUK_TOK_START_RESERVED.
25825. ANSI C typing
25826. These pointers are at the start of the struct so that they pack * nicely. Mixing pointers and integer values is bad on some * platforms (e.g. if int is 32 bits and pointers are 64 bits).
25827. 1=little endian
25828. * "IdentifierStart" production check.
25829. * Compiler state
25830. **comment:** XXX: this is awkward as we use an internal method which doesn't handle * extensibility etc correctly. Basically we'd want to do [[DefineOwnProperty]] * or Object.defineProperty() here.
label: code-design
25831. **comment:** * Entries part is a bit more complex
label: code-design
25832. * On first pass, perform actual parsing. Remember valstack top on entry * to restore it later, and switch to using a new function in comp_ctx.
25833. nop: insert top to top
25834. h_match is borrowed, remains reachable through match_obj
25835. 0 = uppercase, 32 = lowercase (= 'a' - 'A')
25836. catch jump
25837. 3: toLocaleString
25838. not reachable
25839. leave 'cat' as top catcher (also works if catchstack exhausted)
25840. +1 for opcode
25841. 'varname' is in stack in this else branch, leaving an unbalanced stack below, * but this doesn't matter now.
25842. Not halfway, round to nearest.
25843. remove in_val
25844. idx_argbase
25845. Strict by default.
25846. fixed precision and zero padding would be required

25847. Ensuring (reserving) space.
25848. flags: "m"
25849. 'setPrototypeOf'
25850. return 1 to allow callers to tail call
25851. * Not found (even in global object)
25852. * Reallocate memory with garbage collection, using a callback to provide * the current allocated pointer. This variant is used when a mark-and-sweep * (e.g. finalizers) might change the original pointer.
25853. 'number'
25854. label id allocation (running counter)
25855. **comment:** * Shared error messages: declarations and macros ** Error messages are accessed through macros with fine-grained, explicit * error message distinctions. Concrete error messages are selected by the * macros and multiple macros can map to the same concrete string to save * on code footprint. This allows flexible footprint/verbosity tuning with * minimal code impact. There are a few limitations to this approach: * (1) switching between plain messages and format strings doesn't work * conveniently, and (2) conditional strings are a bit awkward to handle. ** Because format strings behave differently in the call site (they need to * be followed by format arguments), they have a special prefix (DUK_STR_FMT_ * and duk_str_fmt_). ** On some compilers using explicit shared strings is preferable; on others * it may be better to use straight literals because the compiler will combine * them anyway, and such strings won't end up unnecessarily in a symbol table.
label: code-design
25856. **comment:** XXX: how to figure correct size?
label: code-design
25857. **comment:** * See the following documentation for discussion: ** doc/execution.rst: control flow details ** Try, catch, and finally "parts" are Blocks, not Statements, so * they must always be delimited by curly braces. This is unlike e.g. * the if statement, which accepts any Statement. This eliminates any * questions of matching parts of nested try statements. The Block * parsing is implemented inline here (instead of calling out). ** Finally part has a 'let scoped' variable, which requires a few kinks * here.
label: code-design
25858. op_flags
25859. # argument registers target function wants (< 0 => never for ecma calls)
25860. Different calling convention than above used because the helper * is shared.
25861. equality not actually possible
25862. **comment:** XXX: make this lenience dependent on flags or strictness?
label: code-design
25863. [... func this arg1 ... argN envobj]
25864. 32: setDate
25865. slot C is a target (default: source)
25866. -> [...] lval rval new_rval]
25867. Sealed and frozen objects cannot gain any more properties, * so this is a good time to compact them.
25868. Parse a single variable declaration (e.g. "i" or "i=10"). A leading 'var' * has already been eaten. These is no return value in 'res', it is used only * as a temporary. ** When called from 'for-in' statement parser, the initializer expression must * not allow the 'in' token. The caller supply additional expression parsing * flags (like DUK__EXPR_FLAG_REJECT_IN) in 'expr_flags'. ** Finally, out_rc_varname and out_reg_varbind are updated to reflect where * the identifier is bound: * * If register bound: out_reg_varbind >= 0, out_rc_varname == 0 (ignore) * If not register bound: out_reg_varbind < 0, out_rc_varname >= 0 * * These allow the caller to use the variable for further assignment, e.g. * as is done in 'for-in' parsing.
25869. sanity
25870. deal with negative values
25871. Unlike for negative arguments, some call sites * want length to be clamped if it's positive.
25872. create bound function object
25873. maximum bytecode copies for {n,m} quantifiers
25874. [...] lval rval]
25875. **comment:** Note: Boolean prototype's internal value property is not writable, * but duk_xdef_prop_stridx() disregards the write protection. Boolean * instances are immutable. * * String and buffer special behaviors are already enabled which is not * ideal, but a write to the internal value is not affected by them.
label: code-design
25876. use duk_double_union as duk_tval directly
25877. [...] obj finalizer obj heapDestruct] -> [...] obj retval]
25878. **comment:** * Assignments are right associative, allows e.g. * a = 5; * a += b = 9; // same as a += (b = 9) * -> expression value 14, a = 14, b = 9 * * Right associativity is reflected in the BP for recursion, * "-1" ensures assignment operations are allowed. ** XXX: just use DUK__BP_COMMA (i.e. no need for 2-step bp levels)?
label: code-design
25879. don't increase 'count'
25880. module.exports = exports
25881. simulate alloc failure on every alloc (except when mark-and-sweep is running)
25882. must relookup act in case of side effects
25883. **comment:** XXX: take advantage of val being unsigned, no need to mask
label: code-design
25884. status booleans
25885. identifier handling
25886. duk_unicode_caseconv_lc()
25887. DUK_TOK_PROTECTED
25888. target
25889. reserved words: future reserved words
25890. Ensure space for maximum multi-character result; estimate is overkill.
25891. * Getters. ** Implementing getters is quite easy. The internal time value is either * NaN, or represents milliseconds (without fractions) from Jan 1, 1970. * The internal time value can be converted to integer parts, and each * part will be normalized and will fit into a 32-bit signed integer. ** A shared native helper to provide all getters. Magic value contains * a set of flags and also packs the date component index argument. The * helper provides: * * getFullYear() * getUTCFullYear() * getMonth() * getUTCMonth() * getDate() * getUTCDate() * getDay() * getUTCDay() * getHours() * getUTCHours() * getMinutes() * getUTCMinutes() * getSeconds() * getUTCSeconds() * getMilliseconds() * getUTCMilliseconds() * getYear() * * Notes: * * - Date.prototype.getDate(): 'date' means day-of-month, and is * zero-based in internal calculations but public API expects it to * be one-based. * * - Date.prototype.getTime() and Date.prototype.valueOf() have identical * behavior. They have separate function objects, but share the same C * function (duk_bi_date_prototype_value_of).
25892. * Pre resize assertions.
25893. In-place unary operation.
25894. Note: expect that caller has already eaten the left paren
25895. implicit leading one
25896. **comment:** * Array index and length ** Array index: E5 Section 15.4 * Array length: E5 Section 15.4.5.1 steps 3.c - 3.d (array length write) ** The DUK_HSTRING_GET_ARRIDX_SLOW() and DUK_HSTRING_GET_ARRIDX_FAST() macros * call duk_js_to_arrayindex_string_helper().
label: code-design
25897. * Internal helpers for managing object 'length'
25898. biased exp == 0 -> denormal, exp -1022
25899. Oxdeadbeef in decimal
25900. strings up to the this length are not cached
25901. DUK_HEAPHDR_H_INCLUDED
25902. [...] key_obj key val]

25903. Checking callability of the immediate target * is important, same for constructability. * Checking it for functions down the bound * function chain is not strictly necessary * because .bind() should normally reject them. * But it's good to check anyway because it's * technically possible to edit the bound function * chain via internal keys.

25904. **comment:** * Three possible element formats: * 1) PropertyName : AssignmentExpression * 2) get PropertyName () { FunctionBody } * 3) set PropertyName (PropertySetParameterList) { FunctionBody } * * PropertyName can be IdentifierName (includes reserved words), a string * literal, or a number literal. Note that IdentifierName allows 'get' and * 'set' too, so we need to look ahead to the next token to distinguish: ** { get : 1 } * * and * * { get foo() { return 1 } } * { get get() { return 1 } } // 'get' as getter propertyname * * Finally, a trailing comma is allowed. * * Key name is coerced to string at compile time (and ends up as a * string constant) even for numeric keys (e.g. "{1:foo}"). * These could be emitted using e.g. LDINT, but that seems hardly * worth the effort and would increase code size.

label: code-design

25905. keep func->h_argnames; it is fixed for all passes

25906. num entries of new func at entry

25907. One digit octal escape, digit validated.

25908. Use result value as is.

25909. step 15.a

25910. formal arg limits

25911. Use the approach described in "Remarks" of FileTimeToLocalFileTime: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx)

25912. [offset noAssert], when ftype != DUK_FLD_VARINT

25913. bufwriter for temp accumulation

25914. normal comparison; known: * - both x and y are not NaNs (but one of them can be) * - both x and y are not zero (but one of them can be) * - x and y may be denormal or infinite

25915. -1 if invalid

25916. convenience helpers

25917. * Use Object.defineProperty() helper for the actual operation.

25918. Ensure compact use of temps.

25919. * Object handling: property access and other support functions.

25920. allocated from valstack (fixed buffer)

25921. Unlike most built-ins, the internal [[PrimitiveValue]] of a Date * is mutable.

25922. DUK_TVAL_H_INCLUDED

25923. * Retry with several GC attempts. Initial attempts are made without * emergency mode; later attempts use emergency mode which minimizes * memory allocations forcibly.

25924. last component

25925. lexing (tokenization) state (contains two valstack slot indices)

25926. * Update an existing property of the base object.

25927. update length (curr points to length, and we assume it's still valid)

25928. in-place setup

25929. Currently all built-in native functions are strict. * duk_push_c_function() now sets strict flag, so * assert for it.

25930. The lo/hi indices may be crossed and hi < 0 is possible at entry.

25931. * Init compilation context

25932. add E

25933. '\xffThread'

25934. Must use memmove() because copy area may overlap (source and target * buffer may be the same, or from different slices).

25935. * Special name handling

25936. If function creation fails due to out-of-memory, the data buffer * pointer may be NULL in some cases. That's actually possible for * GC code, but shouldn't be possible here because the incomplete * function will be unwound from the value stack and never instantiated.

25937. A -> register of target object * B -> first register of key/value pair list * C -> number of key/value pairs

25938. **comment:** XXX: this is a major target for size optimization

label: code-design

25939. 'constructor' property for all built-in objects (which have it) has attributes: * [[Writable]] = true, * [[Enumerable]] = false, * [[Configurable]] = true

25940. XXX: check num_args

25941. pop 'name'

25942. Here 'p' always points to the start of a term. * * We can also unconditionally reset q_last here: if this is * the last (non-empty) term q_last will have the right value * on loop exit.

25943. * All done, switch properties ('p') allocation to new one.

25944. * Duktape.Buffer, Node.js Buffer, and Khronos/ES6 TypedArray built-ins

25945. decode '%xx' to "%xx" if decoded char in reserved set

25946. **comment:** XXX: Finalizer lookup should traverse the prototype chain (to allow * inherited finalizers) but should not invoke accessors or proxy object * behavior. At the moment this lookup will invoke proxy behavior, so * caller must ensure that this function is not called if the target is * a Proxy.

label: code-design

25947. Maximum number of breakpoints. Only breakpoints that are set are * consulted so increasing this has no performance impact.

25948. any other printable -> as is

25949. Called by duk_hstring.h macros

25950. step type: none, step into, step over, step out

25951. **comment:** XXX: faster implementation

label: code-design

25952. cleared before entering catch part

25953. Clamp an input byte length (already assumed to be within the nominal * duk_hbufferobject 'length') to the current dynamic buffer limits to * yield a byte length limit that's safe for memory accesses. This value * can be invalidated by any side effect because it may trigger a user * callback that resizes the underlying buffer.

25954. **comment:** XXX: ignored now

label: code-design

25955. **comment:** * Tagged type definition (duk_tval) and accessor macros. * * Access all fields through the accessor macros, as the representation * is quite tricky. * * There are two packed type alternatives: an 8-byte representation * based on an IEEE double (preferred for compactness), and a 12-byte * representation (portability). The latter is needed also in e.g. * 64-bit environments (it usually pads to 16 bytes per value). * * Selecting the tagged type format involves many trade-offs (memory * use, size and performance of generated code, portability, etc), * see doc/types.rst for a detailed discussion (especially of how the * IEEE double format is used to pack tagged values). * * NB: because macro arguments are often expressions, macros should * avoid evaluating their argument more than once.

label: code-design

25956. * Automatically generated by extract_caseconv.py, do not edit!

25957. * Support functions for duk_heap.

25958. DUK_HCOMPILEDFUNCTION_H_INCLUDED

25959. write_to_array_part:

25960. **comment:** * Parse variant 1 or 2. The first part expression (which differs * in the variants) has already been parsed and its code emitted. * * reg_temps + 0: unused * reg_temps + 1: unused

label: code-design

25961. E5 Section 15.5.5.1

25962. own pointer

25963. Note: although there is no 'undefined' literal, undefined * values can occur during compilation as a result of e.g. * the 'void' operator.

25964. Note: this may cause a corner case situation where a finalizer * may see a currently reachable activation whose 'func' is NULL.

25965. # total registers target function wants on entry (< 0 => never for ecma calls)
25966. bytecode has a stable pointer
25967. * Thread builts
25968. overwrite str1
25969. **comment:** XXX: exposed duk_debug_read_buffer
 label: code-design
25970. DUK_JS_BYTECODE_H_INCLUDED
25971. Convert a duk_tval number (caller checks) to a 32-bit index. Returns * DUK__NO_ARRAY_INDEX if the number is not whole or not a valid array * index.
25972. SCANBUILD: scan-build produces a NULL pointer dereference warning * below; it never actually triggers because holder is actually never * NULL.
25973. Set up pointers to the new property area: this is hidden behind a macro * because it is memory layout specific.
25974. no need for refcount update
25975. **comment:** Zero 'count' is also allowed to make call sites easier. * Arithmetic in bytes generates better code in GCC.
 label: code-design
25976. compiler ensures this
25977. [... old_result] -> [... result]
25978. **comment:** This shouldn't be necessary, but check just in case * to avoid any chance of overruns.
 label: code-design
25979. At the moment Buffer(<str>) will just use the string bytes as * is (ignoring encoding), so we return the string length here * unconditionally.
25980. * concat()
25981. different memory layout, alloc size, and init
25982. t2 = (* (+ r m+) B)
25983. exponent non-negative (and thus not minimum exponent)
25984. default: NULL, length 0
25985. env is closed, should be missing _Callee, _Thread, _Regbase
25986. **comment:** The E5.1 algorithm checks whether or not a decoded codepoint * is below 0x80 and perhaps may be in the "reserved" set. * This seems pointless because the single byte UTF-8 case is * handled separately, and non-shortest encodings are rejected. * So, 'cp' cannot be below 0x80 here, and thus cannot be in * the reserved set.
 label: code-design
25987. force_flag
25988. Used for minimal 'const': initializer required.
25989. * Table for base-64 encoding
25990. Note: no allocation pressure, no need to check refcounts etc
25991. args incorrect
25992. write on next loop
25993. * Helper to format a time value into caller buffer, used by logging. * 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.
25994. **comment:** XXX: the returned value is exotic in ES6, but we use a * simple object here with no prototype. Without a prototype, * [[DefaultValue]] coercion fails which is abit confusing. * No callable check/handling in the current Proxy subset.
 label: code-design
25995. 2nd related value (type specific)
25996. **comment:** Extraop arithmetic opcodes must have destination same as * first source. If second source matches destination we need * a temporary register to avoid clobbering the second source. ** XXX: change calling code to avoid this situation in most cases.
 label: code-design
25997. cannot grow
25998. 0-indexOf, 1=lastIndexOf
25999. bump up "allocated" reg count, just in case
26000. Strict standard behavior, ignore trailing elements for * result 'length'.
26001. For errors a string coerced result is most informative * right now, as the debug client doesn't have the capability * to traverse the error object.
26002. DUK_TOK_IF
26003. **comment:** NOTE: This is a bit fragile. It's important to ensure that * duk_debug_process_messages() never throws an error or * act->curr_pc will never be reset.
 label: code-design
26004. Working with the pointer and current size.
26005. object has any exotic behavior(s)
26006. DUK_USE_STRTAB_PROBE
26007. written by a previous RESUME handling
26008. **comment:** * Macro support functions (use only macros in calling code)
 label: code-design
26009. string 1 of token (borrowed, stored to ctx->slot1_idx)
26010. **comment:** XXX: more accurate?
 label: code-design
26011. Note: assume array part is comprehensive, so that either * the write goes to the array part, or we've abandoned the * array above (and will not come here).
26012. level string
26013. sufficient for keeping temp reg numbers in check
26014. All other object types.
26015. **comment:** Flip highest bit of each byte which changes * the bit pattern 10xxxxxx into 00xxxxxx which * allows an easy bit mask test.
 label: test
26016. mixed endian
26017. min_new_size
26018. length in bytes (not counting NUL term)
26019. No finalizers for ROM objects
26020. last element that was left in the heap
26021. compact; no need to seal because object is internal
26022. 0x90...0x9f
26023. const flag for C
26024. [... obj] -> [...]
26025. number of pairs in current MPUTOBJ set
26026. escaped NonEscapeCharacter
26027. Module id requested
26028. Empty searchstring always matches; cpos must be clamped here. * (If q_blen were < 0 due to clamped coercion, it would also be * caught here.)
26029. Never executed if new size is smaller.
26030. 'protected'
26031. because arg count is 1
26032. 'if'
26033. finally free the struct itself
26034. !DUK_USE_PARANOID_ERRORS
26035. Array or Array-like
26036. coerce towards zero
26037. must not be extensible
26038. **comment:** We've ensured space for one escaped input; then * bail out and recheck (this makes escape handling * quite slow but it's uncommon).

label: code-design

26039. mark-and-sweep: finalizable (on current pass)

26040. [...] replacer match [captures] match_char_offset input]

26041. DUK_OP_TRYCATCH flags in A

26042. single string token

26043. Use byte copy.

26044. Most common cases first.

26045. no point in supporting encodings of 5 or more bytes

26046. '|'

26047. use_prev_pc

26048. We need ' nbytes' even for a failed offset; return value must be * (offset + nbytes) even when write fails due to invalid offset.

26049. Return value handling.

26050. Enable DUKFUNC exotic behavior once properties are set up.

26051. duk_bi_duk_object_yield() and duk_bi_duk_object_resume() ensure all of these are met

26052. Assumes that caller has normalized NaNs, otherwise trouble ahead.

26053. 0x00: finish (not part of number) * 0x01: continue

26054. **comment:** this is relatively expensive**label:** code-design

26055. num_values and temp_start reset at top of outer loop

26056. **comment:** * Panic error * * Panic errors are not relative to either a heap or a thread, and cause * DUK_PANIC() macro to be invoked. Unless a user provides DUK_USE_PANIC_HANDLER, * DUK_PANIC() calls a helper which prints out the error and causes a process * exit. * * The user can override the macro to provide custom handling. A macro is * used to allow the user to have inline panic handling if desired (without * causing a potentially risky function call). * * Panics are only used in debug code such as assertions, and by the default * fatal error handler.**label:** code-design

26057. specific assert for wrapping

26058. Could also rely on native sprintf(), but it will handle * values like NaN, Infinity, -0, exponent notation etc in * a JSON-incompatible way.

26059. [...] arg1 ... argN this loggerLevel loggerName buffer 'raw' buffer]

26060. **comment:** Function declaration for global/eval code is emitted even * for duplicates, because of E5 Section 10.5, step 5.e of * E5.1 (special behavior for variable bound to global object). * * DECLVAR will not re-declare a variable as such, but will * update the binding value.**label:** code-design

26061. bits 2...5: type

26062. As a first approximation, buffer values are coerced to strings * for addition. This means that adding two buffers currently * results in a string.

26063. must be ecmascript

26064. 'toUTCString'

26065. Note: no need to re-lookup tv, conversion is side effect free

26066. require.id of current module

26067. low 32 bits is complete

26068. time when status/peek was last done (Date-based rate limit)

26069. [obj key value desc value]

26070. 'caller'

26071. rbp_flags

26072. 'data'

26073. 'type'

26074. **comment:** XXX: shared strings**label:** code-design

26075. -> [...] source]

26076. default value

26077. output 3 bytes from 't'

26078. Special flags checks. Since these strings are always * reachable and a string cannot appear twice in the string * table, there's no need to check/set these flags elsewhere. * The 'internal' flag is set by string intern code.

26079. loop iterator init and limit changed from standard algorithm

26080. enum_flags

26081. '++' or '--' in a post-increment/decrement position, * and a LineTerminator occurs between the operator and * the preceding expression. Force the previous expr * to terminate, in effect treating e.g. "a\b\n++" as * "a,b;++" (= SyntaxError).

26082. by default, use caller's environment

26083. * Note: we currently assume that the setjmp() catchpoint is * not re-entrant (longjmp() cannot be called more than once * for a single setjmp()). * * See doc/code-issues.rst for notes on variable assignment * before and after setjmp().

26084. value resides in 'valstack_idx'

26085. Verbose errors with key/value summaries (non-paranoid) or without key/value * summaries (paranoid, for some security sensitive environments), the paranoid * vs. non-paranoid distinction affects only a few specific errors.

26086. 'typeof' must handle unresolvable references without throwing * a ReferenceError (E5 Section 11.4.3). Register mapped values * will never be unresolvable so special handling is only required * when an identifier is a "slow path" one.

26087. free inner references (these exist e.g. when external * strings are enabled)

26088. The 'this' after 'sep' will get ToString() coerced by * duk_join() automatically. We don't want to do that * coercion when providing .fileName or .lineNumber (GH-254).

26089. surrogate pairs get encoded here

26090. 11: getUTCMonth

26091. **comment:** XXX: any chance of unifying this with the 'length' key handling?**label:** code-design

26092. assert just a few critical flags

26093. correction

26094. default to false

26095. * Stack slice primitives

26096. Length in elements: take into account shift, but * intentionally don't check the underlying buffer here.

26097. [key val] -> []

26098. DUK_USE_STRHASH_DENSE

26099. * Get old and new length

26100. clipped codepoint

26101. h_new_proto may be NULL

26102. -> [...] errhandler errval]

26103. this is the case for normalized numbers

26104. flags: "gim"

26105. Starting from this round, use emergency mode * for mark-and-sweep.

26106. The eval code is executed within the lexical environment of a specified * activation. For now, use global object eval() function, with the eval * considered a 'direct call to eval'. * * Callstack level for debug commands only affects scope -- the callstack * as seen by, e.g. Duktape.act() will be the same regardless.

26107. * Found existing own or inherited plain property, but original * base is a primitive value.

26108. optional callstack level

26109. e.g. duk_push_string_file_raw() pushed undefined

26110. Assume that either all memory funcs are NULL or non-NUL, mixed * cases will now be unsafe.
26111. 'error'
26112. don't throw for prototype loop
26113. (< (* r 2) s)
26114. [thisArg arg1 ... argN func] (thisArg+args == nargs total)
26115. Note: not set in template (has no "prototype")
26116. unreferenced w/o asserts
26117. if (is_repl_func)
26118. precision:shortest
26119. don't resize stringtable (but may sweep it); needed during stringtable resize
26120. Write curr_pc back for the debugger.
26121. Note: duk_dec_req_stridx() backtracks one char
26122. 'length'
26123. -> [... val retval]
26124. Object property access
26125. * The specification uses RegExp [[Match]] to attempt match at specific * offsets. We don't have such a primitive, so we use an actual RegExp * and tweak lastIndex. Since the RegExp may be non-global, we use a * special variant which forces global-like behavior for matching.
26126. 3
26127. Apply timezone offset to get the main parts in UTC
26128. * Zero sign, see misc/tcc_zerosign2.c.
26129. * Boolean built-ins
26130. LeftHandSideExpression does not allow empty expression
26131. **comment:** XXX: add proper spare handling to dynamic buffer, to minimize * reallocs; currently there is no spare at all.
 label: code-design
26132. Note: escaped characters differentiate directives
26133. defaults, E5 Section 8.6.1, Table 7
26134. * Intern the temporary byte buffer into a valstack slot * (in practice, slot1 or slot2).
26135. special RegExp literal handling after IdentifierName
26136. XXX: byte offset?
26137. 'switch'
26138. * The 'duktape.h' header provides the public API, but also handles all * compiler and platform specific feature detection, Duktape feature * resolution, inclusion of system headers, etc. These have been merged * because the public API is also dependent on e.g. detecting appropriate * C types which is quite platform/compiler specific especially for a non-C99 * build. The public API is also dependent on the resolved feature set. * * Some actions taken by the merged header (such as including system headers) * are not appropriate for building a user application. The define * DUK_COMPILING_DUKTAPE allows the merged header to skip/include some * sections depending on what is being built.
26139. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * Lightfunc detection happens here too. Note that lightweight functions * can be wrapped by (non-lightweight) bound functions so we must resolve * the bound function chain first.
26140. avoid side effects
26141. result in csreg
26142. * Other heap related defines
26143. * Start doing property attributes updates. Steps 12-13. * * Start by computing new attribute flags without writing yet. * Property type conversion is done above if necessary.
26144. strict: non-deletable, non-writable
26145. * Assuming a register binds to a variable declared within this * function (a declarative binding), the 'this' for the call * setup is always 'undefined'. E5 Section 10.2.1.1.6.
26146. This may happen when forward and backward scanning disagree * (possible for non-extended-UTF-8 strings).
26147. use fallback as 'this' value
26148. Caller has already eaten the first character ('') which we don't need.
26149. r <- (* r B) * s <- s * m+ <- (* m+ B) * m- <- (* m- B) * k <- (- k 1)
26150. A -> object reg * B -> key reg/const * C -> value reg/const * * Note: intentional difference to register arrangement * of e.g. GETPROP; 'A' must contain a register-only value.
26151. C recursion check.
26152. b
26153. Misc
26154. catches EOF (0x00)
26155. **comment:** XXX: duk_get_length?
 label: code-design
26156. XXX: if assertions enabled, walk through all valid PCs * and check line mapping.
26157. 'defineProperty'
26158. **comment:** XXX: awkward and bloated asm -- use faster internal accesses
 label: code-design
26159. approximation, close enough
26160. marker for detecting internal "double faults", see duk_error_throw.c
26161. three flags
26162. No difference between raw/ensure because the buffer shrinks.
26163. * Determining which datetime components to overwrite based on * stack arguments is a bit complicated, but important to factor * out from setters themselves for compactness. * * If DUK_DATE_FLAG_TIMESETTER, maxnargs indicates setter type: * * 1 -> millisecond * 2 -> second, [millisecond] * 3 -> minute, [second], [millisecond] * 4 -> hour, [minute], [second], [millisecond] * * Else: * * 1 -> date * 2 -> month, [date] * 3 -> year, [month], [date] * * By comparing nargs and maxnargs (and flags) we know which * components to override. We rely on part index ordering.
26164. False if the object is NULL or the prototype 'p' is NULL. * In particular, false if both are NULL (don't compare equal).
26165. Start filling in the activation
26166. SCANBUILD: complains about use of uninitialized values. * The complaint is correct, but operating in undefined * values here is intentional in some cases and the caller * ignores the results.
26167. prevent duk_expr_led() by using a binding power less than anything valid
26168. borrowed reference; although 'tv1' comes from a register, * its value was loaded using LDCONST so the constant will * also exist and be reachable.
26169. [... proplist enum_obj key val]
26170. * Unix-like Date providers * * Generally useful Unix / POSIX / ANSI Date providers.
26171. Push the current 'this' binding; throw TypeError if binding is not object * coercible (CheckObjectCoercible).
26172. All strings beginning with 0xff are treated as "internal", * even strings interned by the user. This allows user code to * create internal properties too, and makes behavior consistent * in case user code happens to use a string also used by Duktape * (such as string has already been interned and has the 'internal' * flag set).
26173. 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [31 bits]
26174. shrink failure is not fatal
26175. !(l < p)
26176. valstack index of loop detection object
26177. limit is quite low: one array entry is 8 bytes, one normal entry is 4+1+8+4 = 17 bytes (with hash entry)

26178. yes -> move back to heap allocated
26179. strict flag for putvar comes from our caller (currently: fixed)
26180. catchstack limits
26181. * Manually optimized number-to-double conversion
26182. [ToObject(this) ToUint32(length) lowerValue upperValue]
26183. if a tail call: * - an Ecmascript activation must be on top of the callstack * - there cannot be any active catchstack entries
26184. [... enum_target res trap_result val true]
26185. Helper for component getter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), push a specified component as a return value to the * value stack and return 1 (caller can then tail call us).
26186. important for callers
26187. fail
26188. [sep ToObject(this) len]
26189. DUK_TOK_PERIOD
26190. Enumeration keys are checked against the enumeration target (to see * that they still exist). In the proxy enumeration case _Target will * be the proxy, and checking key existence against the proxy is not * required (or sensible, as the keys may be fully virtual).
26191. Easiest way to implement the search required by the specification * is to do a RegExp test() with lastIndex forced to zero. To avoid * side effects on the argument, "clone" the RegExp if a RegExp was * given as input. * * The global flag of the RegExp should be ignored; setting lastIndex * to zero (which happens when "cloning" the RegExp) should have an * equivalent effect.
26192. idx_reviver
26193. **comment:** Require a lot of stack to force a value stack grow/shrink. * Recursive mark-and-sweep is prevented by allocation macros * so this won't trigger another mark-and-sweep.
label: code-design
26194. signed shift
26195. Impose a string maximum length, need to handle overflow * correctly.
26196. **comment:** XXX: casts could be improved, especially for GET/SET DATA
label: code-design
26197. bytes left
26198. **comment:** XXX: these last tricks are unnecessary if the function is made * a genuine native function.
label: code-design
26199. No locale specific formatter; this is OK, we fall back * to ISO 8601.
26200. failed
26201. -> [... key]
26202. **comment:** Tested: not faster on x64
label: code-design
26203. clamp to 10 chars
26204. * "LineTerminator" production check.
26205. A lightfunc might also inherit a . toJSON() so just bail out.
26206. **comment:** Note: this is very tricky; we must never 'overshoot' the * correction downwards.
label: code-design
26207. **comment:** XXX: inefficient; block insert primitive
label: code-design
26208. * Exposed helper for setting up heap longjmp state.
26209. in non-packed representation we don't care about which NaN is used
26210. **comment:** Period followed by a digit can only start DecimalLiteral * (handled in slow path). We could jump straight into the * DecimalLiteral handling but should avoid goto to inside * a block.
label: code-design
26211. zero
26212. Handle change in value stack top. Respect value stack * initialization policy: 'undefined' above top. Note that * DECREF may cause a side effect that reallocates valstack, * so must relookup after DECREF.
26213. **comment:** XXX: tostring?
label: code-design
26214. 'func' wants stack "as is"
26215. * Debug message processing
26216. Bottom of valstack for this activation, used to reset * valstack_bottom on return; index is absolute. Note: * idx_top not needed because top is set to 'nregs' always * when returning to an Ecmascript activation.
26217. * Sweep heap
26218. Object.keys
26219. map is reachable through obj
26220. **comment:** XXX: this needs to be reworked so that we never shrink the value * stack on function entry so that we never need to grow it here. * Needing to grow here is a sandboxing issue because we need to * allocate which may cause an error in the error handling path * and thus propagate an error out of a protected call.
label: code-design
26221. **comment:** * Note: since this is an exposed API call, there should be * no way a mark-and-sweep could have a side effect on the * memory allocation behind 'ptr'; the pointer should never * be something that Duktape wants to change. * * Thus, no need to use DUK_REALLOC_INDIRECT (and we don't * have the storage location here anyway).
label: code-design
26222. required for rehash to succeed, equality not that useful
26223. DUK_TOK_CLASS
26224. **comment:** unused for label
label: code-design
26225. 'this' binding shouldn't matter here
26226. **comment:** remove old value
label: code-design
26227. **comment:** get rid of the strings early to minimize memory use before intern
label: code-design
26228. always in entry part, no need to look up parents etc
26229. * Raw intern and lookup
26230. Copy the property table verbatim; this handles attributes etc. * For ROM objects it's not necessary (or possible) to update * refcounts so leave them as is.
26231. CONDITIONAL EXPRESSION
26232. swap elements; deal with non-existent elements correctly
26233. [... closure/error]
26234. * Create a new property in the original object. * * Exotic properties need to be reconsidered here from a write * perspective (not just property attributes perspective). * However, the property does not exist in the object already, * so this limits the kind of exotic properties that apply.
26235. Initial value for highest_idx is -1 coerced to unsigned. This * is a bit odd, but (highest_idx + 1) will then wrap to 0 below * for out_min_size as intended.
26236. ignore
26237. **comment:** XXX: combine all the integer conversions: they share everything * but the helper function for coercion.
label: code-design
26238. vararg function, thisArg needs special handling
26239. Assert for value stack initialization policy.

26240. **comment:** * Shallow fast path checks for accessing array elements with numeric * indices. The goal is to try to avoid coercing an array index to an * (interned) string for the most common lookups, in particular, for * standard Array objects. ** Interning is avoided but only for a very narrow set of cases: * - Object has array part, index is within array allocation, and * value is not unused (= key exists) * - Object has no interfering exotic behavior (e.g. arguments or * string object exotic behaviors interfere, array exotic * behavior does not). ** Current shortcoming: if key does not exist (even if it is within * the array allocation range) a slow path lookup with interning is * always required. This can probably be fixed so that there is a * quick fast path for non-existent elements as well, at least for * standard Array objects.

label: code-design

26241. curr_token slot1 (matches 'lex' slot1_idx)

26242. eat 'break' or 'continue'

26243. zero-based -> one-based

26244. roughly 512 bytes

26245. * indexOf() and lastIndexOf()

26246. DUK_USE_HEAPPTR16

26247. regexp literal must not follow this token

26248. qmin and qmax will be 0 or 1

26249. same as during entry

26250. Have a catch variable.

26251. Note: in strict mode the compiler should reject explicit * declaration of 'eval' or 'arguments'. However, internal * bytecode may declare 'arguments' in the function prologue. * We don't bother checking (or asserting) for these now.

26252. Object property allocation layout

26253. Node.js accepts only actual Arrays.

26254. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined]

26255. **comment:** XXX: Shrink the stacks to minimize memory usage? May not * be worth the effort because terminated threads are usually * garbage collected quite soon.

label: code-design

26256. * Actual Object.defineProperty() default algorithm.

26257. t1 = milliseconds within day (fits 32 bit) * t2 = day number from epoch (fits 32 bit, may be negative)

26258. 'class'

26259. compare: similar to string comparison but for buffer data.

26260. Note: if 'x' is zero, x->n becomes 0 here

26261. Debugger protocol version is defined in the public API header.

26262. not undefined

26263. Endianness indicator

26264. number of activation records in callstack preventing a yield

26265. DUK_TOK_IMPLEMENT

26266. 4'294967295

26267. XXX: switch cast?

26268. 'trace'

26269. array 'length' is always a number, as we coerce it

26270. NULL is allowed, no output

26271. * Convert duk_compiler_func to a function template

26272. Update duk_tval in-place if pointer provided and the * property is writable. If the property is not writable * (immutable binding), use duk_hobject_putprop() which * will respect mutability.

26273. Better equivalent algorithm. If the compiler is compliant, C and * Ecmascript semantics are identical for this particular comparison. * In particular, NaNs must never compare equal and zeroes must compare * equal regardless of sign. Could also use a macro, but this inlines * already nicely (no difference on gcc, for instance).

26274. Is whole and within 32 bit range. If the value happens to be 0xFFFFFFFF, * it's not a valid array index but will then match DUK_NO_ARRAY_INDEX.

26275. * PLAIN: x1 * ARITH: x1 <op> x2 * PROP: x1.x2 * VAR: x1 (name)

26276. eat closing slash

26277. * Regexp range tables

26278. Value stack intentionally mixed size here.

26279. Computation must not wrap; this limit works for 32-bit size_t: * >>> srclen = 3221225469 * >>> "%x" % ((srclen + 2) / 3 * 4) * 'fffffc'

26280. **comment:** * Special post-tweaks, for cases not covered by the init data format. ** - Set Date.prototype.toGMTString to Date.prototype.toUTCString. * toGMTString is required to have the same Function object as * toUTCString in E5 Section B.2.6. Note that while Smjs respects * this, V8 does not (the Function objects are distinct). ** - Make DoubleError non-extensible. ** - Add info about most important effective compile options to Duktape. ** - Possibly remove some properties (values or methods) which are not * desirable with current feature options but are not currently * conditional in init data.

label: code-design

26281. hobject property layout

26282. **comment:** UNUSED, intentionally empty

label: code-design

26283. double quote or backslash

26284. track utf-8 non-continuation bytes

26285. [hobject props enum(props)]

26286. 22 to 31

26287. **comment:** XXX: very similar to DUK_IVAL_ARITH - merge?

label: code-design

26288. **comment:** XXX: optional check, match_caps is zero if no regexp, * so dollar will be interpreted literally anyway.

label: code-design

26289. * Internal helper to define a property with specific flags, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes unless caller requests * that value only be updated if it doesn't already exists. ** Does not support: * - virtual properties (error if write attempted) * - getter/setter properties (error if write attempted) * - non-default (!= WEC) attributes for array entries (error if attempted) * - array abandoning; if array part exists, it is always extended * - array 'length' updating ** Stack: [... in_val] -> [] ** Used for e.g. built-in initialization and environment record * operations.

26290. -> [... enum key enum_target val]

26291. constants arbitrary, chosen for small loads

26292. reset 'allow_in' for parenthesized expression

26293. **comment:** Internal class is Object: Object.prototype.toString.call(new Buffer(0)) * prints "[object Object]".

label: code-design

26294. * The entire allocated buffer area, regardless of actual used * size, is kept zeroed in resizes for simplicity. If the buffer * is grown, zero the new part.

26295. XXX: integrate support for this into led() instead? * Similar issue as post-increment/post-decrement.

26296. E5 Sections 11.8.2, 11.8.5; x > y --> y < x

26297. if present, forces 16-byte duk_tval

26298. * 'arguments' binding is special; if a shadowing argument or * function declaration exists, an arguments object will * definitely not be needed, regardless of whether the identifier * 'arguments' is referenced inside the function body.

26299. exports (this binding)

26300. disabled

26301. **comment:** XXX: other variants like uint, u32 etc

label: code-design

26302. rely on interning, must be this string

26303. Does not allow e.g. `2**31-1`, but one more would allow overflows of `u32`.
26304. Note: cannot be a bound function either right now, * this would be easy to relax though.
26305. Match labels starting from latest; once `label_id` no longer matches, we can * safely exit without checking the rest of the labels (only the topmost labels * are ever updated).
26306. The 'else' ambiguity is resolved by 'else' binding to the innermost * construct, so greedy matching is correct here.
26307. Short circuit if is safe: if `act->curr_pc != NULL`, 'fun' is * guaranteed to be a non-NULL Ecmascript function.
26308. read-only values 'lifted' for ease of use
26309. return value
26310. 'source'
26311. 'clog'
26312. digits
26313. * Statement value handling. * * Global code and eval code has an implicit return value * which comes from the last statement with a value * (technically a non- "empty" continuation, which is * different from an empty statement). * * Since we don't know whether a later statement will * override the value of the current statement, we need * to coerce the statement value to a register allocated * for implicit return values. In other cases we need * to coerce the statement value to a plain value to get * any side effects out (consider e.g. `"foo.bar;"`).
26314. `is_frozen`
26315. **comment:** Downgrade checks are not made everywhere, so 'length' is not always * a fastint (it is a number though). This can be removed once length * is always guaranteed to be a fastint.
label: code-design
26316. if `duk_uint32_t` is exactly 32 bits, this is a NOP
26317. * Not found as a concrete property, check whether a String object * virtual property matches.
26318. * Function formal arguments, always bound to registers * (there's no support for shuffling them now).
26319. off >= step, and step >= 1
26320. zero-width joiner
26321. Rethrow error to calling state.
26322. ""
26323. next temporary register to allocate
26324. * For bound objects, `[[HasInstance]]` just calls the target function * `[[HasInstance]]`. If that is again a bound object, repeat until * we find a non-bound Function object.
26325. Pointer and buffer primitive values are treated like other * primitives values which have a fully fledged object counterpart: * promote to an object value. Lightfuncs are coerced with * `ToObject()` even they could also be returned as is.
26326. **comment:** XXX: option to fix opcode length so it lines up nicely
label: code-design
26327. `xxx -> DUK_HBUFFEROBJECT_ELEM_UINT16`
26328. relookup after side effects (no side effects currently however)
26329. Note: this order matters (final value before deleting map entry must be done)
26330. 'arguments'
26331. * Fast path for bufferobject `getprop/putprop`
26332. XXX: shrink array allocation or entries compaction here?
26333. inline arithmetic check for constant values
26334. pre-incremented, points to first jump slot
26335. * Helpers for `duk_compiler_func`.
26336. Note: need to re-lookup because `ToNumber()` may have side effects
26337. `expr_flags`
26338. **comment:** XXX: exposed `duk_debug_read_hbuffer`
label: code-design
26339. accept string
26340. unreferenced without assertions
26341. **comment:** XXX: macro checks for array index flag, which is unnecessary here
label: code-design
26342. **comment:** XXX: this algorithm could be optimized quite a lot by using e.g. * a logarithm based estimator for 'k' and performing B^n multiplication * using a lookup table or using some bit-representation based exp * algorithm. Currently we just loop, with significant performance * impact for very large and very small numbers.
label: code-design
26343. `'toString'`
26344. first character has been matched
26345. `0xffff0` is `-Infinity`
26346. **comment:** XXX: don't want to shrink allocation here
label: code-design
26347. `-> [... error]`
26348. Lightfunc coerces to a Function instance with concrete * properties. Since 'length' is virtual for Duktape/C * functions, don't need to define that. * * The result is made extensible to mimic what happens to * strings: * `> Object.isExtensible(Object('foo'))` * true
26349. `curr_token` follows 'function'
26350. * Update preventcount
26351. **comment:** These are not needed to implement quantifier capture handling, * but might be needed at some point.
label: requirement
26352. `'toJSON'`
26353. function executes as a constructor (called via "new")
26354. `heapdr`: * - is not reachable * - is an object * - is not a finalized object * - has a finalizer
26355. `DUK_ERROR_H_INCLUDED`
26356. Number/string-or-buffer -> coerce string to number (e.g. `"1.5" == 1.5"` -> true).
26357. XXX: There are several limitations in the current implementation for * strings with $\geq 0x80000000UL$ characters. In some cases one would need * to be able to represent the range `[-0xffffffff, 0xffffffff]` and so on. * Generally character and byte length are assumed to fit into signed 32 * bits ($< 0x80000000UL$). Places with issues are not marked explicitly * below in all cases, look for signed type usage (`duk_int_t` etc) for * offsets/lengths.
26358. * Reset function state and perform register allocation, which creates * 'varmap' for second pass. Function prologue for variable declarations, * binding value initializations etc is emitted as a by-product. * * Strict mode restrictions for duplicate and invalid argument * names are checked here now that we know whether the function * is actually strict. See: `test-dev-strict-mode-boundary.js`. * * Inner functions are compiled during pass 1 and are not reset.
26359. if 1, doing a fixed format output (not free format)
26360. We want to do a straight memory copy if possible: this is * an important operation because `.set()` is the `TypedArray` * way to copy chunks of memory. However, because `set()` * conceptually works in terms of elements, not all views are * compatible with direct byte copying. * * If we do manage a direct copy, the "overlap issue" handled * below can just be solved using `memmove()` because the source * and destination element sizes are necessarily equal.
26361. borrowed; `NULL` if no step state (NULLed in unwind)
26362. new buffer with string contents
26363. Allow very small lenience because some compilers won't parse * exact IEEE double constants (happened in matrix testing with * Linux gcc-4.8 -m32 at least).
26364. We could fit built-in index into magic but that'd make the magic * number dependent on built-in numbering (`genbuiltins.py` doesn't * handle that yet). So map both class and prototype from the * element type.
26365. * Allocate a new thread. * * Leaves the built-ins array uninitialized. The caller must either * initialize a new global context or share existing built-ins from * another thread.

26366. no regexp instance should exist without a non-configurable bytecode property
26367. original input stack before nargs/nregs handling must be * intact for 'arguments' object
26368. replaces top of stack with new value if necessary
26369. 33: setUTCDate
26370. Notes: * - only numbered indices are relevant, so arr_idx fast reject is good * (this is valid unless there are more than 4**32-1 arguments). * - since variable lookup has no side effects, this can be skipped if * DUK_GETDESC_FLAG_PUSH_VALUE is not set.
26371. Automatic semicolon insertion is allowed if a token is preceded * by line terminator(s), or terminates a statement list (right curly * or EOF).
26372. eat 'for'
26373. init 'curr_token'
26374. Index clamping is a bit tricky, we must ensure that we'll only iterate * through elements that exist and that the specific requirements from E5.1 * Sections 15.4.4.14 and 15.4.4.15 are fulfilled; especially: ** - indexOf: clamp to [-len,len], negative handling -> [0,len], * if clamped result is len, for-loop bails out immediately ** - lastIndexOf: clamp to [-len-1, len-1], negative handling -> [-1, len-1], * if clamped result is -1, for-loop bails out immediately * * If fromIndex is not given, ToInteger(undefined) = 0, which is correct * for indexOf() but incorrect for lastIndexOf(). Hence special handling, * and why lastIndexOf() needs to be a vararg function.
26375. implicit attributes
26376. allow a constant to be returned
26377. prop index in 'entry part', < 0 if not there
26378. add_auto_proto
26379. [... ToObject(this) ToUint32(length) val]
26380. Load bytecode instructions.
26381. unsupported: would consume multiple args
26382. DUK_USE_DEBUGGER_THROW_NOTIFY
26383. * Same type? * * Note: since number values have no explicit tag in the 8-byte * representation, need the awkward if + switch.
26384. There's intentionally no check for * current underlying buffer length.
26385. follow parent chain
26386. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT64
26387. **comment:** XXX: this should be an assert
 label: test
26388. without explicit non-BMP support, assume non-BMP characters * are always accepted as letters.
26389. No debugger support, just pop values.
26390. The lookup byte is intentionally sign extended to (at least) * 32 bits and then ORed. This ensures that at least 1 byte * is negative, the highest bit of 't' will be set at the end * and we don't need to check every byte.
26391. [hobject props enum(props) key desc value? getter? setter?]
26392. start match from beginning
26393. DUK_HOBJECT_FLAG_ARRAY_PART: don't care
26394. Validity checks for various fraction formats ("0.1", ".1", "1.", ".").
26395. !DUK_USE_SECTION_B
26396. !DUK_USE_NONSTDFUNC_CALLER_PROPERTY
26397. **comment:** XXX: better to get base and walk forwards?
 label: code-design
26398. 'ownKeys'
26399. **comment:** Call this.raw(msg); look up through the instance allows user to override * the raw() function in the instance or in the prototype for maximum * flexibility.
 label: code-design
26400. restore stack top
26401. DUK_USE_BASE64_FASTPATH
26402. Valstack should suffice here, required on function valstack init
26403. duk_hnativefunction specific fields.
26404. r <- (* f b2) [if b==2 -> (* f 4)] * s <- (* (expt b (- 1 e)) 2) == b^(1-e) * 2 [if b==2 -> b^(2-e)] * m+ <- b == 2 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round
26405. Insert bytes in the middle of the buffer from an external buffer.
26406. identifier names (E5 Section 7.6)
26407. * Set up the resolution input which is the requested ID directly * (if absolute or no current module path) or with current module * ID prepended (if relative and current module path exists). * * Suppose current module is 'foo/bar' and relative path is '/quux'. * The 'bar' component must be replaced so the initial input here is * 'foo/bar/..//quux'.
26408. A non-Array object should not have an array part in practice. * But since it is supported internally (and perhaps used at some * point), check and abandon if that's the case.
26409. keep heap->dbg_detached_cb
26410. For tv1 == tv2, both pointing to stack top, the end result * is same as duk_pop(ctx).
26411. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.
26412. During parsing, month and day are one-based; set defaults here.
26413. **comment:** * Logger arguments are: * * magic: log level (0-5) * this: logger * stack: plain log args * * We want to minimize memory churn so a two-pass approach * is used: first pass formats arguments and computes final * string length, second pass copies strings either into a * pre-allocated and reused buffer (short messages) or into a * newly allocated fixed buffer. If the backend function plays * nice, it won't coerce the buffer to a string (and thus * intern it).
 label: code-design
26414. Non-zero refcounts should not happen because we refcount * finalize all unreachable objects which should cancel out * refcounts (even for cycles).
26415. token allows automatic semicolon insertion (eof or preceded by newline)
26416. Use special opcodes to load short strings
26417. * Add .fileName and .lineNumber to an error on the stack top.
26418. * CommonJS require() and modules support
26419. * Lexing helpers
26420. * Create built-in objects by parsing an init bitstream generated * by genbuiltins.py.
26421. explicit PropertyList
26422. Special handling for call setup instructions. The target * is expressed indirectly, but there is no output shuffling.
26423. * Create mantissa
26424. NOTE: act may be NULL if an error is thrown outside of any activation, * which may happen in the case of, e.g. syntax errors.
26425. ToNumber() for a fastint is a no-op.
26426. [key] -> [undefined] (default value)
26427. traceback depth fits into 16 bits
26428. [thisArg arg1 ... argN func boundFunc argArray]
26429. * Parser duk__advance() token eating functions
26430. **comment:** cleanup varmap from any null entries, compact it, etc; returns number * of final entries after cleanup.
 label: code-design
26431. **comment:** * Needs a save of multiple saved[] entries depending on what range * may be overwritten. Because the regexp parser does no such analysis, * we currently save the entire saved array here. Lookaheads are thus * a bit expensive. Note that the saved array is not needed for just * the lookahead sub-match, but for the matching of the entire sequel. * * The temporary save buffer is pushed on to the valstack to handle * errors correctly. Each lookahead causes a C recursion and

pushes * more stuff on the value stack. If the C recursion limit is less * than the value stack spare, there is no need to check the stack. * We do so regardless, just in case.

label: code-design

26432. **comment:** XXX: split into separate functions for each field type?

label: code-design

26433. Same as above but for unsigned int range.

26434. comma after a value, expected

26435. duplicate key

26436. Given a day number, determine year and day-within-year.

26437. **comment:** Vararg function: must be careful to check/require arguments. * The JSON helpers accept invalid indices and treat them like * non-existent optional parameters.

label: code-design

26438. Success path handles

26439. **comment:** XXX: getter with class check, useful in built-ins

label: code-design

26440. already updated

26441. checking for DUK__DELETED_MARKER is not necessary here, but helper does it now

26442. match: keep saves

26443. DUK_USE_PARANOIAD_MATH

26444. 'boolean'

26445. * Debugging disabled

26446. fixed offset in valstack

26447. DUK_TOK_ENUM

26448. when key is NULL, value is garbage so no need to set

26449. **comment:** * Error during call. The error value is at heap->lj.value1. * * The very first thing we do is restore the previous setjmp catcher. * This means that any error in error handling will propagate outwards * instead of causing a setjmp() re-entry above.

label: code-design

26450. Standard behavior for map(): trailing non-existent * elements don't invoke the user callback and are not * counted towards result 'length'.

26451. * Append a Unicode codepoint to the temporary byte buffer. Performs * CESU-8 surrogate pair encoding for codepoints above the BMP. * Existing surrogate pairs are allowed and also encoded into CESU-8.

26452. * Proxy helpers

26453. * Found; post-processing (Function and arguments objects)

26454. maps to finalizer 2nd argument

26455. DUK_USE_AVOID_PLATFORM_FUNCPTRS

26456. 'roundpos' is relative to nc_ctx->k and increases to the right * (opposite of how 'k' changes).

26457. key encountered as a plain property

26458. e.g. DUK_OP_POSTINCV

26459. Allow empty string to be interpreted as 0

26460. Math.pow(-0,y) where y<0 should be: * - -Infinity if y<0 and an odd integer * - Infinity otherwise * NetBSD pow() returns -Infinity for all finite y<0. The * if- clause also catches y == -Infinity (which works even * without the fix).

26461. **comment:** XXX: add temporary duk_p pointer here too; sharing

label: code-design

26462. not minimum exponent

26463. * Avoid nested calls. Concretely this happens during debugging, e.g. * when we eval() an expression. * * Also don't interrupt if we're currently doing debug processing * (which can be initiated outside the bytecode executor) as this * may cause the debugger to be called recursively. Check required * for correct operation of throw intercept and other "exotic" halting * scenarios.

26464. Because new_size > duk_count_used_probe(heap), guaranteed to work

26465. bump refcount to prevent refzero during finalizer processing

26466. * Pointers to function data area for faster access. Function * data is a buffer shared between all closures of the same * "template" function. The data buffer is always fixed (non- * dynamic, hence stable), with a layout as follows: * * constants (duk_tval) * inner functions (duk_hobject *) * bytecode (duk_instr_t) * * Note: bytecode end address can be computed from 'data' buffer * size. It is not strictly necessary functionally, assuming * bytecode never jumps outside its allocated area. However, * it's a safety/robustness feature for avoiding the chance of * executing random data as bytecode due to a compiler error. * * Note: values in the data buffer must be incref'd (they will * be decref'd on release) for every compiledfunction referring * to the 'data' element.

26467. **comment:** * Advance lookup window by N characters, filling in new characters as * necessary. After returning caller is guaranteed a character window of * at least DUK_LEXER_WINDOW_SIZE characters. * * The main function duk_advance_bytes() is called at least once per every * token so it has a major lexer/compiler performance impact. There are two * variants for the main duk_advance_bytes() algorithm: a sliding window * approach which is slightly faster at the cost of larger code footprint, * and a simple copying one. * * Decoding directly from the source string would be another lexing option. * But the lookup window based approach has the advantage of hiding the * source string and its encoding effectively which gives more flexibility * going forward to e.g. support chunked streaming of source from flash. * * Decodes UTF-8/CESU-8 leniently with support for code points from U+0000 to * U+10FFFF, causing an error if the input is unparseable. Leniency means: * * * Unicode code point validation is intentionally not performed, * except to check that the codepoint does not exceed 0x10ffff. * * * In particular, surrogate pairs are allowed and not combined, which * allows source files to represent all SourceCharacters with CESU-8. * Broken surrogate pairs are allowed, as Ecmascript does not mandate * their validation. * * * Allow non-shortest UTF-8 encodings. * * Leniency here causes few security concerns because all character data is * decoded into Unicode codepoints before lexer processing, and is then * re-encoded into CESU-8. The source can be parsed as strict UTF-8 with * a compiler option. However, Ecmascript source characters include -all- * 16-bit unsigned integer codepoints, so leniency seems to be appropriate. * * Note that codepoints above the BMP are not strictly SourceCharacters, * but the lexer still accepts them as such. Before ending up in a string * or an identifier name, codepoints above BMP are converted into surrogate * pairs and then CESU-8 encoded, resulting in 16-bit Unicode data as * expected by Ecmascript. * * An alternative approach to dealing with invalid or partial sequences * would be to skip them and replace them with e.g. the Unicode replacement * character U+FFFD. This has limited utility because a replacement character * will most likely cause a parse error, unless it occurs inside a string. * Further, Ecmascript source is typically pure ASCII. * * See: * * <http://en.wikipedia.org/wiki=UTF-8> * <http://en.wikipedia.org/wiki/CESU-8> * <http://tools.ietf.org/html/rfc3629> * http://en.wikipedia.org/wiki/UTF-8#Invalid_byte_sequences * * Future work: * * * Reject other invalid Unicode sequences (see Wikipedia entry for examples) * in strict UTF-8 mode. * * * Size optimize. An attempt to use a 16-byte lookup table for the first * byte resulted in a code increase though. * * * Is checking against maximum 0x10ffff really useful? 4-byte encoding * imposes a certain limit anyway. * * * Support chunked streaming of source code. Can be implemented either * by streaming chunks of bytes or chunks of codepoints.

label: code-design

26468. duk_push_this() + CheckObjectCoercible() + duk_to_string()

26469. pointers for scanning

26470. array part size (entirely gc reachable)

26471. Steps 12 and 13: reorganize elements to make room for itemCount elements

26472. number of tokens parsed

26473. Tear down state.

26474. Don't check for Infinity unless the context allows it. * 'Infinity' is a valid integer literal in e.g. base-36: * * parseInt('Infinity', 36) * 1461559270678

26475. zero handled by caller

26476. Don't accept relative indices now.

26477. [holder name val] -> [holder]

26478. res_obj

26479. * XUTF-8 and CESU-8 encoding/decoding

26480. 1110 xxxx 10xx xxxx 10xx xxxx [16 bits]

26481. function declaration

26482. **comment:** XXX: disable error handlers for duration of compaction?
label: code-design

26483. * PUTVAR ** See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is PutValue'd] * 8.7.2 PutValue (V,W) [see especially step 3.b, undefined -> automatic global in non-strict mode] * 8.12.4 [[CanPut]] (P) * 8.12.5 [[Put]] (P) ** Note: may invalidate any valstack (or object) duk_tval pointers because * putting a value may reallocate any object or any valstack. Caller beware.

26484. ?'

26485. state == 3

26486. Maximum traversal depth for "bound function" chains.

26487. **comment:** XXX: magic for getter/setter? use duk_def_prop()?
label: code-design

26488. xxx -> DUK_HBUFFEROBJECT_ELEM_INT32

26489. [thread value]

26490. * GETPROP: Ecmascript property read.

26491. **comment:** * The __FILE__ and __LINE__ information is intentionally not used in the * creation of the error object, as it isn't useful in the tracedata. The * tracedata still contains the function which returned the negative return * code, and having the file/line of this function isn't very useful.
label: code-design

26492. object is a thread (duk_hthread)

26493. 'info'

26494. add a new pending split to the beginning of the entire disjunction

26495. duk_push_this() + CheckObjectCoercible() + duk_to_object()

26496. [... arr jsonx(arr) res] -> [... res jsonx(arr)]

26497. this may have side effects, so re-lookup act

26498. Fall through to handle the rest.

26499. [... error lineNumber fileName]

26500. * Number built-ins

26501. **comment:** XXX: this compiles to over 500 bytes now, even without special handling * for an array part. Uses signed ints so does not handle full array range correctly.
label: code-design

26502. -> [... closure template env]

26503. Note: if the reg_temp load generated shuffling * instructions, we may need to rewind more than * one instruction, so use explicit PC computation.

26504. 6: toUTCString

26505. **comment:** note: ctx-wide temporary
label: code-design

26506. value popped by call

26507. Note: this may fail, caller should protect the call if necessary

26508. Leaves new timevalue on stack top and returns 1, which is correct * for part setters.

26509. stack[0] = compareFn * stack[1] = ToObject(this) * stack[2] = ToUint32(length)

26510. 4

26511. Boolean/any -> coerce boolean to number and try again. If boolean is * compared to a pointer, the final comparison after coercion now always * yields false (as pointer vs. number compares to false), but this is * not special cased.

26512. s < 2 * b

26513. native functions: 149

26514. * Range parsing is done with a special lexer function which calls * us for every range parsed. This is different from how rest of * the parsing works, but avoids a heavy, arbitrary size intermediate * value type to hold the ranges. * * Another complication is the handling of character ranges when * case insensitive matching is used (see docs for discussion). * The range handler callback given to the lexer takes care of this * as well. * * Note that duplicate ranges are not eliminated when parsing character * classes, so that canonicalization of * * [0-9a-fA-Fx-{}]* creates the result (note the duplicate ranges): * * [0-9A-FA-FX-Z-{}]* where [x-{}] is split as a result of canonicalization. The duplicate * ranges are not a semantics issue: they work correctly.

26515. Although we can allow non-BMP characters (they'll decode * back into surrogate pairs), we don't allow extended UTF-8 * characters; they would encode to URIs which won't decode * back because of strict UTF-8 checks in URI decoding. * (However, we could just as well allow them here.)

26516. exponent 1070

26517. no need to reset temps, as we're finished emitting code

26518. DUK_ERR_ERROR: no macros needed

26519. **comment:** XXX: assumed to fit for now
label: code-design

26520. [... fallback retval]

26521. **comment:** XXX: Migrate bufwriter and other read/write helpers to its own header?
label: code-design

26522. recursive call for a primitive value (guaranteed not to cause second * recursion).

26523. Getter might have arbitrary side effects, * so bail out.

26524. * The traceback format is pretty arcane in an attempt to keep it compact * and cheap to create. It may change arbitrarily from version to version. * It should be decoded/accessible through version specific accessors only. * * See doc/error-objects.rst.

26525. The internal __Target property is kept pointing to the original * enumeration target (the proxy object), so that the enumerator * 'next' operation can read property values if so requested. The * fact that the __Target is a proxy disables key existence check * during enumeration.

26526. Reject a proxy object as the handler because it would cause * potentially unbounded recursion. (ES6 has no such restriction)

26527. absolute req_digits; e.g. digits = 1 -> last digit is 0, * but add an extra digit for rounding.

26528. -> [... voidp voidp]

26529. convert duk_compiler_func into a function template, leaving the result * on top of stack.

26530. r -< r (updated above: r -< (remainder (* r B) s) * s -< s * m+ -< m+ (updated above: m+ -< (* m+ B) * m- -< m- (updated above: m- -< (* m- B) * B, low_ok, high_ok are fixed

26531. special helper for emitting u16 lists (used for character ranges for built-in char classes)

26532. [... re_obj input bc saved_buf]

26533. reg_varbind* is the operation result and can also * become the expression value for top level assignments * such as: "var x; x += y;".

26534. **comment:** * Three possible outcomes: * * A try or finally catcher is found => resume there. * (or) * * The error propagates to the bytecode executor entry * level (and we're in the entry thread) => rethrow * with a new longjmp(), after restoring the previous * catchpoint. * * The error is not caught in the current thread, so * the thread finishes with an error. This works like * a yielded error, except that the thread is finished * and can no longer be resumed. (There is always a * resumer in this case.) * * Note: until we hit the entry level, there can only be * Ecmascript activations.
label: code-design

26535. * Number-to-string and string-to-number conversions. * * Slow path number-to-string and string-to-number conversion is based on * a Dragon4 variant, with fast paths for small integers. Big integer * arithmetic is needed for guaranteeing that the conversion is correct * and uses a minimum number of digits. The big number arithmetic has a * fixed maximum size and does not require dynamic allocations. * * See: doc/number-conversion.rst.

26536. Parser separator indices.

26537. Regexp

26538. k > 0 -> k was too low, and cannot be too high

26539. 'func' in the algorithm

26540. **comment:** * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written. * * Note: must re-lookup because calls above (e.g. duk_alloc_entry_checked()) * may realloc and compact properties and hence change e_idx.
label: code-design

26541. Outside any activation -> look up from global.

26542. Decrease by 25% every round
26543. For manual testing only.
26544. **comment:** Execute the fast path in a protected call. If any error is thrown, * fall back to the slow path. This includes e.g. recursion limit * because the fast path has a smaller recursion limit (and simpler, * limited loop detection).
label: code-design
26545. * Main heap structure
26546. * Argument exotic [[Delete]] behavior (E5 Section 10.6) is * a post-check, keeping arguments internal 'map' in sync with * any successful deletes (note that property does not need to * exist for delete to 'succeed'). * * Delete key from 'map'. Since 'map' only contains array index * keys, we can use arr_idx for a fast skip.
26547. E5 Section 9.2
26548. Backtrack utf-8 input and return a (possibly canonicalized) input character.
26549. Cause an interrupt after executing one instruction.
26550. Compiler
26551. at least errobj must be on stack
26552. 16 bits
26553. * Default panic handler
26554. thr->heap->dbg_detaching may be != 0 if a debugger write outside * the message loop caused a transport error and detach1() to run.
26555. 'modSearch'
26556. **comment:** The condition could be more narrow and check for the * copy area only, but there's no need for fine grained * behavior when the underlying buffer is misconfigured.
label: code-design
26557. **comment:** XXX: for a memory-code tradeoff, remove 'func' and make it's access either a function * or a macro. This would make the activation 32 bytes long on 32-bit platforms again.
label: code-design
26558. Function name is an Identifier (not IdentifierName), but we get * the raw name (not recognizing keywords) here and perform the name * checks only after pass 1.
26559. does not fit into 16 bits
26560. ensures no overflow
26561. [... input buffer]
26562. execution resumes in bytecode executor
26563. Standard JSON omits functions
26564. 7 bits
26565. * Value stack resizing. * * This resizing happens above the current "top": the value stack can be * grown or shrunk, but the "top" is not affected. The value stack cannot * be resized to a size below the current "top". * * The low level reallocation primitive must carefully recompute all value * stack pointers, and must also work if ALL pointers are NULL. The resize * is quite tricky because the valstack realloc may cause a mark-and-sweep, * which may run finalizers. Running finalizers may resize the valstack * recursively (the same value stack we're working on). So, after realloc * returns, we know that the valstack "top" should still be the same (there * should not be live values above the "top"), but its underlying size and * pointer may have changed.
26566. * Objects have internal references. Must finalize through * the "refzero" work list.
26567. different thread
26568. number of values in current MPUTARR set
26569. steps 13-14
26570. Compute result length and validate argument buffers.
26571. DUK_BUILTINS_H_INCLUDED
26572. copy bytecode instructions one at a time
26573. **comment:** E5.1 Section 15.2.4.6, step 3.a, lookup proto once before compare. * Prototype loops should cause an error to be thrown.
label: code-design
26574. [... RegExp val] -> [... res]
26575. load factor max 75%
26576. [value offset noAssert], when ftype != DUK__FLD_VARINT
26577. * Resizing
26578. token closes expression, e.g. ')', ']'
26579. fileName
26580. * Get prototype object for an integer error code.
26581. '\xffNext'
26582. PC/line semantics here are: * - For callstack top we're conceptually between two * opcodes and current PC indicates next line to * execute, so report that (matches Status). * - For other activations we're conceptually still * executing the instruction at PC-1, so report that * (matches error stacktrace behavior). * - See: <https://github.com/svaaraala/duktape/issues/281>
26583. Don't allow negative zero as it will cause trouble with * LDINT+LDINTX. But positive zero is OK.
26584. borrow literal Infinity from builtin string
26585. **comment:** XXX: This now coerces an identifier into a GETVAR to a temp, which * causes an extra LDREG in call setup. It's sufficient to coerce to a * unary ivalue?
label: code-design
26586. prefer direct execution
26587. **comment:** Longjmp state is cleaned up by error handling
label: code-design
26588. **comment:** spaces[] must be static to allow initializer with old compilers like BCC
label: code-design
26589. * Object.prototype.hasOwnProperty() and Object.prototype.propertyIsEnumerable().
26590. Ordinary gap, undefined encodes to 'null' in * standard JSON (and no JX/JC support here now).
26591. fall-through jump to next code of next case (backpatched)
26592. already one-based
26593. **comment:** XXX: could use at least one fewer loop counters
label: code-design
26594. Create function 'data' buffer but don't attach it yet.
26595. * Function built-ins
26596. * No match, E5 Section 15.10.6.2, step 9.a.i - 9.a.ii apply, regardless * of 'global' flag of the RegExp. In particular, if lastIndex is invalid * initially, it is reset to zero.
26597. **comment:** hacky helper for String.prototype.split()
label: code-design
26598. [(builtin objects) name func]
26599. We request a tail call, but in some corner cases * call handling can decide that a tail call is * actually not possible. * See: test-bug-tailcall-preventyield-assert.c.
26600. toString() conditions
26601. Write bytecode executor's curr_pc back to topmost activation (if any).
26602. [] -> [val]
26603. **comment:** XXX: Extensibility check for target uses IsExtensible(). If we * implemented the isExtensible trap and didn't reject proxies as * proxy targets, it should be respected here.
label: code-design
26604. **comment:** Technically return value is not needed because INVLHS will * unconditionally throw a ReferenceError. Coercion is necessary * for proper semantics (consider ToNumber() called for an object). * Use DUK_EXTRAOP_UNP with a dummy register to get ToNumber().
label: code-design

26605. decl type
26606. -> [sep ToObject(this) len sep str]
26607. Shared lenient buffer length clamping helper. Indices are treated as * element indices (though output values are byte offsets) which only * really matters for TypedArray views as other buffer object have a zero * shift. Negative indices are counted from end of input slice; crossed * indices are clamped to zero length; and final indices are clamped * against input slice. Used for e.g. ArrayBuffer slice().
26608. XXX: perhaps refactor this to allow caller to specify some parameters, or * at least a 'compact' flag which skips any spare or round-up .. useful for * emergency gc.
26609. [... enum_target res handler trap]
26610. XXX: Decrementing and restoring act->curr_pc works now, but if the * debugger message loop gains the ability to adjust the current PC * (e.g. a forced jump) restoring the PC here will break. Another * approach would be to use a state flag for the "decrement 1 from * topmost activation's PC" and take it into account whenever dealing * with PC values.
26611. Create a fresh object environment for the global scope. This is * needed so that the global scope points to the newly created RAM-based * global object.
26612. x <- (1<<y)
26613. * HASPROP: Ecmascript property existence check ("in" operator). * * Interestingly, the 'in' operator does not do any coercion of * the target object.
26614. Any 'res' will do.
26615. If neutered must return 0; length is zeroed during * neutering.
26616. XXX: assert idx_base
26617. **comment:** XXX: refactor into internal helper, duk_clone_hobject()
label: code-design
26618. * Get starting character offset for match, and initialize 'sp' based on it. * * Note: lastIndex is non-configurable so it must be present (we check the * internal class of the object above, so we know it is). User code can set * its value to an arbitrary (garbage) value though; E5 requires that lastIndex * be coerced to a number before using. The code below works even if the * property is missing: the value will then be coerced to zero. * * Note: lastIndex may be outside Uint32 range even after ToInteger() coercion. * For instance, ToInteger(+Infinity) = +Infinity. We track the match offset * as an integer, but pre-check it to be inside the 32-bit range before the loop. * If not, the check in E5 Section 15.10.6.2, step 9.a applies.
26619. avoid side effect issues
26620. * Setup call: target and 'this' binding. Three cases: * * 1. Identifier base (e.g. "foo()") * 2. Property base (e.g. "foo.bar()") * 3. Register base (e.g. "foo()0"; i.e. when a return value is a function)
26621. SHIFT EXPRESSION
26622. overflow, fall through
26623. restored if ecma-to-ecma setup fails
26624. Match labels starting from latest label because there can be duplicate empty * labels in the label set.
26625. If debugger is paused, garbage collection is disabled by default.
26626. DUK_ERR_REFERENCE_ERROR: no macros needed
26627. XXX: shared helper for duk_push_hobject_or_undefined()
26628. * Strings have no internal references but do have "weak" * references in the string cache. Also note that strings * are not on the heap_allocated list like other heap * elements.
26629. unconditionally
26630. **comment:** XXX: At the moment Duktape accesses internal keys like _Finalizer using a * normal property set/get which would allow a proxy handler to interfere with * such behavior and to get access to internal key strings. This is not a problem * as such because internal key strings can be created in other ways too (e.g. * through buffers). The best fix is to change Duktape internal lookups to * skip proxy behavior. Until that, internal property accesses bypass the * proxy and are applied to the target (as if the handler did not exist). * This has some side effects, see test-bi-proxy-internal-keys.js.
label: code-design
26631. Get local time offset (in seconds) for a certain (UTC) instant 'd'.
26632. **comment:** Can be called multiple times with no harm. Mark the transport * bad (dbg_read_cb == NULL) and clear state except for the detached * callback and the udata field. The detached callback is delayed * to the message loop so that it can be called between messages; * this avoids corner cases related to immediate debugger reattach * inside the detached callback.
label: code-design
26633. resume
26634. * Parse a function body or a function-like expression, depending * on flags.
26635. actually used, non-NULL entries
26636. Fast path check.
26637. length check
26638. Do decrefs only with safe pointers to avoid side effects * disturbing e_idx.
26639. {'_ninf':true}'
26640. shared checks for all descriptor types
26641. There's no need to check for buffer validity status for the * target here: the property access code will do that for each * element. Moreover, if we did check the validity here, side * effects from reading the source argument might invalidate * the results anyway.
26642. * Process space (3rd argument to JSON.stringify)
26643. Difference to non-strict/strict comparison is that NaNs compare * equal and signed zero signs matter.
26644. DUK_USE_PANIC_HANDLER
26645. avoid array abandoning which interns strings
26646. traits are separate; in particular, arguments not an array
26647. duk_get_min_grow_a() is always >= 1
26648. 9: getUTCFullYear
26649. [key getter this key] -> [key retval]
26650. Source end clamped silently to available length.
26651. First character has already been eaten and checked by the caller. * We can scan until a NUL in stridx string because no built-in strings * have internal NULs.
26652. * Proxy built-in (ES6)
26653. Native function, no relevant lineNumber.
26654. * Limited functionality bigint implementation. * * Restricted to non-negative numbers with less than 32 * DUK_BI_MAX_PARTS bits, * with the caller responsible for ensuring this is never exceeded. No memory * allocation (except stack) is needed for bigint computation. Operations * have been tailored for number conversion needs. * * Argument order is "assignment order", i.e. target first, then arguments: * x <- y * z --> duk_bi_mul(x, y, z);
26655. * InitJS code - Ecmascript code evaluated from a built-in source * which provides e.g. backward compatibility. User can also provide * JS code to be evaluated at startup.
26656. 'res' must be a plain ivalue, and not register-bound variable.
26657. function helpers
26658. 'Null'
26659. internal properties
26660. custom
26661. * XXX: shrink array allocation or entries compaction here?
26662. ToInteger(lastIndex)
26663. * Encoding/decoding helpers
26664. default: NaN
26665. covers -Infinity
26666. Error instance, use augmented error data directly
26667. t1 = (+ r m+)
26668. [... func arg1 ... argN]
26669. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small; some overlap with string * handling.
label: code-design

26670. Some contexts don't allow fractions at all; this can't be a * post-check because the state ('f' and expt) would be incorrect.
26671. variable value (function, we hope, not checked here)
26672. return 'res_obj'
26673. E5 Section 10.4.2
26674. DUK_FORWDECL_H_INCLUDED
26675. avoid unary minus on unsigned
26676. Just flip the single bit.
26677. unicode code points, window[0] is always next, points to 'buffer'
26678. probe
26679. may be set to 0 by duk_debugger_attach() inside callback
26680. Easy to get wrong, so assert for it.
26681. mask out flags not actually stored
26682. 'with' target must be created first, in case we run out of memory
26683. **comment:** Slow writing of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. There's * no special sign handling when writing varints.
label: code-design
26684. Important main primitive.
26685. no throw
26686. [... lval rval(func)]
26687. **comment:** XXX: 'this' will be ToObject() coerced twice, which is incorrect * but should have no visible side effects.
label: code-design
26688. **comment:** XXX: The duk_to_number() cast followed by integer coercion * is platform specific so NaN, +/- Infinity, and out-of-bounds * values result in platform specific output now. * See: test-bi-nodejs-buffer-proto-varint-special.js
label: code-design
26689. 'lastIndex'
26690. 0xd0-0xdf
26691. * Format tag parsing. Since we don't understand all the * possible format tags allowed, we just scan for a terminating * specifier and keep track of relevant modifiers that we do * understand. See man 3 printf.
26692. pop varname
26693. **comment:** XXX: this should probably just operate on the stack top, because it * needs to push stuff on the stack anyway...
label: code-design
26694. 'advtok' indicates how much to advance and which token id to assign * at the end. This shared functionality minimizes code size. All * code paths are required to set 'advtok' to some value, so no default * init value is used. Code paths calling DUK_ERROR() never return so * they don't need to set advtok.
26695. debugger
26696. * Automatically generated by extract_chars.py, do not edit!
26697. * Copy keys and values in the entry part (compacting them at the same time).
26698. expensive init, just want to fill window
26699. 'toLogString'
26700. split2: prefer jump execution (not direct)
26701. Zero special case: fake requested number of zero digits; ensure * no sign bit is printed. Relative and absolute fixed format * require separate handling.
26702. skip line info
26703. heap destruction ongoing, finalizer rescue no longer possible
26704. E5 Section 9.1
26705. value stack indices for tracking objects
26706. two encoding attempts suffices
26707. DUK_ISPEC_XXX
26708. **comment:** XXX: append primitive
label: code-design
26709. current lexical environment (may be NULL if delayed)
26710. explicit NULL inits
26711. [... closure template env]
26712. use a temp: decref only when valstack reachable values are correct
26713. Helper for string conversion calls: check 'this' binding, get the * internal time value, and format date and/or time in a few formats. * Return value allows tail calls.
26714. 0 or -1
26715. **comment:** * Allocate heap struct * * Use a raw call, all macros expect the heap to be initialized
label: code-design
26716. Object.defineProperty() equivalent C binding.
26717. skip jump slots
26718. We can directly access value stack here.
26719. DUK_TOK_TYPEOF
26720. mark as complex
26721. XXX: handling of timestamps outside Windows supported range. * How does Windows deal with dates before 1600? Does windows * support all Ecmascript years (like -200000 and +200000)? * Should equivalent year mapping be used here too? If so, use * a shared helper (currently integrated into timeval-to-parts).
26722. **comment:** m+ != m- (very rarely)
label: code-design
26723. * Duktape debugger
26724. Ecmascript activation + Duktape.Thread.resume() activation
26725. "\n"
26726. [... str]
26727. -> [... value]
26728. Use temporaries and update lex_ctx only when finished.
26729. mark-and-sweep not running -> must be empty
26730. **comment:** * Normal error * * Normal error is thrown with a longjmp() through the current setjmp() * catchpoint record in the duk_heap. The 'curr_thread' of the duk_heap * identifies the throwing thread. * * Error formatting is usually unnecessary. The error macros provide a * zero argument version (no formatting) and separate macros for small * argument counts. Variadic macros are not used to avoid portability * issues and avoid the need for stash-based workarounds when they're not * available. Vararg calls are avoided for non-formatted error calls * because vararg call sites are larger than normal, and there are a lot * of call sites with no formatting. * * Note that special formatting provided by debug macros is NOT available. * * The _RAW variants allow the caller to specify file and line. This makes * it easier to write checked calls which want to use the call site of the * checked function, not the error macro call inside the checked function.
label: code-design
26731. **comment:** Fast canonicalization lookup at the cost of 128kB footprint.
label: code-design
26732. DUK_TOK_EXPORT
26733. array of active label names
26734. * Helper for C function call negative return values.
26735. entry_top + 0
26736. * Thread switching * * To switch heap->curr_thread, use the macro below so that interrupt counters * get updated correctly. The macro allows a NULL target thread because that * happens e.g. in call handling.
26737. entry allocation updates hash part and increases the key * refcount; may need a props allocation resize but doesn't * 'recheck' the valstack.

26738. Not necessary to unwind catchstack: break/continue * handling will do it. The finally flag of 'cat' is * no longer set. The catch flag may be set, but it's * not checked by break/continue handling.

26739. Shared helper.

26740. x1 must be a string

26741. [... value? getter? setter?]

26742. **comment:** XXX: the insert here is a bit expensive if there are a lot of items. * It could also be special cased in the outermost for loop quite easily * (as the element is dup'd anyway).

label: code-design

26743. Gather in big endian

26744. **comment:** SCANBUILD: warning about 'thr' potentially being NULL here, * warning is incorrect because thr != NULL always here.

label: code-design

26745. format is bit packed

26746. See test-bug-netbsd-math-pow.js: NetBSD 6.0 on x86 (at least) does not * correctly handle some cases where x=+/-0. Specific fixes to these * here.

26747. parse args starting from "next temp", reg_target + 1

26748. Note: multiple threads may be simultaneously in the RUNNING * state, but not in the same "resume chain".

26749. * Named function expression, name needs to be bound * in an intermediate environment record. The "outer" * lexical/variable environment will thus be: * * a { funcname: <func>, __prototype: outer_lex_env } * b { funcname: <func>, __prototype: <globalenv> } (if outer_lex_env missing)

26750. * x is finite and non-zero * * -1.6 -> floor(-1.1) -> -2 * -1.5 -> floor(-1.0) -> -1 (towards +Inf) * -1.4 -> floor(-0.9) -> -1 * -0.5 -> -0.0 (special case) * -0.1 -> -0.0 (special case) * +0.1 -> +0.0 (special case) * +0.5 -> floor(+1.0) -> 1 (towards +Inf) * +1.4 -> floor(+1.9) -> 1 * +1.5 -> floor(+2.0) -> 2 (towards +Inf) * +1.6 -> floor(+2.1) -> 2

26751. * Context init

26752. **comment:** XXX: rework

label: code-design

26753. Call stack. [0,callstack_top[is GC reachable.

26754. DUK_NUMCONV_H_INCLUDED

26755. is a function declaration (as opposed to function expression)

26756. Precomputed pointers when using 16-bit heap pointer packing.

26757. The E5.1 Section 15.4.4.4 algorithm doesn't set the length explicitly * in the end, but because we're operating with an internal value which * is known to be an array, this should be equivalent.

26758. XXX: Allow customizing the pause and notify behavior at runtime * using debugger runtime flags. For now the behavior is fixed using * config options.

26759. = IsAccessorDescriptor(Desc)

26760. 36: setFullYear

26761. **comment:** not covered, return all zeroes

label: test

26762. * Unicode range matcher * * Matches a codepoint against a packed bitstream of character ranges. * Used for slow path Unicode matching.

26763. DUK_HTHREAD_H_INCLUDED

26764. DUK_USE_ES6_PROXY

26765. Located in duk_heap.hdr h_extra16. Subclasses of duk_hobject (like * duk_hcompiledfunction) are not free to use h_extra16 for this reason.

26766. When looking for .fileName/.lineNumber, blame compilation * or C call site unless flagged not to do so.

26767. Pretend like we got EOM

26768. [regexp source bytecode]

26769. catch depth at point of definition

26770. -> [... cons target]

26771. Grow entry part allocation for one additional entry.

26772. **comment:** * Calls. * * Protected variants should avoid ever throwing an error.

label: code-design

26773. **comment:** XXX: better multiline

label: code-design

26774. object literal key tracking flags

26775. curr is accessor -> cannot be in array part

26776. stmt has explicit/implicit semicolon terminator

26777. lexing

26778. [... result/error]

26779. catches EOF (NUL) and initial comma

26780. **comment:** If no explicit message given, put error code into message field * (as a number). This is not fully in keeping with the EcmaScript * error model because messages are supposed to be strings (Error * constructors use ToString() on their argument). However, it's * probably more useful than having a separate 'code' property.

label: code-design

26781. The #ifdef clutter here needs to handle the three cases: * (1) JX+JC, (2) JX only, (3) JC only.

26782. currently running thread

26783. temproots

26784. Tail call check: if last opcode emitted was CALL(I), and * the context allows it, change the CALL(I) to a tail call. * This doesn't guarantee that a tail call will be allowed at * runtime, so the RETURN must still be emitted. (Duktape * 0.10.0 avoided this and simulated a RETURN if a tail call * couldn't be used at runtime; but this didn't work * correctly with a thread yield/resume, see * test-bug-tailcall-thread-yield-resume.js for discussion.) * * In addition to the last opcode being CALL, we also need to * be sure that 'rc_val' is the result register of the CALL(I). * For instance, for the expression 'return 0, (function () { return 1; })', 2' the last opcode emitted is CALL (no * bytecode is emitted for '2') but 'rc_val' indicates * constant '2'. Similarly if '2' is replaced by a register * bound variable, no opcodes are emitted but tail call would * be incorrect. * * This is tricky and easy to get wrong. It would be best to * track enough expression metadata to check that 'rc_val' came * from that last CALL instruction. We don't have that metadata * now, so we check that 'rc_val' is a temporary register result * (not a constant or a register bound variable). There should * be no way currently for 'rc_val' to be a temporary for an * expression following the CALL instruction without emitting * some opcodes following the CALL. This proxy check is used * below. * * See: test-bug-comma-expr-gh131.js. * * The non-standard 'caller' property disables tail calls * because they pose some special cases which haven't been * fixed yet.

26785. * Helper: handle Array object 'length' write which automatically * deletes properties, see E5 Section 15.4.5.1, step 3. This is * quite tricky to get right. * * Used by duk_hobject_putprop().

26786. vararg

26787. Use 'res' as the expression value (it's side effect * free and may be a plain value, a register, or a * constant) and write it to the LHS binding too.

26788. Allow NULL 'msg'

26789. borrowed reference

26790. [... s1 s2] -> [... s1+s2]

26791. DUK_TOK_DIV

26792. **comment:** XXX: lithuanian not implemented

label: requirement

26793. Explicit zero size check to avoid NULL 'tv1'.

26794. * Get holder object

26795. XXX: can shift() / unshift() use the same helper? * shift() is (close to?) <-> splice(0, 1) * unshift is (close to?) <-> splice(0, 0, [items])?

26796. lead zero + 'digits' fractions + 1 for rounding

26797. stack discipline consistency check

26798. * Update the interrupt counter

26799. these non-standard properties are copied for convenience

26800. still reachable
26801. * Forward declarations for all Duktape structures.
26802. [... obj key]
26803. **comment:** * Object.preventExtensions() and Object.isExtensible() (E5 Sections 15.2.3.10, 15.2.3.13) ** Not needed, implemented by macros DUK_HOBJECT_{HAS,CLEAR,SET}_EXTENSIBLE * and the Object built-in bindings.
label: code-design
26804. * Macros for property handling
26805. val2 = min count, val1 = max count
26806. find and remove string from stringtable; caller must free the string itself
26807. Inner functions recursively.
26808. == DUK_MAX_TEMPS is OK
26809. Don't free h->resumer because it exists in the heap. * Callstack entries also contain function pointers which * are not freed for the same reason.
26810. use bigint area as a temp
26811. 'n'
26812. * Allocate initial stacks for a thread. Note that 'thr' must be reachable * as a garbage collection may be triggered by the allocation attempts. * Returns zero (without leaking memory) if init fails.
26813. step 4.b
26814. (Number.MAX_VALUE).toString(2).length == 1024, + spare
26815. * Check whether already declared. * * We need to check whether the binding exists in the environment * without walking its parents. However, we still need to check * register-bound identifiers and the prototype chain of an object * environment target object.
26816. Catch stack. [0,catchstack_top[is GC reachable.
26817. DUK_USE_DATE_TZO_WINDOWS
26818. -> [...] res_obj]
26819. * E5 Section 15.10.2.6. The previous and current character * should -not- be canonicalized as they are now. However, * canonicalization does not affect the result of IsWordChar() * (which depends on Unicode characters never canonicalizing * into ASCII characters) so this does not matter.
26820. flags is unsigned
26821. 0
26822. **comment:** XXX: better place for this
label: code-design
26823. Zero encoded pointer is required to match NULL
26824. Given a (year, month, day-within-month) triple, compute day number. * The input triple is un-normalized and may contain non-finite values.
26825. upper limit, assuming no whitespace etc
26826. Copy slice, respecting underlying buffer limits; remainder * is left as zero.
26827. **comment:** not necessary to init, disabled for faster parsing
label: code-design
26828. We already ate 'x', so backup one byte.
26829. maximum token count before error (sanity backstop)
26830. * For ASCII strings, the answer is simple.
26831. 'super'
26832. **comment:** XXX: This won't be shown in practice now * because this code is not run when builtins * are in ROM.
label: code-design
26833. 'JSON'
26834. combine left->x1 and res->x1 (right->x1, really) -> (left->x1 OP res->x1)
26835. We can't reliably pop anything here because the stack input * shape is incorrect. So we throw an error; if the caller has * no catch point for this, a fatal error will occur. Another * alternative would be to just return an error. But then the * stack would be in an unknown state which might cause some * very hard to diagnose problems later on. Also note that even * if we did not throw an error here, the underlying call handler * might STILL throw an out-of-memory error or some other internal * fatal error.
26836. marker: copy t if not changed
26837. thread has terminated
26838. * Context management
26839. Return codes for protected calls (duk_safe_call(), duk_pcall())
26840. RangeError
26841. set/clear writable
26842. DUK_USE_64BIT_OPS
26843. EcmaScript date range is 100 million days from Epoch: * > 100e6 * 24 * 60 * 60 * 1000 // 100M days in millisecs * 8640000000000000 * (= 8.64e15)
26844. * Coercion operations: in-place coercion, return coerced value where * applicable. If index is invalid, throw error. Some coercions may * throw an expected error (e.g. from a toString() or valueOf() call) * or an internal error (e.g. from out of memory).
26845. packed tvl
26846. (internal) catch compilation errors
26847. * END PUBLIC API
26848. E == 0x7ff, topmost four bits of F != 0 => assume NaN
26849. APIError
26850. * BEGIN PUBLIC API
26851. timeval breakdown: internal time value NaN -> RangeError (toISOString)
26852. internal flag: external buffer
26853. enumerate internal properties (regardless of enumerability)
26854. EcmaScript E5 specification error codes
26855. * Debugger (debug protocol)
26856. lstring
26857. timeval breakdown: internal time value NaN -> zero
26858. include time part in string conversion result
26859. prefer number
26860. Note: parentheses are required so that the comma expression works in assignments.
26861. * Duktape public API for Duktape 1.5.2. * * See the API reference for documentation on call semantics. * The exposed API is inside the DUK_API_PUBLIC_H_INCLUDED * include guard. Other parts of the header are Duktape * internal and related to platform/compiler/feature detection. * * Git commit cad34ae155acb0846545ca6bf2d29f9463b22bbb (v1.5.2). * Git branch HEAD. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.
26862. internal flag value: throw if mask doesn't match
26863. set getter (given on value stack)
26864. no error if file does not exist
26865. * Memory management * * Raw functions have no side effects (cannot trigger GC).
26866. Coercion hints
26867. is_copy
26868. * Variable access
26869. weekday: 0 to 6, 0=sunday, 1=monday, etc
26870. DUK_DBLEUNION_H_INCLUDED
26871. lightweight function pointer
26872. * String manipulation

26873. getter: subtract 1900 from year when getting year part
26874. Value mask types, used by e.g. duk_get_type_mask()
26875. Ecmascript undefined
26876. file
26877. Log levels
26878. Ecmascript boolean: 0 or 1
26879. * Public API specific typedefs * * Many types are wrapped by Duktape for portability to rare platforms * where e.g. 'int' is a 16-bit type. See practical typing discussion * in Duktape web documentation.
26880. flags
26881. Assertion Error
26882. only enumerate array indices
26883. * Stack management
26884. Flags for duk_def_prop() and its variants
26885. NOTE: when writing a Date provider you only need a few specific * flags from here, the rest are internal. Avoid using anything you * don't need.
26886. * Date provider related constants * * NOTE: These are "semi public" - you should only use these if you write * your own platform specific Date provider, see doc/datetime.rst.
26887. internal: request fixed buffer result
26888. Number of value stack entries (in addition to actual call arguments) * guaranteed to be allocated on entry to a Duktape/C function.
26889. * Indexes of various types with respect to big endian (logical) layout
26890. no error (e.g. from duk_get_error_code())
26891. LICENSE.txt
26892. SyntaxError
26893. * Duktape/C function magic value
26894. internal: request dynamic buffer result
26895. * Avoid C++ name mangling
26896. **comment:** XXX: replace with TypeError?
 label: code-design
26897. **comment:** XXX: to be removed?
 label: code-design
26898. Git commit, describe, and branch for Duktape build. Useful for * non-official snapshot builds so that application code can easily log * which Duktape snapshot was used. Not available in the Ecmascript * environment.
26899. DUK_API_PUBLIC_H_INCLUDED
26900. Compilation flags for duk_compile() and duk_eval()
26901. timeval breakdown: convert month and day-of-month parts to one-based (default is zero-based)
26902. Duktape version, (major * 10000) + (minor * 100) + patch. Allows C code * to #ifdef against Duktape API version. The same value is also available * to Ecmascript code in Duktape.version. Unofficial development snapshots * have 99 for patch level (e.g. 0.10.99 would be a development version * after 0.10.0 but before the next official release).
26903. * Buffer
26904. set enumerable (effective if DUK_DEFPROP_HAVE_ENUMERABLE set)
26905. set configurable (effective if DUK_DEFPROP_HAVE_CONFIGURABLE set)
26906. don't walk prototype chain, only check own properties
26907. UnimplementedError
26908. (internal) omit eval result
26909. **comment:** * If no variadic macros, __FILE__ and __LINE__ are passed through globals * which is ugly and not thread safe.
 label: requirement
26910. last value is func pointer, arguments follow in parens
26911. set/clear configurable
26912. Return codes for C functions (shortcut for throwing an error)
26913. * Pop operations
26914. * Require operations: no coercion, throw error if index or type * is incorrect. No defaulting.
26915. * Bytecode load/dump
26916. compile function code (instead of global code)
26917. enumerate a proxy object itself without invoking proxy behavior
26918. **comment:** XXX: native 64-bit byteswaps when available
 label: code-design
26919. **comment:** XXX: These calls are incomplete and not usable now. They are not (yet) * part of the public API.
 label: code-design
26920. Error
26921. DUK_USE_FILE_IO
26922. ReferenceError
26923. timeval breakdown: replace year with equivalent year in the [1971,2037] range for DST calculations
26924. * Ecmascript operators
26925. args
26926. * Some defines forwarded from feature detection
26927. (internal) take strlen() of src_buffer (avoids double evaluation in macro)
26928. set/clear enumerable
26929. internal use
26930. * Error handling
26931. EvalError
26932. * Object prototype
26933. setter: perform 2-digit year fixup (00...99 -> 1900...1999)
26934. nop: no need to normalize
26935. * Compilation and evaluation
26936. or is a normalized NaN
26937. AllocError
26938. month: 0 to 11
26939. internal flag: create backing ArrayBuffer; keep in one byte
26940. additional values begin at bit 12
26941. * Constants
26942. **comment:** XXX: replace with plain Error?
 label: code-design
26943. * Object finalizer
26944. Reverse operation is the same.
26945. duk_context is now defined in duk_config.h because it may also be * referenced there by prototypes.
26946. * C++ name mangling
26947. UncaughtError
26948. Ecmascript year range: * > new Date(100e6 * 24 * 3600e3).toISOString() * '+275760-09-13T00:00:00.000Z' * > new Date(-100e6 * 24 * 3600e3).toISOString() * '-271821-04-20T00:00:00.000Z'

26949. create a new global environment
26950. * Global object
26951. DUKTAPE_H_INCLUDED
26952. E == 0x7ff, F != 0 => NaN
26953. DUK_USE_PACKED_TVAL
26954. * Logging
26955. safe variants of a few coercion operations
26956. Indicates that a native function does not have a fixed number of args, * and the argument stack should not be capped/extended at all.
26957. * Get operations: no coercion, returns default value for invalid * indices and invalid value types. ** duk_get_undefined() and duk_get_null() would be pointless and * are not included.
26958. **comment:** use locale specific formatting if available
label: code-design
26959. External duk_config.h provides platform/compiler/OS dependent * typedefs and macros, and DUK_USE_xxx config options so that * the rest of Duktape doesn't need to do any feature detection.
26960. internal: don't care about fixed/dynamic nature
26961. * Property access ** The basic function assumes key is on stack. The _string variant takes * a C string as a property name, while the _index variant takes an array * index as a property name (e.g. 123 is equivalent to the key "123").
26962. * Debugging
26963. 64-bit byteswap, same operation independent of target endianness.
26964. * Type checks ** duk_is_none(), which would indicate whether index is outside of stack, * is not needed; duk_is_valid_index() gives the same information.
26965. sort array indices, use with DUK_ENUM_ARRAY_INDICES_ONLY
26966. force change if possible, may still fail for e.g. virtual properties
26967. Millisecond count constants.
26968. used by packed duk_tval, assumes sizeof(void *) == 4
26969. set value (given on value stack)
26970. Used to represent invalid index; if caller uses this without checking, * this index will map to a non-existent stack entry. Also used in some * API calls as a marker to denote "no value".
26971. * Module helpers: put multiple function or constant properties
26972. raw void pointer
26973. Flags for duk_push_thread_raw()
26974. * ===== * Duktape authors * ===== * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang\u00f3f \u00e1llangol\u00e1szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6tzte (<https://github.com/jaseg>) * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6man * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstruktrup> * * Michael Drake (<https://github.com/tlsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn\u00e1s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.
26975. InternalError
26976. Value types, used by e.g. duk_get_type()
26977. **comment:** Duktape specific error codes (must be 8 bits at most, see duk_error.h)
label: code-design
26978. no value, e.g. invalid index
26979. * Stack manipulation (other than push/pop)
26980. Byteswap an IEEE double in the duk_double_union from host to network * order. For a big endian target this is a no-op.
26981. prefer string
26982. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2016 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
26983. external use
26984. Flags for duk_push_string_file_raw()
26985. Duktape debug protocol version used by this build.
26986. **comment:** DUK_COMPILE_xxx bits 0-2 are reserved for an internal 'nargs' argument * (the nargs value passed is direct stack arguments + 1 to account for an * internal extra argument).
label: code-design
26987. fixed or dynamic, garbage collected byte buffer
26988. * Thread management
26989. * Double NaN manipulation macros related to NaN normalization needed when * using the packed duk_tval representation. NaN normalization is necessary * to keep double values compatible with the duk_tval format. * * When packed duk_tval is used, the NaN space is used to store pointers * and other tagged values in addition to NaNs. Actual NaNs are normalized * to a specific quiet NaN. The macros below are used by the implementation * to check and normalize NaN values when they might be created. The macros * are essentially NOPs when the non-packed duk_tval representation is used. * * A FULL check is exact and checks all bits. A NOTFULL check is used by * the packed duk_tval and works correctly for all NaNs except those that * begin with 0x7ff0. Since the 'normalized NaN' values used with packed * duk_tval begin with 0x7ff8, the partial check is reliable when packed * duk_tval is used. The 0x7ff8 prefix means the normalized NaN will be a * quiet NaN regardless of its remaining lower bits. * * The ME variant below is specifically for ARM byte order, which has the * feature that while doubles have a mixed byte order (32107654), unsigned * long long values has a little endian byte order (76543210). When writing * a logical double value through a ULL pointer, the 32-bit words need to be * swapped; hence the #ifdefs below for ULL writes with DUK_USE_DOUBLE_ME. * This is not full ARM support but suffices for some environments.
26990. either not a NaN
26991. convert time value to local time
26992. not directly applicable, byte order differs from a double
26993. end "C" wrapper
26994. * Object operations
26995. set writable (effective if DUK_DEFPROP_HAVE_WRITABLE set)

26996. UnsupportedError

26997. **comment:** Internal API call flags, used for various functions in this file. * Certain flags are used by only certain functions, but since the flags * don't overlap, a single flags value can be passed around to multiple * functions. * * The unused top bits of the flags field are also used to pass values * to helpers (duk_get_part_helper() and duk_set_part_helper()). * * (Must be in-sync with genbuiltins.py.)

label: code-design

26998. There are currently no native functions to yield/resume, due to the internal * limitations on coroutine handling. These will be added later.

26999. Concrete macros for NaN handling used by the implementation internals. * Chosen so that they match the duk_tval representation: with a packed * duk_tval, ensure NaNs are properly normalized; with a non-packed duk_tval * these are essentially NOPs.

27000. (internal) no source string on stack

27001. * Function (method) calls

27002. plain

27003. Ecmascript string: CESU-8 / extended UTF-8 encoded

27004. internal flag: don't zero allocated buffer

27005. use strict (outer) context for global, eval, or function code

27006. in non-packed representation we don't care about which NaN is used

27007. * Union for accessing double parts, also serves as packed duk_tval

27008. include date part in string conversion result

27009. internal flag: dynamic buffer

27010. Ecmascript null

27011. enumerate non-enumerable properties in addition to enumerable

27012. **comment:** XXX: replace with RangeError?

label: code-design

27013. * Push operations * * Push functions return the absolute (relative to bottom of frame) * position of the pushed value for convenience. * * Note: duk_dup() is technically a push.

27014. string

27015. * ROM pointer compression

27016. * Union to access IEEE double memory representation, indexes for double * memory representation, and some macros for double manipulation. * * Also used by packed duk_tval. Use a union for bit manipulation to * minimize aliasing issues in practice. The C99 standard does not * guarantee that this should work, but it's a very widely supported * practice for low level manipulation. * * IEEE double format summary: * * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * See http://en.wikipedia.org/wiki/Double_precision_floating-point_format. * * NaNs are represented as exponent 0x7ff and mantissa != 0. The NaN is a * signaling NaN when the highest bit of the mantissa is zero, and a quiet * NaN when the highest bit is set. * * At least three memory layouts are relevant here: * * A B C D E F G H Big endian (e.g. 68k) DUK_USE_DOUBLE_BE * H G F E D C B A Little endian (e.g. x86) DUK_USE_DOUBLE_LE * D C B A H G F E Mixed/cross endian (e.g. ARM) DUK_USE_DOUBLE_ME * * ARM is a special case: ARM double values are in mixed/cross endian * format while ARM duk_uint64_t values are in standard little endian * format (H G F E D C B A). When a double is read as a duk_uint64_t * from memory, the register will contain the (logical) value * E F G H A B C D. This requires some special handling below. * * Indexes of various types (8-bit, 16-bit, 32-bit) in memory relative to * the logical (big endian) order: * * byte order duk_uint8_t duk_uint16_t duk_uint32_t * BE 01234567 0123 01 * LE 76543210 3210 10 * ME (ARM) 32107654 1032 01 * * Some processors may alter NaN values in a floating point load+store. * For instance, on X86 a FLD + FSTP may convert a signaling NaN to a * quiet one. This is catastrophic when NaN space is used in packed * duk_tval values. See: misc/clang_aliasing.c.

27017. * Helper macros for reading/writing memory representation parts, used * by duk_numconv.c and duk_tval.h.

27018. string conversion: use 'T' instead of '' as a separator

27019. Ecmascript number: double

27020. TypeError

27021. Support array for ROM pointer compression. Only declared when ROM * pointer compression is active.

27022. **comment:** Although extra/top could be an unsigned type here, using a signed type * makes the API more robust to calling code calculation errors or corner * cases (where caller might occasionally come up with negative values). * Negative values are treated as zero, which is better than casting them * to a large unsigned number. (This principle is used elsewhere in the * API too.)

label: code-design

27023. * Misc conversion

27024. Ecmascript object: includes objects, arrays, functions, threads

27025. URIError

27026. all doubles are considered normalized

27027. AUTHORS.rst

27028. day within month: 0 to 30

27029. setter: call is a time setter (affects hour, min, sec, ms); otherwise date setter (affects year, month, day-in-month)

27030. (internal) no filename on stack

27031. compile eval code (instead of global code)

27032. One problem with this macro is that expressions like the following fail * to compile: "(void) duk_error(...)". But because duk_error() is noreturn, * they make little sense anyway.

27033. set setter (given on value stack)

27034. prefer number, unless input is a Date, in which * case prefer string (E5 Section 8.12.8)

27035. Part indices for internal breakdowns. Part order from DUK_DATE_IDX_YEAR * to DUK_DATE_IDX_MILLISECOND matches argument ordering of Ecmascript API * calls (like Date constructor call). Some functions in duk_bi_date.c * depend on the specific ordering, so change with care. 16 bits are not * enough for all parts (year, specifically). * * (Must be in-sync with genbuiltins.py.)

27036. E == 0x7ff, F == 8 => normalized NaN

27037. Enumeration flags for duk_enum()

27038. year

27039. * Other state related functions

27040. Shared helper to generate DUK_OPT_xxx and DUK_USE_xxx documentation. XXX: unfinished placeholder

27041. Add a header snippet for detecting presence of DUK_OPT_xxx feature options which will be removed in Duktape 2.x.

27042. Add a header snippet for providing a __OVERRIDE_DEFINES__ section.

27043. XXX: unsigned constants?

27044. Globals holding scanned metadata, helper snippets, etc

27045. verbatim text for the entire line

27046. **comment:** Derived defines (DUK_USE_INTEGER_LE, etc) from DUK_USE_BYTEORDER. Duktape internals currently rely on the derived defines. This is after sanity checks because the derived defines are marked removed.

label: code-design

27047. Add original snippets. This fills in the required nodes recursively.

27048. **comment:** DUK_OPT_xxx always come from outside

label: code-design

27049. print('#snippet ' + sub_fn)

27050. already present

27051. **comment:** XXX: more checks

label: code-design

27052. Assume these provides come from outside.

27053. Feature selection, system include, Date provider Most #include statements are here

27054. Remaining tags in alphabetic order

27055. Automatic DUK_OPT_xxx feature option handling

27056. key is known

27057. Byte order and alignment defines are allowed to be missing, a fill-in will handle them. This is necessary because for some architecture byte order and/or alignment may vary between targets and may be software configurable.

27058. integer value

27059. Figure out fill-ins by looking for snippets not in original list and without any unserialized dependent nodes.

27060. **comment:** Autogeneration of option documentation
label: documentation

27061. duk_config.h generation

27062. print(repr(graph)) print(repr(snlist)) print('Resolved helper defines: %r' % resolved)

27063. Detect and reject 'fast math'

27064. **comment:** XXX: conditional warning, happens in some normal cases print('WARNING: define %r required, not provided so far' % k)
label: code-design

27065. Helper for building a text file from individual lines, injected files, etc. Inserted values are converted to Snippets so that their provides/requires information can be tracked. When non-C outputs are created, these will be bogus but ignored.

27066. Don't allow e.g. DUK_USE_ which results from matching DUK_USE_xxx

27067. NOTE: careful with Python equality, e.g. "0 == False" is true.

27068. Metadata to scan from config files.

27069. Generate a duk_config.h where platform, architecture, and compiler are all either autodetected or specified by user. Autodetection is based on a configured list of supported platforms, architectures, and compilers. For example, platforms.yaml defines the supported platforms and provides a helper define (DUK_F_xxx) to use for detecting that platform, and names the header snippet to provide the platform-specific definitions. Necessary dependencies (DUK_F_xxx) are automatically pulled in. Automatic "fill ins" are used for mandatory platform, architecture, and compiler defines which have a reasonable portable default. This reduces e.g. compiler-specific define count because there are a lot compiler macros which have a good default.

27070. **comment:** XXX: require automatic detection to be signaled? e.g. define DUK_USE_ALIGN_BY -1 define DUK_USE_BYTE_ORDER -1
label: code-design

27071. Development time helper: add DUK_ACTIVE which provides a runtime C string indicating what DUK_USE_xxx config options are active at run time. This is useful in genconfig development so that one can e.g. diff the active run time options of two headers. This is intended just for genconfig development and is not available in normal headers.

27072. Default value is false, and caller has emitted an unconditional #undef, so don't emit a duplicate

27073. Platform, architecture, compiler fillins. These are after all detection so that e.g. DUK_SPRINTF() can be provided by platform or compiler before trying a fill-in.

27074. Main

27075. **comment:** XXX: indicate feature option support, sanity checks enabled, etc in general summary of options, perhaps genconfig command line?
label: code-design

27076. sort under primary tag

27077. **comment:** for line in stripped_lines: print(line)
label: code-design

27078. **comment:** XXX: better to lookup compilers metadata
label: code-design

27079. Compiler files must provide at least these (additional checks in validate_compiler_file()). Fill-ins provide missing optionals.

27080. Header snippet representation: lines, provides defines, requires defines.

27081. **comment:** Find a header snippet which provides the missing define. Some DUK_F_xxx files provide multiple defines, so we don't necessarily know the snippet filename here.
label: code-design

27082. signal to fillin

27083. print('Tags: %r' % use_tags_list)

27084. must be #define'd

27085. Compatibility with Duktape 1.3

27086. compiling Duktape from a single source file (duktape.c) version compiling Duktape (not user application) artifact, include guard

27087. Forced options from multiple sources are gathered into a shared list so that the override order remains the same as on the command line.

27088. Avoid ambiguous hex escapes

27089. Snippet provides it's own require; omit

27090. print('config option %s not covered by manual snippets, emitting default automatically' % k)

27091. **comment:** XXX: rst or other format
label: code-design

27092. C string value

27093. Platform files must provide at least these (additional checks in validate_platform_file()). Fill-ins provide missing optionals.

27094. shallow copy

27095. Architecture files must provide at least these (additional checks in validate_architecture_file()). Fill-ins provide missing optionals.

27096. Generate a duk_config.h header with platform, compiler, and architecture either autodetected (default) or specified by user. Support for autogenerated DUK_OPT_xxx flags is also selected by user.

27097. **comment:** XXX: aware of target version
label: code-design

27098. Preferred tags first

27099. snippet list

27100. **comment:** XXX: better to lookup architectures metadata
label: code-design

27101. graph[A] = [B, ...] <-> B, ... provide something A requires.

27102. x is a Snippet

27103. Helper headers snippets.

27104. lines of text and/or snippets map from define to 'True' for now map from define to 'True' for now

27105. **comment:** Insert missing define dependencies into index 'idx_deps' repeatedly until no unsatisfied dependencies exist. This is used to pull in the required DUK_F_xxx helper defines without pulling them all in. The resolution mechanism also ensures dependencies are pulled in the correct order, i.e. DUK_F_xxx helpers may depend on each other (as long as there are no circular dependencies). XXX: this can be simplified a lot
label: code-design

27106. strip last newline to avoid empty line

27107. **comment:** DUK_USE_xxx are internal and they should not be 'requirements'
label: code-design

27108. This is a fallback in case config option metadata is wrong.

27109. byteorder provided by all architecture files alignment provided by all architecture files packed tval provided by all architecture files

27110. print('Autoscanning snippet: %s' % fn)

27111. for printing only

27112. **comment:** XXX: detect and handle loops cleanly
label: code-design

27113. Emit a default #define / #undef for an option based on a config option metadata node (parsed YAML doc).

27114. Object layout

27115. Careful with order, snippet may self-reference its own defines in which case there's no outward dependency. (This is not 100% because the order of require/provide matters and this is not handled now.) Also, some snippets may #undef/#define another define but they don't "provide" the define as such. Such redefinitions are marked /* redefine */ in the snippets. They're best avoided (and not currently needed in Duktape 1.4.0).

27116. **comment:** Add a header snippet for checking consistency of DUK_USE_xxx config options, e.g. inconsistent options, invalid option values.

label: code-design

27117. Helpers for duk_config.h generation

27118. must be #define'd may be #undef'd, as long as provided may be #undef'd, as long as provided

27119. Snippet may have further unresolved provides; add recursively

27120. For some options like DUK_OPT_PACKED_TVAL the default comes from platform definition.

27121. **comment:** XXX: assume no newlines etc

label: code-design

27122. Date provider snippet is after custom header and overrides, so that the user may define e.g. DUK_USE_DATE_NOW_GETTIMEOFDAY in their custom header.

27123. **comment:** /usr/bin/env python2 Process Duktape option metadata and produce various useful outputs: - duk_config.h with specific or autodetected platform, compiler, and architecture; forced options; sanity checks; etc - option documentation for Duktape 1.x feature options (DUK_OPT_xxx) - option documentation for Duktape 1.x/2.x config options (DUK_USE_xxx) Genconfig tries to build all outputs based on modular metadata, so that managing a large number of config options (which is hard to avoid given the wide range of targets Duktape supports) remains maintainable. Genconfig does *not* try to support all exotic platforms out there. Instead, the goal is to allow the metadata to be extended, or to provide a reasonable starting point for manual duk_config.h tweaking. For Duktape 1.3 release the main goal was to autogenerate a Duktape 1.2 compatible "autodetect" header from legacy snippets, with other outputs being experimental. For Duktape 1.4 duk_config.h is always created from modular sources.

label: code-design

27124. **comment:** Not exact but close enough. Doesn't handle string literals etc, but these are not a concrete issue for scanning preprocessor #define references. Comment contents are stripped of any DUK_ prefixed text to avoid incorrect requires/provides detection. Other comment text is kept; in particular a /* redefine */ comment must remain intact here. (The 'redefine' hack is not actively needed now.) Avoid Python 2.6 vs. Python 2.7 argument differences.

label: code-design

27125. If manually-edited snippets don't #define or #undef a certain config option, emit a default value here. This is useful to fill-in for new config options not covered by manual snippets (which is intentional).

27126. position where to emit DUK_F_xxx dependencies

27127. **comment:** XXX: platform/compiler could provide types; if so, need some signaling defines like DUK_F_TYPEDEFS_DEFINED

label: code-design

27128. **comment:** Compilers need a lot of defines; missing defines are automatically filled in with defaults (which are mostly compiler independent), so the requires define list is not very large.

label: code-design

27129. **comment:** C preprocessor '#warning' is often supported

label: code-design

27130. print('Resolving %r' % k)

27131. Miscellaneous helpers

27132. If a related feature option exists, it can be used to force enable/disable the target feature. If neither feature option (DUK_OPT_xxx or DUK_OPT_NO_xxx) is given, revert to default.

27133. print('Deleting temporary directory: %r' % dirname)

27134. at least one other node provides 'k'

27135. Preferred tag order for generated C header files.

27136. **comment:** XXX: better plumbing for lookup path

label: code-design

27137. Add automatic DUK_OPT_XXX and DUK_OPT_NO_XXX handling for backwards compatibility with Duktape 1.2 and before.

27138. DUK_SETJMP, DUK_LONGJMP, DUK_JMPBUF_TYPE are optional, fill-in provides if none defined.

27139. for trailing newline

27140. Forced options, last occurrence wins (allows a base config file to be overridden by a more specific one).

27141. DUK_F_xxx snippets

27142. emit tag heading only if there are subsections

27143. **comment:** XXX: placeholder, need to decide on markup conventions for YAML files

label: code-design

27144. Number types

27145. print('REQUIRES: %r' % m)

27146. **comment:** XXX: better to lookup platforms metadata

label: code-design

27147. verbatim value

27148. **comment:** uppercase only, don't match DUK_USE_xxx for example

label: code-design

27149. **comment:** XXX: assume no newlines etc XXX: support compiler specific warning mechanisms

label: code-design

27150. print('Effective tag order: %r' % tags)

27151. **comment:** Don't allow e.g. DUK_USE_ which results from matching DUK_USE_xxx print('PROVIDES: %r' % m.group(1))

label: code-design

27152. DUK_F_UCLIBC is special because __UCLIBC__ is provided by an #include file, so the check must happen after platform includes. It'd be nice for this to be automatic (e.g. DUK_F_UCLIBC.h.in could indicate the dependency somehow).

27153. Emit forced options. If a corresponding option is already defined by a snippet above, #undef it first.

27154. DLL build affects visibility attributes on Windows but unfortunately cannot be detected automatically from preprocessor defines or such. DLL build status is hidden behind DUK_F_DLL_BUILD and there are two ways for that to be set: - Duktape 1.3 backwards compatible DUK_OPT_DLL_BUILD - Genconfig --dll option

27155. Preferred tag order for option documentation.

27156. XXX: check that YAML parses

27157. Pretty print a debugger command.

27158. Duktape internal class numbers, must match C headers

27159. Write a string into a buffer interpreting codepoints U+0000...U+00FF * as bytes. Drop higher bits.

27160. **comment:** XXX: add constants inline to preformatted output (e.g. for strings, add a short escaped snippet as a comment on the line after the compact argument list).

label: code-design

27161. absFn -> true

27162. Errors

27163. true

27164. We shouldn't come here, but if we do, JSON is a reasonable default.

27165. Could emit a 'debug-value' event here, but that's not necessary because the receiver will just collect statistics which can also be done using the finished message.

27166. **comment:** XXX: move this to the web UI so that the UI can control what locals are listed (or perhaps show locals for all levels with an expandable tree view).

label: code-design

27167. There is no close() or destroy() for a passthrough stream, so just close the outputParser which will cancel timers etc.

27168. Raw bytes Unicode string

27169. Still waiting for version identification to complete.

27170. **comment:** Pending flags are used to avoid requesting the same thing twice while a previous request is pending. The flag-based approach is quite awkward. Rework to use promises.

label: code-design

27171. 2-byte string

27172. undefined

27173. * JSON debug proxy
27174. msg.varname is a proper Unicode strings here, and needs to be converted into a protocol string (U+0000...U+00FF).
27175. new object, old may be in circulation for a while
27176. list of command names, merged client/target
27177. XXX: normalize last newline (i.e. force a newline if contents don't end with a newline)?
27178. accumulate data accumulated message until EOM
27179. A PutVar call quite possibly changes the local variables so always re-read locals afterwards. We don't need to wait for PutVar to complete here; the requests will pipeline automatically and be executed in order.
27180. 2-byte buffer
27181. './foo/bar.js' -> 'foo/bar.js'
27182. whole
27183. Commands initiated by Duktape
27184. Not all streams will emit this.
27185. Pretty print a dvalue. Useful for dumping etc.
27186. Could also check for an empty request queue, but that's probably too strict?
27187. Technically we should wait for each delbreak reply but because target processes the requests in order, it doesn't matter.
27188. Protocol version handling. When dumping an output stream, the caller gives a non-null protocolVersion so we don't read one here.
27189. **comment:** Check if 'msg' would encode to the same JSON which was previously sent to the web client. The caller then avoid resending unnecessary stuff.
 label: code-design
27190. Up-to-date list of breakpoints on target
27191. **comment:** XXX: do pretty printing in debug client for now
 label: code-design
27192. console.log(JSON.stringify(msg));
27193. Duktape heapdrt type constants, must match C headers
27194. * Debugger implementation * * A debugger instance communicates with the debug target and maintains * persistent debug state so that the current state can be resent to the * socket.io client (web UI) if it reconnects. Whenever the debugger state * is changed an event is generated. The socket.io handler will listen to * state change events and push the necessary updates to the web UI, often * in a rate limited fashion or using a client pull to ensure the web UI * is not overloaded. * * The debugger instance assumes that if the debug protocol connection is * re-established, it is always to the same target. There is no separate * abstraction for a debugger session.
27195. error
27196. Decode and normalize source file contents: UTF-8, tabs to 8, * CR LF to LF.
27197. **comment:** unused/none
 label: code-design
27198. * Command line parsing and initialization
27199. "notify" can be a string or "true"
27200. Shouldn't come here.
27201. 4-byte signed integer
27202. For the unquoted version we don't need to escape single or double quotes.
27203. unsigned
27204. msg.input is a proper Unicode strings here, and needs to be converted into a protocol string (U+0000...U+00FF).
27205. nop: no callback
27206. <https://github.com/ryanmcgrath/wrench-js>
27207. varvalue is JSON parsed by the web UI for now, need special string encoding here.
27208. Command line options (defaults here, overwritten if necessary)
27209. truncate higher bits
27210. No actual requests sent by the target right now (just notifies).
27211. Encode an ordinary Unicode string into a dvalue compatible format, i.e. * into a byte array represented as codepoints U+0000...U+00FF. Concretely, * encode with UTF-8 and then represent the bytes with U+0000...U+00FF.
27212. console.log('Received json proxy input line: ' + line.toString('utf8'));
27213. * Miscellaneous helpers
27214. indicate 'v' is actually set
27215. Dump effective options. Also provides a list of option names.
27216. Although the underlying transport may not have a close() or destroy() method or even a 'close' event, this method is always available and will generate a 'transport-close'. The caller is responsible for closing the underlying stream if that is necessary.
27217. A byline-parser is simple and good enough for now (assume compact JSON with no newlines).
27218. override
27219. update run state now that we're paused
27220. duk_tval number, any IEEE double decode into hex big endian ieee double
27221. just trigger a sync, gets rate limited
27222. **comment:** Convert a buffer into a string using Unicode codepoints U+0000...U+00FF. * This is the NodeJS 'binary' encoding, but since it's being deprecated, * reimplement it here. We need to avoid parsing strings as e.g. UTF-8: * although Duktape strings are usually UTF-8/CESU-8 that's not always the * case, e.g. for internal strings. Buffer values are also represented as * strings in the debug protocol, so we must deal accurately with arbitrary * byte arrays.
 label: code-design
27223. Marker objects for special protocol values
27224. Fetch basic info right away
27225. **comment:** Pretty print a dvalue for UI usage. Everything comes out as a ready-to-use * string. * * XXX: Currently the debug client formats all values for UI use. A better * solution would be to pass values in typed form and let the UI format them, * so that styling etc. could take typing into account.
 label: code-design
27226. An eval call quite possibly changes the local variables so always re-read locals afterwards. We don't need to wait for Eval to complete here; the requests will pipeline automatically and be executed in order.
27227. Commands initiated by the debug client (= us)
27228. res.status(200).json(val);
27229. raw identification string
2730. Create debugger and web UI singletons, tie them together and start them.
27231. * Debug protocol parser * * The debug protocol parser is an EventEmitter which parses debug messages * from an input stream and emits 'debug-message' events for completed * messages ending in an EOM. The parser also provides debug dumping, stream * logging functionality, and statistics gathering functionality. * * This parser is used to parse both incoming and outgoing messages. For * outgoing messages the only function is to validate and debug dump the * messages we're about to send. The downside of dumping at this low level * is that we can't match request and reply/error messages here. * * <http://www.sitepoint.com/nodejs-events-and-eventemitter/>
27232. Pretty print a dvalue string (bytes represented as U+0000...U+00FF) * for UI usage without quotes.
27233. Loose matching is tempting but counterproductive: filenames must match 1:1 between the debug client and the debug target for e.g. breakpoints to work as expected. Note that a breakpoint may be assigned by selecting a file from a dropdown populated by scanning the filesystem for available sources and there's no way of knowing if the debug target uses the exact same name.
27234. web UI singleton transport connection to target dummy passthrough for message dumping parser for incoming debug messages parser for outgoing debug messages (stats, dumping)
27235. **comment:** XXX: make the client do this?
 label: code-design

27236. // This fails with "RangeError: Maximum call stack size exceeded" for some // reason, so use a much slower variant. for (i = 0, n = buf.length; i < n; i++) { cp[i] = buff[i]; } return String.fromCharCode.apply(String, cp);
27237. We require a destroy() method from the actual target stream
27238. Debug protocol integer
27239. shift forces unsigned
27240. Send a status request to trigger a status notify, result is ignored: target sends a status notify instead of a meaningful reply
27241. See doc/debugger.rst for format description.
27242. Pretty print a dvalue string (bytes represented as U+0000...U+00FF) * for UI usage. Try UTF-8 decoding to get a nice Unicode string (JSON * encoded) but if that fails, ensure that bytes are encoded transparently. * The result is a quoted string with a special quote marker for a "raw" * string when UTF-8 decoding fails. Very long strings are optionally * clipped.
27243. console.dir(argv);
27244. Intentional invalid command
27245. When we fall back to representing bytes, indicate that the string is "raw" with a 'r"' prefix (a somewhat arbitrary convention). U+0022 = ", U+0027 = '
27246. number (IEEE double), big endian
27247. We want the fileMap to contain the filename relative to the search dir root.
27248. Initial debugger connection
27249. Represented signed 32-bit integers as plain integers. Debugger code expects this for all fields that are not duk_tval representations (e.g. command numbers and such).
27250. Right now bytecode is a string containing a direct dump of the bytecode in target endianness. Decode here so that the web UI doesn't need to.
27251. **comment:** The value key should not be used programmatically, it is just there to make the dumps more readable.
 label: code-design
27252. msg.varname and msg.varvalue are proper Unicode strings here, they need to be converted into protocol strings (U+0000...U+00FF).
27253. Note: typeof null === 'object', so null special case explicitly
27254. Parse complete dvalues (quite inefficient now) by trial parsing. Consume a value only when it's fully present in 'buf'. See doc/debugger.rst for format description.
27255. 0x80...0xbf: integers 0-63
27256. pointer
27257. * Source file manager * * Scan the list of search directories for Ecmascript source files and * build an index of them. Provides a mechanism to find a source file * based on a raw 'fileName' property provided by the debug target, and * to provide a file list for the web UI. * * NOTE: it's tempting to do loose matching for filenames, but this does * not work in practice. Filenames must match 1:1 with the debug target * so that e.g. breakpoints assigned based on filenames found from the * search paths will match 1:1 on the debug target. If this is not the * case, breakpoints won't work as expected.
27258. not cached, send (cache already updated)
27259. CUSTOMTRANSPORT: to use a custom transport, change this.targetStream to use your custom transport.
27260. Resend all debugger state for new client clear client state cache
27261. Parse arguments.
27262. not negative zero
27263. stats and dumping
27264. map from (merged) command name to number
27265. stats for current debug connection
27266. "request" can be a string or "true"
27267. Use a PassThrough stream to debug dump and get stats for output messages. Simply write outgoing data to both the targetStream and this.passsthrough separately.
27268. object
27269. * Debugger output formatting
27270. Read final, effective breakpoints from the target
27271. cached
27272. null
27273. **comment:** XXX: unprintable characters etc? In some UI cases we'd want to e.g. escape newlines and in others not.
 label: code-design
27274. 4-byte string
27275. String map for commands (debug dumping). A single map works (instead of separate maps for each direction) because command numbers don't currently overlap. So merge the YAML metadata.
27276. nop
27277. lightfunc
27278. false
27279. stream is closed/broken, don't parse anymore
27280. trailing "" intentionally missing
27281. heapptr
27282. Explicit rate limiter because this is a source of a lot of traffic.
27283. * Minimal debug web console for Duktape command line tool * * See debugger/README.rst. * * The web UI socket.io communication can easily become a bottleneck and * it's important to ensure that the web UI remains responsive. Basic rate * limiting mechanisms (token buckets, suppressing identical messages, etc) * are used here now. Ideally the web UI would pull data on its own terms * which would provide natural rate limiting. * * Promises are used to structure callback chains. * * <https://github.com/petkaantonov/bluebird> * <https://github.com/petkaantonov/bluebird/blob/master/API.md> * <https://github.com/petkaantonov/bluebird/wiki/Promise-anti-patterns>
27284. Pretty print a debugger message given as an array of parsed dvalues. * Result should be a pure ASCII one-liner.
27285. Constants
27286. Not possible in practice.
27287. * Express setup and socket.io
27288. Represent non-integers as IEEE double dvalues
27289. UTF-8
27290. **comment:** console logging is done at a higher level to match request/response
 label: code-design
27291. Ignore unknown notify messages
27292. console.log(msg);
27293. 0x60...0x7f: strings with length 0-31
27294. Delete matching breakpoints in reverse order so that indices remain valid. We do this for all operations so that duplicates are eliminated if present.
27295. 0xc0...0xff: integers 0-16383
27296. close previous target connection
27297. 4-byte buffer
27298. inform web UI
27299. XXX: signal success to UI?
27300. Bytecode opcode/extracode metadata
27301. Token bucket rate limiter for a given callback. Calling code calls * trigger() to request 'cb' to be called, and the rate limiter ensures * that 'cb' is not called too often.
27302. Poll various state items when running
27303. pc is preincremented before adding
27304. cache to avoid resending identical data
27305. Here utf8.decode() is better than decoding using NodeJS buffer operations because we want strict UTF-8 interpretation.
27306. Right now the implementation is setInterval-based, but could also be made timerless. There are so few rate limited resources that this doesn't matter in practice.
27307. Value could be a Node.js buffer directly, but we prefer all dvalues to be JSON compatible
27308. this.targetStream.destroy();

27309. nop
27310. Don't add a 'value' key to numbers.
27311. preformatted dvalues
27312. used to flag special values like undefined
27313. **comment:** Pretty print a number for UI usage. Types and values should be easy to * read and typing should be obvious. For numbers, support Infinity, NaN, * and signed zeroes properly.
label: code-design
27314. no bias in LDINTX
27315. debugger singleton current socket (or null)
27316. Try to detach cleanly, timeout if no response
27317. 4-byte signed integer
27318. heapptr
27319. plain buffer
27320. * Target binary connection handler
27321. number (IEEE double), big endian
27322. Encode arguments into dvalues.
27323. **comment:** XXX: write a notify to target?
label: code-design
27324. disconnect JSON client too (if not already disconnected)
27325. * JSON proxy server * * Accepts an incoming JSON proxy client and connects to a debug target, * tying the two connections together. Supports both a single connection * and a persistent mode.
27326. true
27327. truncate higher bits
27328. * JSON connection handler
27329. Represented signed 32-bit integers as plain integers. Debugger code expects this for all fields that are not duk_tval representations (e.g. command numbers and such).
27330. Not possible in practice.
27331. * Main
27332. indicate 'v' is actually set
27333. **comment:** The value key should not be used programmatically, it is just there to make the dumps more readable.
label: code-design
27334. 2-byte string
27335. Once we're connected to the target, start read both binary and JSON input. We don't want to read JSON input before this so that we can always translate incoming messages to dvalues and write them out without queueing. Any pending JSON messages will be queued by the OS instead.
27336. Generic handshake format: only relies on initial version field.
27337. undefined
27338. 0x80...0xbf: integers 0-63
27339. pointer
27340. 0x60...0x7f: strings with length 0-31
27341. Represent non-integers as IEEE double dvalues.
27342. 0xc0...0xff: integers 0-16383
27343. Trial parse dvalue(s) and debug messages.
27344. default logger log.l = 0; // enable debug and trace logging
27345. 4-byte buffer
27346. **comment:** for manual testing of binary/json parsing robustness
label: code-design
27347. 2-byte buffer
27348. Parse message type, determine initial marker for binary message.
27349. Note that typeof null === 'object'.
27350. * Config
27351. * Misc helpers
27352. **comment:** * JSON debug proxy written in DukLuv * * This single file JSON debug proxy implementation is an alternative to the * Node.js-based proxy in duk_debug.js. DukLuv is a much smaller dependency * than Node.js so embedding DukLuv in a debug client is easier.
label: code-design
27353. Feed the data one byte at a time when torture testing.
27354. When this is invoked the proxy connection and the target connection have both been closed.
27355. Add an EOM, and write out the dvalues to the debug target.
27356. * Detect missing 'var' declarations
27357. whole
27358. not negative zero
27359. Receive data into 'incoming', resizing as necessary.
27360. explicit flag for e.g. v === undefined
27361. disconnect target too (if not already disconnected)
27362. **comment:** XXX: it'd be nice to log remote peer host:port
label: code-design
27363. make a copy, ensuring there's no slice offset get underlying plain buffer
27364. Trial parse JSON message(s).
27365. This is still pretty awkward in Duktape 1.4.x. Argument may be a "slice" and we want a copy of the slice (not the full underlying buffer).
27366. **comment:** XXX: Code assumes uv.write() will write fully. This is not necessarily true; should add support for partial writes (or at least failing when a partial write occurs).
label: code-design
27367. Prevent new bindings on global object. This detects missing 'var' declarations, e.g. "x = 123;" in a function without declaring it.
27368. object
27369. lightfunc
27370. skip dukluv and script name
27371. null
27372. More detailed v1 handshake line.
27373. **comment:** unnecessary but just in case
label: code-design
27374. 4-byte string
27375. **comment:** unused/none
label: code-design
27376. don't register any sockets/timers etc to exit
27377. **comment:** In lenient mode if JSON parse fails just send back an _Error and ignore the line (useful for initial development). In non-lenient mode drop the connection here; if the failed line was a request the client is expecting a reply/error message back (otherwise it may go out of sync) but we can't send a synthetic one (as we can't parse the request).
label: code-design
27378. Trial parse handshake unless done.

27379. false
27380. !/usr/bin/env python2 Merge debugger YAML metadata files and output a merged JSON metadata file.
27381. #exec-status
27382. #center-area
27383. #part-middle
27384. #part-footer
27385. ?
27386. No source loaded
27387. #about-dialog
27388. #part-header
27389. #left-area
27390. #right-area
27391. **comment:** XXX: how to minimize the chance we'll further communicate with the server or reconnect to it? socket.reconnection()
label: code-design
27392. **comment:** XXX: prevent retry of no-such-file by negative caching?
label: code-design
27393. Eval may take seconds to complete so indicate it is pending.
27394. Bytecode dialog highlight
27395. If previous update is pending, abort and start a new one.
27396. <http://diveintohtml5.info/storage.html>
27397. Note: loadedFileName can be either from target or from server, but they must match exactly. We could do a loose match here, but exact matches are needed for proper breakpoint handling anyway.
27398. Remove eval button "pulsating" glow when we get a result
27399. may be null
27400. Remove pending highlight once we're no longer running.
27401. Update the "console" output based on lines sent by the server. The server rate limits these updates to keep the browser load under control. Even better would be for the client to pull this (and other stuff) on its own.
27402. * Duktape debugger web client * * Talks to the NodeJS server using socket.io. * * <http://unixpapa.com/js/key.html>
27403. If we just became paused, check for eval watch
27404. Make copies of the requested file/line so that we have the proper values in case they've changed.
27405. * UI update handling when exec-status update arrives
27406. Active breakpoints follow
27407. Exact match is required.
27408. **comment:** XXX: any faster way; elems doesn't have e.g. indexOf()
label: code-design
27409. **comment:** XXX: ignore issue with last empty line for now
label: code-design
27410. returns a Manager
27411. Update interval for custom source highlighting.
27412. Duktape now restricts execution status updates quite effectively so there's no need to rate limit UI updates now.
27413. **comment:** XXX: hacky transition, make source change visible
label: code-design
27414. puff slide, puff
27415. onclick handler for exec status text
27416. Source is updated periodically. Other code can also call doSourceUpdate() directly if an immediate update is needed.
27417. * AJAX request handling to fetch source files
27418. **comment:** XXX: error transition here
label: requirement
27419. If we're executing the file shown, highlight current line
27420. **comment:** Scroll to requested line. This is not very clean, so a better solution should be found: <https://developer.mozilla.org/en-US/docs/Web/API/Element.scrollIntoView> <http://erraticdev.blogspot.fi/2011/02/jquery-scroll-into-view-plugin-with.html> <http://flesler.blogspot.fi/2007/10/jqueryscrollto.html>
label: code-design
27421. Make source window grey when running for a longer time, use a small delay to avoid flashing grey when stepping.
27422. **comment:** Source view file that we want to be loaded in source view scroll to line once file has been loaded line that we want to highlight (if any) currently loaded (shown) file currently loaded file line count true if currFileName (loosely) matches loadedFileName if set, scroll loaded file to requested line highlight line line numbers which have been modified (added classes etc, tracked for removing) timer for updating source view current AJAX request for fetching a source file (if any) hack to reset button states bytecode dialog active index of currently highlighted line (or null) index to first line of bytecode instructions
label: code-design
27423. Remove previously added custom classes
27424. Not really interesting in the UI \$('#server-info').text(new Date() + ' : ' + JSON.stringify(msg));
27425. Not 100% reliable if callstack has several functions of the same name
27426. If no-one requested us to scroll to a specific line, finish.
27427. Update buttons
27428. <http://stackoverflow.com/questions/14918787/jquery-scroll-to-bottom-of-div-even-after-it-updates> Stop queued animations so that we always scroll quickly to bottom
27429. First line is special
27430. AJAX request for the source.
27431. * Initialization
27432. About dialog, shown automatically on first startup.
27433. Enter handling for eval input <https://forum.jquery.com/topic/bind-html-input-to-enter-key-keypress>
27434. Eval watch handling
27435. We'd like to window.close() here but can't (not allowed from scripts). Alert is the next best thing.
27436. float
27437. Bytecode dialog
27438. Force source view to match currFileName only when running or when just became paused (from running or detached).
27439. nop
27440. Force line update (scrollTop) only when running or just became paused. Otherwise let user browse and scroll source files freely.
27441. Not worth alerting about because source fetch errors happen all the time, e.g. for dynamically evaluated code.
27442. Execution state previous execution state ('paused', 'running', etc) previous debugger attached state (true, false, null) current filename being executed current function name being executed current line being executed current bytecode PC being executed current execution state ('paused', 'running', 'detached', etc) current debugger attached state (true or false) current local variables current callstack (from top to bottom) current breakpoints timestamp when last started running (if running) (used to grey out the source file if running for long enough)
27443. * Source view periodic update handling
27444. If we just started running, store a timestamp so we can grey out the source view only if we execute long enough (i.e. we're not just stepping).
27445. **comment:** The variable value is parsed as JSON right now, but it'd be better to also be able to parse buffer values etc.
label: code-design
27446. * Init socket.io and add handlers

27447. don't clear eval input
27448. * AJAX request for fetching the source list
27449. This is worth alerting about as the UI is somewhat unusable if we don't get a source list.
27450. Update current execution state
27451. Add breakpoints
27452. * Helpers
27453. Pause may take seconds to complete so indicate it is pending.
27454. This must fit into the smallest pool entry.
27455. Define to enable some debug printfs.
27456. Use 'st' as udata.
27457. **comment:** #define DUK_ALLOC_HYBRID_DEBUG
 label: code-design
27458. **comment:** * Example memory allocator with pool allocation for small sizes and * fallback into malloc/realloc/free for larger sizes or when the pools * are exhausted. * * Useful to reduce memory churn or work around a platform allocator * that doesn't handle a lot of small allocations efficiently.
 label: code-design
27459. Still fits, no shrink support.
27460. 'ptr' cannot be NULL.
27461. DUK_ALLOC_HYBRID_H_INCLUDED
27462. **comment:** The double value in the union is there to ensure alignment is * good for IEEE doubles too. In many 32-bit environments 4 bytes * would be sufficiently aligned and the double value is unnecessary.
 label: code-design
27463. Handle the ptr-NULL vs. size-zero cases explicitly to minimize * platform assumptions. You can get away with much less in specific * well-behaving environments.
27464. **comment:** * Example memory allocator with machine parseable logging. * * Also sizes for reallocs and frees are logged so that the memory * behavior can be essentially replayed to accurately determine e.g. * optimal pool sizes for a pooled allocator. * * Allocation structure: * * [alloc_hdr | user area] * * ^ * | --- pointer returned to Duktape * --- underlying malloc ptr
 label: code-design
27465. Suppress warning.
27466. **comment:** DUK_ALLOC_LOGGING_H_INCLUDED
 label: code-design
27467. Note: ajduk doesn't log oldsize (uses -1 instead)
27468. **comment:** !/usr/bin/env python2 Analyze allocator logs and write total-bytes-in-use after every operation to stdout. The output can be gnuplotted as: \$ python log2gnuplot.py </tmp/duk-alloc-log.txt >/tmp/output.txt \$ gnuplot > plot "output.txt" with lines
 label: code-design
27469. A ptr/NULL/FAIL size F ptr/NULL size R ptr/NULL oldsize ptr/NULL/FAIL newsize
27470. **comment:** * Example torture memory allocator with memory wiping and check for * out-of-bounds writes. * * Allocation structure: * * [alloc_hdr | red zone before | user area | red zone after] * * ^ * | --- pointer returned to Duktape * --- underlying malloc ptr
 label: code-design
27471. Handle the ptr-NULL vs. size-zero cases explicitly to minimize * platform assumptions. You can get away with much less in specific * well-behaving environments.
27472. **comment:** The double value in the union is there to ensure alignment is * good for IEEE doubles too. In many 32-bit environments 4 bytes * would be sufficiently aligned and the double value is unnecessary.
 label: code-design
27473. Suppress warning.
27474. Force address change on every realloc.
27475. DUK_ALLOC_TORTURE_H_INCLUDED
27476. * Enter interactive mode if options indicate it
27477. [... bytecode_filename src_data src_len filename]
27478. * Main
27479. Example of sending an application specific debugger notification.
27480. 'p' points to a NUL (p == p_end) or a period.
27481. avoid SIGPIPE killing process
27482. Full completion, add a period, e.g. input 'Math' -> 'Math.'
27483. Here we know the eval arg exists but check anyway
27484. DUK_CMDLINE_RLIMIT
27485. * Misc helpers
27486. A virtual '/tmp' exists by default: * <https://gist.github.com/evanw/e6be28094f34451bd5bd#file-temp-js-L3806-L3809>
27487. DUK_CMDLINE_LINENOISE_COMPLETION
27488. Callback should avoid errors for now, so use * duk_check_stack() rather than duk_require_stack().
27489. Manual test for bytecode dump/load cycle: dump and load before * execution. Enable manually, then run "make qecmatest" for a * reasonably good coverage of different functions and programs.
27490. try { FS.unmount("/"); } catch (e) { console.log("Failed to unmount default '/' MEMFS mount: " + e); }
27491. original, including partial last component
27492. Ensure socket is closed even when detach is initiated by Duktape * rather than debug client.
27493. * Minimal Linenoise completion support
27494. caller coercers
27495. [... bytecode_filename src_data src_len function]
27496. Optional bytecode dump.
27497. Print error objects with a stack trace specially. * Note that getting the stack trace may throw an error * so this also needs to be safe call wrapped.
27498. * Duktape heap lifecycle
27499. Manual test for duk_debugger_cooperate()
27500. 'idx_obj' points to the object matching the last * full component, use [p_curr,p] as a filter for * that object.
27501. Running stdin like a full file (reading all lines before * compiling) is useful with emduk: * cat test.js | ./emduk --run-stdin
27502. should never happen, but just in case
27503. Catches e.g. 'foo..bar' -> we want 'bar' only.
27504. Print error to stderr and pop error.
27505. * Signal handling setup
27506. ~2 GB
27507. **comment:** * Cleanup and exit
 label: code-design
27508. Partial ends in a period, e.g. 'Math.' -> complete all Math properties.
27509. fatal_handler
27510. **comment:** Very simplified "is identifier part" check.
 label: code-design
27511. Read until EOF, avoid fseek/stat because it won't work with stdin.
27512. global
27513. Defined in duk_cmdline_ajduk.c or alljoyn.js headers.
27514. **comment:** This is not necessary but should be harmless.

label: code-design

27515. 'p' will either be p_start - 1 (ran out of buffer) or point to * the first offending character.

27516. nrets

27517. **comment:** XXX: Here it'd be nice to get some stats for the compilation result * when a suitable command line is given (e.g. code size, constant * count, function count. These are available internally but not through * the public API.**label:** code-design

27518. In non-interactive mode, success results are not written at all. * It is important that the result value is not string coerced, * as the string coercion may cause an error in some cases.

27519. * In interactive mode, write to stdout so output won't * interleave as easily. * * NOTE: the ToString() coercion may fail in some cases; * for instance, if you evaluate: * * ({valueOf: function() {return {}}, * toString: function() {return {}}}); * * The error is: * * TypeError: failed to coerce with [[DefaultValue]] * duk_api.c:1420 * * These are handled now by the caller which also has stack * trace printing support. User code can print out errors * safely using duk_safe_to_string().

27520. DUK_CMDLINE_FILEIO

27521. **comment:** +25% and some extra**label:** code-design

27522. * Command line execution tool. Useful for test cases and manual testing. * * To enable linenoise and other fancy stuff, compile with -DDUK_CMDLINE_FANCY. * It is not the default to maximize portability. You can also compile in * support for example allocators, grep for DUK_CMDLINE_*.

27523. get_value

27524. nret

27525. for properties of plain strings etc

27526. Not an Error instance, don't read "stack".

27527. 'this' binding

27528. Source code.

27529. **comment:** XXX: handle partial writes**label:** code-design

27530. DUK_CMDLINE_SIGNAL

27531. **comment:** Emscripten specific: stdin EOF doesn't work as expected. * Instead, when 'emduk' is executed using Node.js, a file * piped to stdin repeats (!). Detect that repeat and cut off * the stdin read. Ensure the loop repeats enough times to * avoid detecting spurious loops. * * This only seems to work for inputs up to 256 bytes long.**label:** code-design

27532. skip code

27533. * Simple file read/write bindings

27534. * Usage

27535. **comment:** XXX: There's no key quoting now, it would require replacing the * last component with a ['foo\bar'] style lookup when appropriate.**label:** code-design

27536. 'p' now points to a string of the form 'foo.bar.quux'. Look up * all the components except the last; treat the last component as * a partial name which is used as a filter for the previous full * component. All lookups are from the global object now.

27537. p_curr == p_end allowed on purpose, to handle 'Math.' for example.

27538. For automatic reattach testing.

27539. Not found.

27540. Use duk_compile_lstring_filename() variant which avoids interning * the source code. This only really matters for low memory environments.

27541. heap_udata: ignored by AjsHeap, use as marker

27542. * Memory limit

27543. fall thru

27544. original, e.g. 'Math.'

27545. DUK_CMDLINE_LINENOISE

27546. * Execute from file handle etc

27547. * Create heap

27548. completion to last component

27549. 128 MB

27550. signal(SIGPIPE, SIG_IGN);

27551. EMSCRIPTEN

27552. Bytecode.

27553. in interactive mode, write to stdout

27554. Try to use NODEFS to provide access to local files. Mount the * CWD as /working, and then prepend "/working/" to relative native * paths in file calls to get something that works reasonably for * relative paths. Emscripten doesn't support replacing virtual * "/" with host "/" (the default MEMFS at "/" can't be unmounted) * but we can mount "/tmp" as host "/tmp" to allow testcase runs. * * https://kripken.github.io/emscripten-site/docs/api_reference/Filesystem-API.html#filesystem-api-nodefs * https://github.com/kripken/emscripten/blob/master/tests/fs/test_nodefs_rw.c

27555. **comment:** Suppress warnings about plain fopen() etc.**label:** code-design

27556. suppress warning

27557. try { FS.mkdir("/tmp"); } catch (e) { console.log("Failed to create virtual /tmp: " + e); }

27558. Should never happen, just in case.

27559. * Execute any argument file(s)

27560. **comment:** duk_safe_call() cleans up**label:** code-design

27561. * Parse options

27562. an error 'taints' the execution

27563. At the moment it's not possible to replace the default MEMFS mounted at '/': * https://github.com/kripken/emscripten/issues/2040 *

https://github.com/kripken/emscripten/blob/incoming/src/library_fs.js#L1341-L1358

27564. **comment:** This is useful at the global level; libraries should avoid SIGPIPE though**label:** code-design

27565. nargs

27566. Helper define to enable a feature set; can also use separate defines.

27567. Last component is partial, complete.

27568. **comment:** Workaround for snprintf() missing in older MSVC versions. * Note that _snprintf() may not NUL terminate the string, but * this difference does not matter here as a NUL terminator is * always explicitly added.**label:** code-design

27569. Ignore array index keys: usually not desirable, and would * also require ['0'] quoting.

27570. Scan backwards for a maximal string which looks like a property * chain (e.g. foo.bar.quux).

27571. skip filename

27572. **comment:** * Heap initialization when using AllJoyn.js pool allocator (without any * other AllJoyn.js integration). This serves as an example of how to * integrate Duktape with a pool allocator and is useful for low memory * testing. * * The pool sizes are not optimized here. The sizes are chosen so that * you can look at the high water mark (hwm) and use counts (use) and see * how much allocations are needed for each pool size. To optimize pool * sizes more accurately, you can use -alloc-logging and inspect the memory * allocation log which provides exact byte counts etc. * * https://git.allseenalliance.org/cgit/core/alljoyn-js.git *

https://git.allseenalliance.org/cgit/core/alljoyn-js.git/tree/ajs.c

label: code-design

27573. This extern declaration is provided by duktape.h, array provided by duktape.c.

27574. heapConfig
27575. Enable heap dumps
27576. duk_heap, with heap ptr compression, ROM strings+objects
27577. ...
27578. Scan ROM pointer range for faster detection of "is 'p' a ROM pointer" * later on.
27579. heapSz
27580. **comment:** not needed
 label: requirement
27581. numPools
27582. **comment:** * Simplified example of an external strings strategy where incoming strings * are written sequentially into a fixed, memory mapped flash area. ** The example first scans if the string is already in the flash (which may * happen if the same string is interned multiple times), then adds it to * flash if there is space. ** This example is too slow to be used in a real world application: there * should be e.g. a hash table to quickly check for strings that are already * present in the string data (similarly to how string interning works in * Duktape itself).
 label: code-design
27583. * Not present yet, check if we have space. Again, be careful to * ensure there is space for a NUL following the input data.
27584. Linear scan. An actual implementation would need some acceleration * structure, e.g. select a sublist based on first character. ** NOTE: input string (behind 'ptr' with 'len' bytes) DOES NOT have a * trailing NUL character. Any strings returned from this function * MUST have a trailing NUL character.
27585. * 'ajduk' specific functionality, examples for low memory techniques
27586. Pointer compression with ROM strings/objects: ** For now, use DUK_USE_ROM_OBJECTS to signal the need for compressed ROM * pointers. DUK_USE_ROM_PTRCOMP_FIRST is provided for the ROM pointer * compression range minimum to avoid duplication in user code.
27587. to avoid empty source file
27588. Ensure that we always get the heap_udata given in heap creation. * (Useful for Duktape development, not needed for user programs.)
27589. AjsHeap.dump(), allows Ecmascript code to dump heap status at suitable * points.
27590. heapNum
27591. protected call not yet running
27592. **comment:** The if-condition should be the fastest possible check * for "is 'p' in ROM?". If pointer is in ROM, we'd like * to compress it quickly. Here we just scan a ~1K array * which is very bad for performance and for illustration * only.
 label: code-design
27593. Ensure that we always get the heap_udata given in heap creation.
27594. heap
27595. * These strings are from util/duk_meta_to_strarray.py
27596. **comment:** * Check if we already have the string. Be careful to compare for * NUL terminator too, it is NOT present in 'ptr'. This algorithm * is too simplistic and way too slow for actual use.
 label: code-design
27597. duk_heap, with heap ptr compression, RAM strings+objects
27598. This is a blind lookup, could check index validity. * Duktape should never decompress a pointer which would * be out-of-bounds here.
27599. **comment:** * External strings strategy intended for valgrind testing: external strings * are allocated using malloc()/free() so that valgrind can be used to ensure * that strings are e.g. freed exactly once.
 label: code-design
27600. * These strings are manually added, and would be gathered in some * application specific manner.
27601. **comment:** It's not worth it to make very small strings external, as * they would take the same space anyway. Also avoids zero * length degenerate case.
 label: code-design
27602. **comment:** Somewhat portable way of losing a const without warnings. * Another approach is to cast through intptr_t, but that * type is not always available.
 label: code-design
27603. * Example pointer compression functions. ** 'base' is chosen so that no non-NULL pointer results in a zero result * which is reserved for NULL pointers.
27604. * There is space, add the string to our collection, being careful * to append the NUL.
27605. DUK_CMDLINE_AJSHEAP
27606. **comment:** * Wrapped ajs_heap.c alloc functions ** Used to write an alloc log.
 label: code-design
27607. until_nul
27608. We should really never be here: Duktape should only be * compressing pointers which are in the ROM compressed * pointers list, which are known at 'make dist' time. * We go on, causing a pointer compression error.
27609. * Simplified example of an external strings strategy where a set of strings * is gathered during application compile time and baked into the application * binary. ** Duktape built-in strings are available from duk_build_meta.json, see * util/duk_meta_to_strarray.py. There may also be a lot of application * specific strings, e.g. those used by application specific APIs. These * must be gathered through some other means, see e.g. util/scan_strings.py.
27610. * Execution timeout example
27611. See userdata discussion in ajsheap_enc16().
27612. duk_hthread, with heap ptr compression, ROM strings+objects
27613. **comment:** potentially unused
 label: code-design
27614. **comment:** Userdata is not needed in this case but would be useful if heap * pointer compression were used for multiple heaps. The userdata * allows the callback to distinguish between heaps and their base * pointers. ** If not needed, the userdata can be left out during compilation * by simply ignoring the userdata argument of the pointer encode * and decode macros. It is kept here so that any bugs in actually * providing the value inside Duktape are revealed during compilation.
 label: code-design
27615. numHeaps
27616. * Helpers
27617. seconds
27618. * Convert an 8-bit input string (e.g. ISO-8859-1) into CESU-8. * Calling code supplies the "code page" as a 256-entry array of * codepoints for the conversion. ** This is useful when input data is in non-UTF-8 format and must * be converted at runtime, e.g. when compiling non-UTF-8 source * code. Another alternative is to use e.g. iconv.
27619. Temporary buffer length wraps.
27620. max expansion is 1 input byte -> 3 output bytes
27621. Decode an 8-bit string using 'codepage' into Unicode codepoints and * re-encode into CESU-8. Codepage argument must point to a 256-entry * table. Only supports BMP (codepoints U+0000 to U+FFFF).
27622. [... tmp res]
27623. In CESU-8 all codepoints in [0x0000,0xFFFF] are * allowed, including surrogates.
27624. DUK_CODEPAGE_CONV_H_INCLUDED
27625. <http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP1252.TXT>
27626. Example: compile and run test source encoded in Windows codepage 1252.
27627. undefined
27628. Exercise all 3 byte lengths: any ASCII character is 1 byte, 0xFC maps to * U+00FC which is 2 bytes, and 0x80 maps to U+20AC which is 3 bytes.
27629. * Same as above, but if the exception inherits from std::exception, it's * "what()" will be included in the error message.
27630. * If you let your own C++ exceptions propagate out of a Duktape/C function * it will be caught by Duktape and considered a programming error. Duktape * will catch the exception and convert it to a Duktape error. ** This may be allowed in a later version once all the implications have been * worked out.
27631. ERROR: exception propagated to Duktape
27632. * Example class with a destructor
27633. suppress warning

27634. * Example of how to use DUK_USE_CPP_EXCEPTIONS to support automatic * variables (e.g. destructor calls) in Duktape/C functions. ** Compile with -DDUK_OPT_CPP_EXCEPTIONS: ** \$ g++ -otest -DDUK_OPT_CPP_EXCEPTIONS -I<duktape_dist>/src/ \ * <duktape_dist>/src/duktape.cpp_exceptions.cpp -lm * or ensure duk_config.h has DUK_USE_CPP_EXCEPTIONS enabled using * genconfig. When executed you should see something like: ** \$./test * my_class instance created * my_class instance destroyed <= destructor gets called * --> rc=1 (SyntaxError: parse error (line 1)) * [...] ** Duktape uses a custom exception class (duk_internal_exception) which * doesn't inherit from any base class, so that catching any base classes * in user code won't accidentally catch exceptions thrown by Duktape.

27635. * You can use C++ exceptions inside Duktape/C functions for your own * purposes but you should catch them before they propagate to Duktape.

27636. * SyntaxError caused by eval exits Duktape/C function but destructors * are executed.

27637. * Same as above, but if the exception inherits from std::exception with * a NULL what(). Duktape will describe the error as 'unknown' if so.

27638. Handshake line is available for caller for the * duration of the callback, and must not be freed * by the caller.

27639. Too small, resize.

27640. just in case, if dvalues changed something

27641. mixed endian (arm)

27642. Define to enable error prints to stderr.

27643. 0x80 ... 0xbf

27644. **comment:** IEEE double byte order, detect at run time (could also use * preprocessor defines but that's verbose to make portable). ** >>> struct.unpack('>d', '1122334455667788'.decode('hex')) * (3.841412024471731e-226,) * >>> struct.unpack('>d', '8877665544332211'.decode('hex')) * (-7.086876636573014e-268,) * >>> struct.unpack('>d', '4433221188776655'.decode('hex')) * (3.529430307187744e+20,)
label: code-design

27645. Handshake line is returned as a dvalue for convenience; it's * not actually a part of the dvalue phase of the protocol.

27646. Too big, resize so that we reclaim memory if we have just * received a large string/buffer value.

27647. little endian

27648. 0x00 ... 0x1f

27649. **comment:** * Example debug transport with a local debug message encoder/decoder. ** Provides a "received dvalue" callback for a fully parsed dvalue (user * code frees dvalue) and a "cooperate" callback for e.g. UI integration. * There are a few other callbacks. See test.c for usage examples. ** This transport implementation is not multithreaded which means that: ** - Callbacks to "received dvalue" callback come from the Duktape thread, * either during normal execution or from duk_debugger_cooperate(). ** - Calls into duk_trans_dvalue_send() must be made from the callbacks * provided (e.g. "received dvalue" or "cooperate") which use the active * Duktape thread. ** - The only exception to this is when Duktape is idle: you can then call * duk_trans_dvalue_send() from any thread (only one thread at a time). * When you next call into Duktape or call duk_debugger_cooperate(), the * queued data will be read and processed by Duktape. ** There are functions for creating and freeing values; internally they use * malloc() and free() for memory management. Duktape heap alloc functions * are not used to minimize disturbances to the Duktape heap under debugging. ** Doesn't depend on C99 types; assumes "int" is at least 32 bits, and makes * a few assumptions about format specifiers.
label: code-design

27650. never here

27651. 0x60 ... 0x7f

27652. Caller must provide a buffer at least DUK_DVALUE_TOSTRING_BUflen in size.

27653. Harmless warning on some platforms (re: range)

27654. Alloc size is len + 1 so that a NUL terminator is always * guaranteed which is convenient, e.g. you can printf() the * value safely.

27655. big endian

27656. Integers are network endian, read back into host format.

27657. Portable IEEE double byteswap. Relies on runtime detection of * host endianness.

27658. Integers are written in network endian format.

27659. IEEE doubles are network endian, read back into host format.

27660. 32 hex encoded or \xXX escaped bytes, possible "...", NUL

27661. * Dvalue transport handling

27662. Integers are network endian, read back into host format in * a portable manner.

27663. 0xc0 ... 0xff

27664. * Dvalue handling

27665. **comment:** Define to enable debug prints to stderr.
label: code-design

27666. end switch

27667. * Duktape callbacks

27668. Portable sign handling, doesn't assume 'int' is exactly 32 bits * like a direct cast would.

27669. Trial parse handshake line or dvalue(s).

27670. IEEE doubles are written in network endian format.

27671. Must cooperate until user callback provides data. From * Duktape's perspective we MUST block until data is received.

27672. mixed endian

27673. 0x20 ... 0x5f

27674. **comment:** some extra to reduce resizes
label: code-design

27675. Append data.

27676. When read_offset is large enough, "rebase" buffer by deleting already * read data and updating offsets.

27677. big endian: ok as is

27678. Convert argument dvalue into Duktape debug protocol format. * Literal constants are used here for the debug protocol, * e.g. initial byte 0x02 is REP, see doc/debugger.rst.

27679. tolerates NULL

27680. d: ieee double

27681. sending towards Duktape (duktape read callback)

27682. Buffer size needed by duk_dvalue_to_string().

27683. buf: buffer data, len: buffer length

27684. DUK_TRANS_DVALUE_H_INCLUDED

27685. struct duk_dvalue 'tag' values, note that these have nothing to do with * Duktape debug protocol initial byte. Struct fields used with the type * are noted next to the define.

27686. i: class number, buf: pointer data, len: pointer length

27687. i: 32-bit signed integer

27688. buf: pointer data, len: pointer length

27689. Sending dvalues towards Duktape.

27690. Duktape debug callbacks provided by the transport.

27691. receiving from Duktape (duktape write callback)

27692. buf: string data, len: string length

27693. Could use a union for the value but the gain would be relatively small.

27694. no fields

27695. **comment:** 0=little endian, 1=big endian, 2=mixed endian
label: code-design

27696. Initializing and freeing the transport context.

27697. Dvalue handling.

27698. i: lightfunc flags, buf: pointer data, len: pointer length

27699. Attach debugger; this will fail with a fatal error here unless * debugger support is compiled in. To fail more gracefully, call * this under a duk_safe_call() to catch the error.

27700. DumpHeap

27701. fake ptr len

27702. The Duktape handshake line is given in 'line' (without LF). * The 'line' argument can be accessed for the duration of the * callback (read only). Don't free 'line' here, the transport * handles that.

27703. medium, >= 32 chars

27704. Evaluate simple test code, callbacks will "step over" until end. ** The test code here is just for exercising the debug transport. * The 'evalMe' variable is evaluated (using debugger command Eval) * before every step to force different dvalues to be carried over * the transport.

27705. duk_trans_dvalue_send_req_cmd() sends a REQ dvalue followed by * an integer dvalue (command) for convenience.

27706. classnum

27707. Detached call forwarded as is.

27708. 0x14 = StepOver

27709. * Example program using the dvalue debug transport.

27710. long buffer, >= 65536 chars

27711. TriggerStatus

27712. suppress warning

27713. Here a normal debug client would wait for dvalues until an EOM * dvalue was received (which completes a debug message). The * debug message would then be handled, possibly causing UI changes * and/or causing debug commands to be sent to Duktape. ** The callback is responsible for eventually freeing the dvalue. * Here we free it immediately, but an actual client would probably * gather dvalues into an array or linked list to handle when the * debug message was complete.

27714. Duktape is not blocked; you can cooperate with e.g. a user * interface here and send dvalues to Duktape, but don't block.

27715. Also send a dummy TriggerStatus request with trailing dvalues * ignored by Duktape; Duktape will parse the dvalues to be able to * skip them, so that the dvalue encoding is exercised.

27716. 0x1e = Eval

27717. **comment:** First time Duktape becomes blocked, send DumpHeap which * exercises a lot of parsing code. ** NOTE: Valgrind may complain about reading uninitialized * bytes. This is caused by the DumpHeap command writing out * verbatim duk_tval values which are intentionally not * always fully initialized for performance reasons.

label: code-design

27718. short, <= 31 chars

27719. Duktape is blocked on a read and won't continue until debug * command(s) are sent. ** Normally you'd enter your own event loop here, and process * events until something needs to be sent to Duktape. For * example, the user might press a "Step over" button in the * UI which would cause dvalues to be sent. You can then * return from this callback. ** The code below sends some example messages for testing the * dvalue handling of the transport. ** If you create dvalues manually and send them using * duk_trans_dvalue_send(), you must free the dvalues after * the send call returns using duk_dvalue_free().

27720. lf_flags

27721. DUK_TRANS_SOCKET_H_INCLUDED

27722. **comment:** XXX: For now, close the listen socket because we won't accept new * connections anyway. A better implementation would allow multiple * debug attaches.

label: code-design

27723. Duktape debug transport callback: (possibly partial) read.

27724. This shouldn't happen.

27725. should never happen

27726. This TCP transport requires no write flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.

27727. * Example debug transport using a Linux/Unix TCP socket ** Provides a TCP server socket which a debug client can connect to. * After that data is just passed through.

27728. This TCP transport requires no read flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.

27729. backlog

27730. In a production quality implementation there would be a sanity * timeout here to recover from "black hole" disconnects.

27731. Duktape debug transport callback: (possibly partial) write.

27732. Write flush. If the transport combines multiple writes * before actually sending, a write flush is an indication * to write out any pending bytes: Duktape may not be doing * any more writes on this occasion.

27733. * Transport init and finish

27734. * Duktape callbacks

27735. nothing to read

27736. also returns 0, which is correct

27737. Read flush: Duktape may not be making any more read calls at this * time. If the transport maintains a receive window, it can use a * read flush as a signal to update the window status to the remote * peer. A read flush is guaranteed to occur before Duktape stops * reading for a while; it may occur in other situations as well so * it's not a 100% reliable indication.

27738. **comment:** not needed by the example

label: requirement

27739. something to read

27740. **comment:** XXX: For now, close the listen socket because we won't accept new * connections anyway. A better implementation would allow multiple * debug attaches.

label: code-design

27741. Duktape debug transport callback: (possibly partial) read.

27742. **comment:** MinGW workaround for missing getaddrinfo() etc: * <http://programmingrants.blogspot.fi/2009/09/tips-on-undefined-reference-to.html>

label: code-design

27743. This shouldn't happen.

27744. This TCP transport requires no write flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.

27745. This TCP transport requires no read flush handling so ignore. * You can also pass a NULL to duk_debugger_attach() and not * implement this callback at all.

27746. In a production quality implementation there would be a sanity * timeout here to recover from "black hole" disconnects.

27747. Duktape debug transport callback: (possibly partial) write.

27748. Write flush. If the transport combines multiple writes * before actually sending, a write flush is an indication * to write out any pending bytes: Duktape may not be doing * any more writes on this occasion.

27749. * Transport init and finish

27750. * Example debug transport using a Windows TCP socket ** Provides a TCP server socket which a debug client can connect to. * After that data is just passed through. ** [https://msdn.microsoft.com/en-us/library/windows/desktop/ms737593\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms737593(v=vs.85).aspx) ** Compiling 'duk' with debugger support using MSVC (Visual Studio): ** > cl /W3 /O2 /Feduk.exe * /DDUK_OPT_DEBUGGER_SUPPORT /DDUK_OPT_INTERRUPT_COUNTER * /DDUK_CMDLINE_DEBUGGER_SUPPORT * /lexamples\debug-trans-socket\Isrc * examples\cmdline\duk_cmdline.c * examples\debug-trans-socket\duk_trans_socket_windows.c * src\duktape.c * * With MinGW: ** \$ gcc -oduks.exe -Wall -O2 \ * -DDUK_OPT_DEBUGGER_SUPPORT -DDUK_OPT_INTERRUPT_COUNTER \ * -DDUK_CMDLINE_DEBUGGER_SUPPORT \ * -lexamples\debug-trans-socket -Isrc \ * examples\cmdline\duk_cmdline.c * examples\debug-trans-socket\duk_trans_socket_windows.c * src\duktape.c -lm -lws2_32

27751. * Duktape callbacks

27752. also returns 0, which is correct

27753. nothing to read

27754. never here

27755. Read flush: Duktape may not be making any more read calls at this * time. If the transport maintains a receive window, it can use a * read flush as a signal to update the window status to the remote * peer. A read flush is guaranteed to occur before Duktape stops * reading for a while; it may occur in other situations as well so * it's not a 100% reliable indication.

27756. **comment:** not needed by the example
label: requirement

27757. something to read

27758. Return a fixed time here as a dummy example.

27759. * Dummy Date provider * * There are two minimally required macros which you must provide in * duk_config.h: * * extern duk_double_t dummy_get_now(void); * #define DUK_USE_DATE_GET_NOW(ctx) dummy_get_now() * #define DUK_USE_DATE_GET_LOCAL_TZOFFSET(d) 0 * * Note that since the providers are macros, you don't need to use * all arguments. Similarly, you can "return" fixed values as * constants. Above, local timezone offset is always zero i.e. * we're always in UTC. * * You can also provide optional macros to parse and format timestamps * in a platform specific format. If not provided, Duktape will use * ISO 8601 only (which is often good enough).

27760. * Very simple example program for evaluating expressions from * command line

27761. nargs

27762. nrets

27763. * A few basic tests

27764. XXX: or t->target + t->delay?

27765. **comment:** Active timers. Dense list, terminates to end of list or first unused timer. * The list is sorted by 'target', with lowest 'target' (earliest expiry) last * in the list. When a timer's callback is being called, the timer is moved * to 'timer_expiring' as it needs special handling should the user callback * delete that particular timer.
label: code-design

27766. ignore errors for now -> [... stash eventTimers]

27767. not found, append to list

27768. Initialize global stash 'eventTimers'.

27769. Update timer_count.

27770. Misc

27771. -> [...]

27772. -> [... stash eventTimers retval]

27773. One-shot timer (always removed) or removed by user callback.

27774. Finally, register the callback to the global stash 'eventTimers' object.

27775. * Determine poll() timeout (as close to poll() as possible as * the wait is relative).

27776. Interval timer, not removed by user callback. Queue back to * timer list and bubble to its final sorted position.

27777. deleted, perhaps by previous callback

27778. Timer is now at the last position; use swaps to "bubble" it to its * correct sorted position.

27779. * If exit has been requested, exit without running further * callbacks.

27780. **comment:** mark to-be-deleted, cleaned up by next poll
label: code-design

27781. last timer at timer_count - 1

27782. Zero last element for clarity.

27783. Socket poll state.

27784. [... stash eventTimers]

27785. error

27786. * Compact poll list by removing pollfds with fd == 0.

27787. Set global 'EventLoop'.

27788. eventTimers[timer_id] = callback

27789. **comment:** zeroize unused entries for sanity
label: code-design

27790. Last timer expires first (list is always kept sorted).

27791. -> [... stash eventTimers func]

27792. delay/interval

27793. oneshot=1 (setTimeout), repeated=0 (setInterval)

27794. The callback associated with the timer is held in the "global stash", * in <stash>.eventTimers[String(id)]. The references must be deleted * when a timer struct is deleted.

27795. Get Javascript compatible 'now' timestamp (millisecs since 1970).

27796. events == 0 means stop listening to the FD

27797. -> [timer_id stash eventTimers]

27798. * Expire timers.

27799. Shift elements downwards to keep the timer list dense * (no need if last element).

27800. * If exit requested, bail out as fast as possible.

27801. 't' expires later than 't-1', so swap them and repeat.

27802. Timer to bubble is at index i, timer to compare to is * at i-1 (both guaranteed to exist).

27803. Return timer id.

27804. Bubble last timer on timer list backwards until it has been moved to * its proper sorted position (based on 'target' time).

27805. next target time

27806. i = input index * j = output index (initially same as i)

27807. The C state is now up-to-date, but we still need to delete * the timer callback state from the global 'stash'.

27808. 'rc' fds active

27809. clamping ensures that fits

27810. * Expired timer(s) still exist?

27811. timeout

27812. * Move the timer to 'expiring' for the duration of the callback. * Mark a one-shot timer deleted, compute a new target for an interval.

27813. * Call timer callback. The callback can operate on the timer list: * add new timers, remove timers. The callback can even remove the * expired timer whose callback we're calling. However, because the * timer being expired has been moved to 'timer_expiring', we don't * need to worry about the timer's offset changing on the timer list.

27814. **comment:** this is quite excessive for embedded use, but good for testing
label: code-design

27815. if timeout, no need to check pollfd

27816. The Ecmascript poll handler is passed through EventLoop.fdPollHandler * which c_eventloop.js sets before we come here.

27817. indexes: * 0 = function (callback) * 1 = delay * 2 = boolean: oneshot

27818. -> [func delay oneshot stash eventTimers]

27819. **comment:** * Unlike insertion, deletion needs a full scan of the timer list * and an expensive remove. If no match is found, nothing is deleted. * Caller gets a boolean return code indicating match. * * When a timer is being expired and its user callback is running, * the timer has been moved to 'timer_expiring' and its deletion * needs special handling: just mark it to-be-deleted and let the * expiry code remove it.
label: code-design

27820. nargs

27821. * Poll for activity or timeout.

27822. copy only if indices have diverged

27823. timer has been requested for removal

27824. -> [global EventLoop fdPollHandler]

27825. Because a user callback can mutate the timer list (by adding or deleting * a timer), we expire one timer and then rescan from the end again. There * is a sanity limit on how many times we do this per expiry round.

27826. keep output index the same
27827. **comment:** * Check socket activity, handle all sockets. Handling is offloaded to * Ecmascript code (fd + revents). ** If FDs are removed from the poll list while we're processing callbacks, * the entries are simply marked unused (fd set to 0) without actually * removing them from the poll list. This ensures indices are not * disturbed. The poll list is compacted before next poll().
label: code-design
27828. numeric ID (returned from e.g. setTimeout); zero if unused
27829. Should never happen, so return whatever.
27830. delete eventTimers[timer_id]
27831. 't' expires earlier than (or same time as) 't-1', so we're done.
27832. indexes: * 0 = timer id
27833. **comment:** * C eventloop example. ** Timer management is similar to eventloop.js but implemented in C. * In particular, timer insertion is an O(n) operation; in a real world * eventloop based on a heap insertion would be O(log N).
label: code-design
27834. print('unexpected revents, close fd');
27835. no size control now print('READ', Duktape.enc('jx', data));
27836. * C eventloop example (c_eventloop.c). ** Ecmascript code to initialize the exposed API (setTimeout() etc) when * using the C eventloop. ** <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Timers>
27837. This simple example doesn't have support for write blocking / draining
27838. Connected
27839. print('ACCEPT:', Duktape.enc('jx', acc_res));
27840. Special case: callback arguments are provided. [arg1, arg2, ...] [global(this), arg1, arg2, ...]
27841. print('activity on fd', fd, 'revents', revents);
27842. * Timer API
27843. **comment:** * Socket handling ** Ideally this would be implemented more in C than here for more speed * and smaller footprint: C code would directly maintain the callback state * and such. ** Also for more optimal I/O, the buffer churn caused by allocating and * freeing a lot of buffer values could be eliminated by reusing buffers. * Socket reads would then go into a pre-allocated buffer, for instance.
label: code-design
27844. retval ignored
27845. print('UNKNOWN');
27846. Legacy case: callback is a string.
27847. oneshot
27848. Normal case: callback given as a function without arguments.
27849. Math.exp(0)...Math.exp(8) is an uneven distribution between 1...~2980.
27850. Here the inserts take a lot of time because the underlying timer manager * data structure has O(n) insertion performance.
27851. * Test using timers and intervals with curses.
27852. * Pure Ecmascript eventloop example. ** Timer state handling is inefficient in this trivial example. Timers are * kept in an array sorted by their expiry time which works well for expiring * timers, but has O(n) insertion performance. A better implementation would * use a heap or some other efficient structure for managing timers so that * all operations (insert, remove, get nearest timer) have good performance. ** <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Timers>
27853. **comment:** If exit requested, don't call any more callbacks. This allows a callback to do cleanups and request exit, and can be sure that no more callbacks are processed.
label: code-design
27854. Remove the timer from the active list and process it. The user callback may add new timers which is not a problem. The callback may also delete timers which is not a problem unless the timer being deleted is the timer whose callback we're running; this is why the timer is recorded in this.expiring so that clearTimeout() and clearInterval() can detect this situation.
27855. sockets fd -> callback fd -> callback fd -> callback
27856. If the timer was one-shot, it's marked 'removed'. If the user callback requested deletion for the timer, it's also marked 'removed'. If the timer is an interval (and is not marked removed), insert it back into the timer list.
27857. this.dumpState();
27858. Timer has expired and we're processing its callback. User callback has requested timer deletion. Mark removed, so that the timer is not reinserted back into the active list. This is actually a common case because an interval may very well cancel itself.
27859. Special case: callback arguments are provided. [arg1, arg2, ...] [global(this), arg1, arg2, ...]
27860. This simple example doesn't have support for write blocking / draining
27861. no such ID, ignore
27862. print('zero read for fd ' + fd + ', closing forcibly'); ignore result
27863. print('UNKNOWN');
27864. Legacy case: callback is a string.
27865. timers active timers, sorted (nearest expiry last) set to timer being expired (needs special handling in clearTimeout/clearInterval)
27866. no size control now print('READ', Duktape.enc('jx', data));
27867. print('revents ' + t.revents + ' for fd ' + fd + ', closing forcibly'); ignore result
27868. misc
27869. print('ACCEPT:', Duktape.enc('jx', acc_res));
27870. * Create poll socket list. This is a very naive approach. * On Linux, one could use e.g. epoll() and manage socket lists * incrementally.
27871. start
27872. deleteCount
27873. * Find 'i' such that we want to insert *after* timers[i] at index i+1. * If no such timer, for-loop terminates with i-1, and we insert at -1+1=0.
27874. print('exit requested, exit');
27875. Remove timer/interval with a timer ID. The timer/interval can reside either on the active list or it may be an expired timer (this.expiring) whose user callback we're running when this function gets called.
27876. Get timer with lowest expiry time. Since the active timers list is sorted, it's always the last timer.
27877. * Wait timeout for timer closest to expiry. Since the poll * timeout is relative, get this as close to poll() as possible.
27878. print(new Date(), 'poll_set OUT:', Duktape.enc('jx', poll_set));
27879. * Do the actual poll.
27880. * Timer API ** These interface with the singleton EventLoop.
27881. print('UNKNOWN POLLOUT');
27882. Timers to expire?
27883. * Exit check (may be requested by a user callback)
27884. **comment:** Timer on active list: mark removed (not really necessary, but nice for dumping), and remove from active list.
label: code-design
27885. insert after 't', to index i+1
27886. * Event loop ** Timers are sorted by 'target' property which indicates expiry time of * the timer. The timer expiring next is last in the array, so that * removals happen at the end, and inserts for timers expiring in the * near future displace as few elements in the array as possible.
27887. print(new Date(), 'poll_set IN:', Duktape.enc('jx', poll_set));
27888. * Here we must be careful with mutations: user callback may add and * delete an arbitrary number of timers. ** Current solution is simple: check whether the timer at the end of * the list has expired. If not, we're done. If it has expired, * remove it from the active list, record it in this.expiring, and call * the user callback. If user code deletes the this.expiring timer, * there is special handling which just marks the timer deleted so * it won't get inserted back into the active list. ** This process is repeated at most maxExpiry times to ensure we don't * get stuck forever; user code could in principle add more and more * already expired timers.
27889. custom call
27890. * Process expired timers.

27891. * Process all sockets so that nothing is left unhandled for the * next round.
27892. flag for removal
27893. Timer has not expired, and no other timer could have expired either because the list is sorted.
27894. Reinsert interval timer to correct sorted position. The timer must be an interval timer because one-shot timers are marked 'removed' above.
27895. Eat errors silently. When resizing curses window an EINTR happens now.
27896. Normal case: callback given as a function without arguments.
27897. Set global 'FileIo'.
27898. * File I/O binding example.
27899. **comment:** XXX: pcall the string coercion
 label: code-design
27900. fall thru
27901. **comment:** This is useful at the global level; libraries should avoid SIGPIPE though
 label: code-design
27902. nargs
27903. nrets
27904. [... global func]
27905. Duktape/C function, safe called
27906. nret
27907. Start a zero timer which will call _USERCODE from within * the event loop.
27908. Finally, launch eventloop. This call only returns after the * eventloop terminates.
27909. signal(SIGPIPE, SIG_IGN);
27910. Compile input and place it into global _USERCODE
27911. Print error to stderr and pop error.
27912. NO_SIGNAL
27913. **comment:** XXX: print error objects specially
 label: code-design
27914. * Main for evloop command line tool. * * Runs a given script from file or stdin inside an eventloop. The * script can then access setTimeout() etc.
27915. Set global 'Ncurses'.
27916. XXX: no screen management now
27917. **comment:** * Ncurses bindings example. * * VALGRIND NOTE: when you use ncurses, there seems to be no way to get a * clean valgrind run. Even if ncurses state is properly shut down, there * will still be some residual leaks. * * Debian: install libncurses5-dev
 label: code-design
27918. [retarr]
27919. -> [... retarr key]
27920. Linux 2.6.17 and upwards, requires _GNU_SOURCE etc, not added * now because we don't use it.
27921. -> [... retarr fd]
27922. enum_flags
27923. leave enum on stack
27924. * C wrapper for poll().
27925. update revents
27926. Set global 'Poll' with functions and constants.
27927. -> [... enum key val events]
27928. -> [... enum key val]
27929. -> [... enum]
27930. -> [... retarr val "revents" fds[i].revents]
27931. [... enum key]
27932. rc = ppoll(fds, n, &ts, NULL);
27933. -> [... enum key key]
27934. -> [... retarr val]
27935. uppercase
27936. ensure we get a plain buffer
27937. Handle socket data carefully: if you convert it to a string, it may not be valid UTF-8 etc. Here we operate on the data directly in the buffer.
27938. backlog
27939. Set global 'Socket'.
27940. MSG_NOSIGNAL: avoid SIGPIPE
27941. * TCP sockets binding example.
27942. fib.js
27943. Pure Ecmascript version of low level helper
27944. Check 'val' for primality
27945. prime.js
27946. Select available helper at load time
27947. Find primes below one million ending in '9999'.
27948. primecheck.c
27949. nargs
27950. ignore result
27951. process.js
27952. nargs
27953. index
27954. processlines.c
27955. pop result/error
27956. ignore result
27957. uppercase.c
27958. We're going to need 'sz' additional entries on the stack.
27959. one return value
27960. * Very simple example program
27961. pop global
27962. #args
27963. suppress warning
27964. pop eval result
27965. nargs
27966. **comment:** * Pretty print JSON from stdin into indented JX.
 label: code-design
27967. **comment:** suppress warnings
 label: code-design
27968. nrets
27969. -1 for filename
27970. **comment:** The double value in the union is there to ensure alignment is * good for IEEE doubles too. In many 32-bit environments 4 bytes * would be sufficiently aligned and the double value is unnecessary.

label: code-design

27971. -> [... source filename]
27972. nrets
27973. * Main
27974. * Setup sandbox
27975. Handle the ptr-NULL vs. size-zero cases explicitly to minimize * platform assumptions. You can get away with much less in specific * well-behaving environments.
27976. must not return
27977. 256kB sandbox
27978. * Sandboxing example ** Uses custom memory allocation functions which keep track of total amount * of memory allocated, imposing a maximum total allocation size.
27979. -> [... filename source]
27980. * Sandbox setup and test
27981. Should be zero.
27982. Suppress warning.
27983. nargs
27984. flags
27985. Compile as program
27986. * Memory allocator which backs to standard library memory functions but * keeps a small header to track current allocation size. ** Many other sandbox allocation models are useful, e.g. preallocated pools.
27987. * Execute code from specified file
27988. * Mandelbrot example: *** \$./duk mandel.js * [...]
27989. z-> z^2 + c * -> (xx+i*yy)^2 + (x0+i*y0) * -> xx*xx+i*2*xx*yy-yy + x0 + i*y0 * -> (xx*xx - yy*yy + x0) + i*(2*xx*yy + y0)
27990. xx^2 + yy^2 >= 4.0
27991. capture in closure in case changed later
27992. * Minimal console.log() polyfill
27993. nop
27994. * Ensure Error .fileName, .lineNumber, and .stack are not directly writable, * but can be written using Object.defineProperty(). This matches Duktape * 1.3.0 and prior. ** See: <https://github.com/svaarala/duktape/pull/390>.
27995. already non-writable
27996. already writable
27997. * Ensure Error .fileName, .lineNumber, and .stack are directly writable * without having to use Object.defineProperty(). This matches Duktape * 1.4.0 behavior. * See: <https://github.com/svaarala/duktape/pull/390>.
27998. tag for unpacked duk_tval
27999. tag for packed duk_tval
28000. internal tag is fastint
28001. public type is DUK_TYPE_NUMBER
28002. * Helper to check if a number is internally represented as a fastint: * if (Duktape.isFastint(x)) { * print('fastint'); * } else { * print('not a fastint (or not a number)'); * } ** NOTE: This helper depends on the internal tag numbering (defined in * duk_tval.h) which is both version specific and depends on whether * duk_tval is packed or not.
28003. Tag number depends on duk_tval packing.
28004. null or undefined
28005. enumerable own keys
28006. * Object.assign(), described in E6 Section 19.1.2.1 ** <http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.assign>
28007. * Object.prototype.__defineGetter__ polyfill
28008. * Object.prototype.__defineSetter__ polyfill
28009. **comment:** * Performance.now() polyfill ** <http://www.w3.org/TR/hr-time/#sec-high-resolution-time> ** Dummy implementation which uses the Date built-in and has no higher * resolution. If/when Duktape has a built-in high resolution timer * interface, reimplement this.
label: code-design
28010. This should not really happen, but would indicate x64.
28011. **comment:** Minimize warnings for unused internal functions with GCC >= 3.1.1 and * Clang. Based on documentation it should suffice to have the attribute * in the declaration only, but in practice some warnings are generated unless * the attribute is also applied to the definition.
label: code-design
28012. Emscripten (provided explicitly by user), improve if possible
28013. Both MinGW and MSVC have a 64-bit type.
28014. * Platform autodetection
28015. --- TinyC ---
28016. mint clib is missing these
28017. SuperH
28018. --- x64 ---
28019. MSVC does not have sys/param.h
28020. --- Motorola 68k ---
28021. since gcc-2.5
28022. --- MIPS 64-bit ---
28023. **comment:** XXX: DUK_UNREACHABLE for msvc?
label: code-design
28024. OpenBSD
28025. NetBSD
28026. Apple OSX, iOS
28027. Cannot determine byte order; __ORDER_PDP_ENDIAN__ is related to 32-bit * integer ordering and is not relevant.
28028. Type for public API calls.
28029. **comment:** Rely as little as possible on compiler behavior for NaN comparison, * signed zero handling, etc. Currently never activated but may be needed * for broken compilers.
label: code-design
28030. --- Generic BSD ---
28031. uclibc may be missing these
28032. FreeBSD
28033. not defined by default
28034. **comment:** Windows, both 32-bit and 64-bit
label: code-design
28035. --- OpenBSD ---
28036. **comment:** * Alternative customization header ** If you want to modify the final DUK_USE_xxx flags directly (without * using the available DUK_OPT_xxx flags), define DUK_OPT_HAVE_CUSTOM_H * and tweak the final flags there.
label: code-design
28037. Cygwin
28038. integer endianness is little on purpose
28039. e.g. gettimeofday_r
28040. * Autogenerated defaults

28041. Atari Mint
28042. **comment:** GCC older than 4.6: avoid overflow warnings related to using INFINITY
 label: code-design
28043. Most portable, wastes space
28044. GCC/clang inaccurate math would break compliance and probably duk_tval, * so refuse to compile. Relax this if -ffast-math is tested to work.
28045. --- Generic ---
28046. VS2012+ has stdint.h, < VS2012 does not (but it's available for download).
28047. **comment:** * Alignment requirement and support for unaligned accesses * * Assume unaligned accesses are not supported unless specifically allowed * in the target platform. Some platforms may support unaligned accesses * but alignment to 4 or 8 may still be desirable.
 label: requirement
28048. Clang
28049. AmigaOS. Neither AMIGA nor __amigaos__ is defined on VBCC, so user must * define 'AMIGA' manually when using VBCC.
28050. * duk_config.h configuration header generated by genconfig.py. * * Git commit: 0a70d7e4c5227c84e3fed5209828973117d02849 * Git describe: v1.8.0 * Git branch: v1.8-maintenance * * Supported platforms: * - Mac OSX, iPhone, Darwin * - OpenBSD * - Generic BSD * - Atari ST TOS * - AmigaOS * - Windows * - Flashplayer (Crossbridge) * - QNX * - TI-Nspire * - Emscripten * - Linux * - Solaris * - Generic POSIX * - Cygwin * - Generic UNIX * - Generic fallback * * Supported architectures: * - x86 * - x64 * - x32 * - ARM 32-bit * - ARM 64-bit * - MIPS 32-bit * - MIPS 64-bit * - PowerPC 32-bit * - PowerPC 64-bit * - SPARC 32-bit * - SPARC 64-bit * - SuperH * - Motorola 68k * - Emscripten * - Generic * * Supported compilers: * - Clang * - GCC * - MSVC * - Emscripten * - TinyC * - VBCC * - Bruce's C compiler * - Generic *
28051. --- GCC ---
28052. Duktape/C function return value, platform int is enough for now to * represent 0, 1, or negative error code. Must be compatible with * assigning truth values (e.g. duk_ret_t rc = (foo == bar);).
28053. Low memory algorithm: separate chaining using arrays, fixed size hash
28054. __ OVERRIDE_DEFINES __
28055. <http://stackoverflow.com/questions/5919996/how-to-detect-reliably-mac-os-x-ios-linux-windows-in-c-preprocessor>
28056. vbcc + AmigaOS has C99 but no inttypes.h
28057. MSVC
28058. * Check whether or not a packed duk_tval representation is possible. * What's basically required is that pointers are 32-bit values * (sizeof(void *) == 4). Best effort check, not always accurate. * If guess goes wrong, crashes may result; self tests also verify * the guess.
28059. --- Linux ---
28060. --- AmigaOS ---
28061. BCC, assume we're on x86.
28062. Pointer size determination based on __WORDSIZE or architecture when * that's not available.
28063. Unsigned index variant.
28064. External provider already defined.
28065. Byte order varies, so rely on autodetect.
28066. DUK_USE_UNION_INITIALIZERS: required from compilers, so no fill-in.
28067. **comment:** Special naming to avoid conflict with e.g. DUK_FREE() in duk_heap.h * (which is unfortunately named). May sometimes need replacement, e.g. * some compilers don't handle zero length or NULL correctly in realloc().
 label: code-design
28068. empty
28069. --- MSVC ---
28070. We're generally assuming that we're working on a platform with a 32-bit * address space. If DUK_SIZE_MAX is a typecast value (which is necessary * if SIZE_MAX is missing), the check must be avoided because the * preprocessor can't do a comparison.
28071. 64-bit constants. Since LL / ULL constants are not always available, * use computed values. These values can't be used in preprocessor * comparisons; flag them as such.
28072. --- x86 ---
28073. --- MIPS 32-bit ---
28074. XXX: add feature options to force basic types from outside?
28075. Array index values, could be exact 32 bits. * Currently no need for signed duk_arridx_t.
28076. SIZE_MAX may be missing so use an approximate value for it.
28077. DUK_F_BCC
28078. Complex condition broken into separate parts.
28079. POSIX
28080. If not provided, use safe default for alignment.
28081. * Date provider selection * * User may define DUK_USE_DATE_GET_NOW() etc directly, in which case we'll * rely on an external provider. If this is not done, revert to previous * behavior and use Unix/Windows built-in provider.
28082. Cannot determine byte order.
28083. --- Cygwin ---
28084. !defined(DUK_USE_BYTERORDER) && defined(__BYTE_ORDER__)
28085. More or less standard endianness predefines provided by header files. * The ARM hybrid case is detected by assuming that __FLOAT_WORD_ORDER * will be big endian, see: <http://lists.mysql.com/internals/443>. * On some platforms some defines may be present with an empty value which * causes comparisons to fail: <https://github.com/svaarala/duktape/issues/453>.
28086. !defined(DUK_USE_BYTERORDER)
28087. **comment:** For custom platforms allow user to define byteorder explicitly. * Since endianness headers are not standardized, this is a useful * workaround for custom platforms for which endianness detection * is not directly supported. Perhaps custom hardware is used and * user cannot submit upstream patches.
 label: code-design
28088. **comment:** Don't know how to declare unreachable point, so don't do it; this * may cause some spurious compilation warnings (e.g. "variable used * uninitialized").
 label: code-design
28089. Good default is a bit arbitrary because alignment requirements * depend on target. See <https://github.com/svaarala/duktape/issues/102>.
28090. --- Windows ---
28091. User provided InitJS.
28092. These have been tested from VS2008 onwards; may work in older VS versions * too but not enabled by default.
28093. Error codes are represented with platform int. High bits are used * for flags and such, so 32 bits are needed.
28094. Intermediate define for 'have inttypes.h'
28095. build is not C99 or C++11, play it safe
28096. GCC: test not very accurate; enable only in relatively recent builds * because of bugs in gcc-4.4 (<http://lists.debian.org/debian-gcc/2010/04/msg00000.html>)
28097. Intel x86 (32-bit), x64 (64-bit) or x32 (64-bit but 32-bit pointers), * define only one of DUK_F_X86, DUK_F_X64, DUK_F_X32. *
 <https://sites.google.com/site/x32abi/>
28098. Enabled with debug/assertions just so that any issues can be caught.
28099. C99 or compatible
28100. --- Mac OSX, iPhone, Darwin ---
28101. C99 or above
28102. MinGW. Also GCC flags (DUK_F_GCC) are enabled now.
28103. This detection is not very reliable.
28104. Support for 48-bit signed integer duk_tval with transparent semantics.
28105. GCC and Clang provide endianness defines as built-in predefines, with * leading and trailing double underscores (e.g. __BYTE_ORDER__). See * output of "make gcprefs" and "make clangprefs". Clang doesn't * seem to provide __FLOAT_WORD_ORDER__; assume not mixed endian for clang. *
 <http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html>

28106. 64-bit type detection is a bit tricky. ** `ULLONG_MAX` is a standard define. `__LONG_LONG_MAX__` and `__ULONG_LONG_MAX__` * are used by at least GCC (even if system headers don't provide `ULLONG_MAX`). * Some GCC variants may provide `__LONG_LONG_MAX__` but not `__ULONG_LONG_MAX__`. ** `ULL / LL` constants are rejected / warned about by some compilers, even if * the compiler has a 64-bit type and the compiler/system headers provide an * unsupported constant (`ULL/LL`)! Try to avoid using `ULL / LL` constants. * As a side effect we can only check that e.g. `ULONG_MAX` is larger than 32 * bits but can't be sure it is exactly 64 bits. Self tests will catch such * cases.

28107. Convert any input pointer into a "void *", losing a const qualifier. * This is not fully portable because casting through `duk_uintptr_t` may * not work on all architectures (e.g. those with long, segmented pointers).

28108. Small integers (16 bits or more) can fall back to the 'int' type, but * have a typedef so they are marked "small" explicitly.

28109. **comment:** Most portable, potentially wastes space
label: code-design

28110. PowerPC

28111. --- Solaris ---

28112. **comment:** Technically C99 (C++11) but found in many systems. On some systems * `__STDC_LIMIT_MACROS` and `__STDC_CONSTANT_MACROS` must be defined before * including `stdint.h` (see above).
label: code-design

28113. Missing some obvious constants.

28114. No provider for `DUK_USE_DATE_PARSE_STRING()`, fall back to ISO 8601 only.

28115. no `endian.h` or `stdint.h`

28116. --- Atari ST TOS ---

28117. GCC: assume we have `_va_copy()` in non-C99 mode.

28118. User forced alignment to 4 or 8.

28119. C99 / C++11 and above: rely on `va_copy()` which is required.

28120. <http://bellard.org/tcc/tcc-doc.html#SEC9>

28121. Check that architecture is two's complement, standard C allows e.g. * `INT_MIN` to be $-2^{31}+1$ (instead of -2^{31}).

28122. Based on 'make checkalign' there are no alignment requirements on * Linux MIPS except for doubles, which need align by 4. Alignment * requirements vary based on target though.

28123. AmigaOS on M68k

28124. not configured for DLL build

28125. * You may add overriding `#define/#undef` directives below for * customization. You of course cannot un-`#include` or un-`typedef` * anything; these require direct changes above.

28126. **comment:** cannot detect 64-bit type, not always needed so don't error
label: code-design

28127. --- Generic UNIX ---

28128. std::exception

28129. A few types are assumed to always exist.

28130. Float word order not known, assume not a hybrid.

28131. --- Generic fallback ---

28132. **comment:** --- Generic POSIX ---
label: code-design

28133. no parsing (not an error)

28134. **comment:** ANSI C (various versions) and some implementations require that the * pointer arguments to `memset()`, `memcpy()`, and `memmove()` be valid values * even when byte size is 0 (even a NULL pointer is considered invalid in * this context). Zero-size operations as such are allowed, as long as their * pointer arguments point to a valid memory area. The `DUK_MEMSET()`, * `DUK_MEMCPY()`, and `DUK_MEMMOVE()` macros require this same behavior, i.e.: * (1) pointers must be valid and non-NULL, (2) zero size must otherwise be * allowed. If these are not fulfilled, a macro wrapper is needed. * * <http://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0> * <http://lists.cs.uiuc.edu/pipermail/llvmdev/2007-October/011065.html> * * Not sure what's the required behavior when a pointer points just past the * end of a buffer, which often happens in practice (e.g. zero size memmoves). * For example, if allocation size is 3, the following pointer would not * technically point to a valid memory byte: * * <-- alloc --> * | 0 | 1 | 2 | * ^ - p=3, points after last valid byte (2)
label: code-design

28135. C++11 or above

28136. SPARC byte order varies so rely on autodetection.

28137. `DUK_F_PACKED_TVAL_PROVIDED`

28138. <http://www.monkey.org/openbsd/archive/ports/0401/msg00089.html>

28139. VBCC

28140. `sigsetjmp()` alternative

28141. GCC. Clang also defines `__GNUC__` so don't detect GCC if using Clang.

28142. --- SuperH ---

28143. Pre-C99: `va_list` type is implementation dependent. This replacement * assumes it is a plain value so that a simple assignment will work. * This is not the case on all platforms (it may be a single-array element, * for instance).

28144. **comment:** The best type for an "all around int" in Duktape internals is "at least * 32 bit signed integer" which is most convenient. Same for unsigned type. * Prefer 'int' when large enough, as it is almost always a convenient type.
label: code-design

28145. uclibc

28146. Explicit marker needed; may be 'defined', 'undefined', 'or 'not provided'.

28147. No provider for `DUK_USE_DATE_FORMAT_STRING()`, fall back to ISO 8601 only.

28148. **comment:** Many platforms are missing `fpclassify()` and friends, so use replacements * if necessary. The replacement constants (FP_NAN etc) can be anything but * match Linux constants now.
label: code-design

28149. * Checks for config option consistency (`DUK_USE_xxx`)

28150. autodetect compiler

28151. Rely on C89 headers only; `time.h` must be here.

28152. These functions don't currently need replacement but are wrapped for * completeness. Because these are used as function pointers, they need * to be defined as concrete C functions (not macros).

28153. --- TI-Nspire ---

28154. **comment:** We need `va_copy()` which is defined in C99 / C++11, so an awkward * replacement is needed for pre-C99 / pre-C++11 environments. This * will quite likely need portability hacks for some non-C99 * environments.
label: code-design

28155. AmigaOS on M68K or PPC is always big endian.

28156. Feature option forcing.

28157. --- ARM 64-bit ---

28158. * Feature option handling

28159. **comment:** * Wrapper typedefs and constants for integer types, also sanity check types. * * C99 typedefs are quite good but not always available, and we want to avoid * forcibly redefining the C99 typedefs. So, there are Duktape wrappers for * all C99 typedefs and Duktape code should only use these typedefs. Type * detection when C99 is not supported is best effort and may end up detecting * some types incorrectly. * * Pointer sizes are a portability problem: pointers to different types may * have a different size and function pointers are very difficult to manage * portably. * * http://en.wikipedia.org/wiki/C_data_types#Fixed-width_integer_types * * Note: there's an interesting corner case when trying to define minimum * signed integer value constants which leads to the current workaround of * defining e.g. `-0x80000000` as `(-0x7fffffffL - 1L)`. See doc/code-issues.txt * for a longer discussion. * * Note: avoid typecasts and computations in

macro integer constants as they * can then no longer be used in macro relational expressions (such as * #if DUK_SIZE_MAX < 0xffffffffUL). There is internal code which relies on * being able to compare DUK_SIZE_MAX against a limit.

label: code-design

28160. Not standard but common enough

28161. --- QNX ---

28162. **comment:** Object property allocation layout has implications for memory and code * footprint and generated code size/speed. The best layout also depends * on whether the platform has alignment requirements or benefits from * having mostly aligned accesses.

label: code-design

28163. `(v)snprintf()` is missing before MSVC 2015. Note that `(v)snprintf()` does * NOT NUL terminate on truncation, but Duktape code never assumes that. *

<http://stackoverflow.com/questions/2915672/snprintf-and-visual-studio-2010>

28164. Motorola 68K. Not defined by VBCC, so user must define one of these * manually when using VBCC.

28165. [http://msdn.microsoft.com/en-us/library/aa235362\(VS.60\).aspx](http://msdn.microsoft.com/en-us/library/aa235362(VS.60).aspx)

28166. Non-C99 case, still relying on DUK_UINTPTR_MAX, as long as it is not a computed value

28167. <http://bellard.org/tcc/tcc-doc.html#SEC7>

28168. * Intermediate helper defines

28169. **comment:** Macro hackery to convert e.g. __LINE__ to a string without formatting, * see: <http://stackoverflow.com/questions/240353/convert-a-preprocessor-token-to-a-string>

label: code-design

28170. Flash player (e.g. Crossbridge)

28171. `no endian.h`

28172. Shared includes: C89

28173. integer byte order

28174. float word order

28175. * Convert DUK_USE_BYTEORDER, from whatever source, into currently used * internal defines. If detection failed, #error out.

28176. e.g. `ptrdiff_t`

28177. **comment:** XXX: DUK_NOINLINE, DUK_INLINE, DUK_ALWAYS_INLINE for msvc?

label: code-design

28178. SPARC

28179. VBCC supports C99 so check only for C99 for union initializer support. * Designated union initializers would possibly work even without a C99 check.

28180. C++11 apparently ratified stdint.h

28181. QNX gcc cross compiler seems to define e.g. __LITTLEENDIAN__ or __BIGENDIAN__: * \$ /opt/qnx650/host/linux/x86/usr/bin/i486-pc-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 67:#define __LITTLEENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/mips-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 70:#define __BIGENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/arm-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian *

70:#define __LITTLEENDIAN__ 1

28182. At least some uclibc versions have broken floating point math. For * example, `fpclassify()` can incorrectly classify certain NaN formats. * To be safe, use replacements.

28183. Placeholder fix for (detection is wider than necessary): * http://llvm.org/bugs/show_bug.cgi?id=17788

28184. `snprintf()` is technically not part of C89 but usually available.

28185. --- x32 ---

28186. Basic integer typedefs and limits, preferably from inttypes.h, otherwise * through automatic detection.

28187. **comment:** Only include when compiling Duktape to avoid polluting application build * with a lot of unnecessary defines.

label: code-design

28188. --- VBCC ---

28189. On some systems SIZE_MAX can be smaller than max unsigned 32-bit value * which seems incorrect if size_t is (at least) an unsigned 32-bit type. * However, it doesn't seem useful to error out compilation if this is the * case.

28190. **comment:** On other platforms use layout 2, which requires some padding but * is a bit more natural than layout 3 in ordering the entries. Layout * 3 is currently not used.

label: code-design

28191. VS2005+ should have variadic macros even when they're not C99.

28192. illumos / Solaris

28193. DUK_COMPILING_DUKTAPE

28194. DUK_OPT_FORCE_BYTEORDER

28195. QNX

28196. since gcc-4.5

28197. **comment:** Windows 32-bit and 64-bit are currently the same.

label: code-design

28198. This is optionally used by panic handling to cause the program to segfault * (instead of e.g. `abort()`) on panic. Valgrind will then indicate the C * call stack leading to the panic.

28199. IEEE float/double typedef.

28200. --- ARM 32-bit ---

28201. **comment:** XXX: This is technically not guaranteed because it's possible to configure * an x86 to require aligned accesses with Alignment Check (AC) flag.

label: code-design

28202. TinyC

28203. **comment:** Note: the funny looking computations for signed minimum 16-bit, 32-bit, and * 64-bit values are intentional as the obvious forms (e.g. -0x80000000L) are * -not- portable. See code-issues.txt for a detailed discussion.

label: code-design

28204. These are necessary wild guesses.

28205. Already provided above

28206. Atari ST TOS. __TOS__ defined by PureC. No platform define in VBCC * apparently, so to use with VBCC user must define __TOS__ manually.

28207. Byte order is big endian but cannot determine IEEE double word order.

28208. --- Flashplayer (Crossbridge) ---

28209. MIPS byte order varies so rely on autodetection.

28210. stdint.h not available

28211. **comment:** The most portable way to figure out local time offset is `gmtime()`, * but it's not thread safe so use with caution.

label: requirement

28212. Codepoint type. Must be 32 bits or more because it is used also for * internal codepoints. The type is signed because negative codepoints * are used as internal markers (e.g. to mark EOF or missing argument). * (X)UTF-8/CESU-8 encode/decode take and return an unsigned variant to * ensure duk_uint32_t casts back and forth nicely. Almost everything * else uses the signed one.

28213. DUK_USE_VARIADIC_MACROS: required from compilers, so no fill-in.

28214. TOS on M68K is always big endian.

28215. don't use `strftime()` for now

28216. **comment:** not sure, not needed with C99 anyway

label: requirement

28217. C++ doesn't have standard designated union initializers ({ .foo = 1 }).

28218. --- Emscripten ---

28219. nop

28220. DLL build detection

28221. **comment:** Avoid custom date parsing and formatting for portability.

label: code-design28222. Workaround for older C++ compilers before including <inttypes.h>, * see e.g.: https://sourceware.org/bugzilla/show_bug.cgi?id=15366

28223. * Compiler autodetection

28224. --- SPARC 32-bit ---

28225. --- Clang ---

28226. On some platforms int is 16-bit but long is 32-bit (e.g. PureC)

28227. **comment:** * Byte order and double memory layout detection * * Endianness detection is a major portability hassle because the macros * and headers are not standardized. There's even variance across UNIX * platforms. Even with "standard" headers, details like underscore count * varies between platforms, e.g. both __BYTE_ORDER and _BYTE_ORDER are used * (Crossbridge has a single underscore, for instance). * * The checks below are structured with this in mind: several approaches are * used, and at the end we check if any of them worked. This allows generic * approaches to be tried first, and platform/compiler specific hacks tried * last. As a last resort, the user can force a specific endianness, as it's * not likely that automatic detection will work on the most exotic platforms. * * Duktape supports little and big endian machines. There's also support * for a hybrid used by some ARM machines where integers are little endian * but IEEE double values use a mixed order (12345678 -> 43218765). This * byte order for doubles is referred to as "mixed endian".

label: code-design28228. **comment:** Check whether we should use 64-bit integers or not. * * Quite incomplete now. Use 64-bit types if detected (C99 or other detection) * unless they are known to be unreliable. For instance, 64-bit types are * available on VBCC but seem to misbehave.**label:** code-design28229. VS2013+ supports union initializers but there's a bug involving union-inside-struct: * <https://connect.microsoft.com/VisualStudio/feedback/details/805981> * The bug was fixed (at least) in VS2015 so check for VS2015 for now: * <https://blogs.msdn.microsoft.com/vcblog/2015/07/01/c-compiler-front-end-fixes-in-vs2015/> * Manually tested using VS2013, CL reports 18.00.31101, so enable for VS2013 too.

28230. no user declarations

28231. DUK_CONFIG_H_INCLUDED

28232. BCC (Bruce's C compiler): this is a "torture target" for compilation

28233. Generic Unix (includes Cygwin)

28234. There was a curious bug where test-bi-date-canceling.js would fail e.g. * on 64-bit Ubuntu, gcc-4.8.1, -m32, and no -std=c99. Some date computations * using doubles would be optimized which then broke some corner case tests. * The problem goes away by adding 'volatile' to the datetime computations. * Not sure what the actual triggering conditions are, but using this on * non-C99 systems solves the known issues and has relatively little cost * on other platforms.

28235. C99 / C++11 and above: rely on va_copy() which is required. * Omit parenthesis on macro right side on purpose to minimize differences * to direct use.

28236. C99 types

28237. DUK_SIZE_MAX (= SIZE_MAX) is often reliable

28238. --- PowerPC 64-bit ---

28239. See: /opt/qnx650/target/qnx6/usr/include/sys/platform.h

28240. --- SPARC 64-bit ---

28241. On platforms without any alignment issues, layout 1 is preferable * because it compiles to slightly less code and provides direct access * to property keys.

28242. Strict C99 case: DUK_UINTPTR_MAX (= UINTPTR_MAX) should be very reliable

28243. autodetect platform

28244. **comment:** NetBSD 6.0 x86 (at least) has a few problems with pow() semantics, * see test-bug-netbsd-math-pow.js. Use NetBSD specific workaround. * (This might be a wider problem; if so, generalize the define name.)**label:** code-design

28245. autodetect architecture

28246. In VBCC (0.0 / 0.0) results in a warning and 0.0 instead of NaN. * In MSVC (VS2010 Express) (0.0 / 0.0) results in a compile error. * Use a computed NaN (initialized when a heap is created at the * latest).

28247. Clang: assume we have __va_copy() in non-C99 mode.

28248. --- Bruce's C compiler ---

28249. Same as 'duk_int_t' but guaranteed to be a 'fast' variant if this * distinction matters for the CPU. These types are used mainly in the * executor where it might really matter.

28250. vbcc + AmigaOS may be missing these

28251. C99 or C++11, no known issues

28252. **comment:** XXX: DUK_LIKELY, DUK_UNLIKELY for msvc?**label:** code-design

28253. Initial fix: disable secure CRT related warnings when compiling Duktape * itself (must be defined before including Windows headers). Don't define * for user code including duktape.h.

28254. varargs

28255. * Fill-ins for platform, architecture, and compiler

28256. e.g. strftime

28257. **comment:** Note: PRS and FMT are intentionally left undefined for now. This means * there is no platform specific date parsing/formatting but there is still * the ISO 8601 standard format.**label:** code-design

28258. vsnprintf() is technically not part of C89 but usually available.

28259. Rely on autodetection for byte order, alignment, and packed tval.

28260. AmigaOS + M68K seems to have math issues even when using GCC cross * compilation. Use replacements for all AmigaOS versions on M68K * regardless of compiler.

28261. Byte order is little endian but cannot determine IEEE double word order.

28262. C++

28263. MIPS. Related defines: __MIPSEB__, __MIPSEL__, __mips_isa_rev, __LP64__

28264. VBCC is missing the built-ins even in C99 mode (perhaps a header issue).

28265. __MSC_VER

28266. **comment:** Macro for suppressing warnings for potentially unreferenced variables. * The variables can be actually unreferenced or unreferenced in some * specific cases only; for instance, if a variable is only debug printed, * it is unreferenced when debug printing is disabled.**label:** code-design

28267. Byte order varies, rely on autodetection.

28268. Use __setjmp() on Apple by default, see GH-55.

28269. TI-Nspire (using Ndless)

28270. --- PowerPC 32-bit ---

28271. Some math functions are C99 only. This is also an issue with some * embedded environments using uclibc where uclibc has been configured * not to provide some functions. For now, use replacements whenever * using uclibc.

28272. Most portable

28273. Boolean values are represented with the platform 'int'.

28274. Shared includes: stdint.h is C99

28275. BSD variant

28276. Old uclibcs have a broken memcpy so use memmove instead (this is overly wide * now on purpose): <http://lists.uclibc.org/pipermail/uclibc-cvs/2008-October/025511.html>

28277. date.h is omitted, and included per platform

28278. * Architecture autodetection

28279. The most portable current time provider is time(), but it only has a * one second resolution.

28280. Convenience, e.g. gcc 4.5.1 == 40501; <http://stackoverflow.com/questions/6031819/emulating-gccs-built-in-unreachable>

28281. defined(DUK_USE_BYTEORDER)

28282. In VBCC (1.0 / 0.0) results in a warning and 0.0 instead of infinity. * Use a computed infinity (initialized when a heap is created at the * latest).

28283. For now, hash part is dropped if and only if 16-bit object fields are used.

28284. MSVC dllexport/dllimport: appropriate `__declspec` depends on whether we're * compiling Duktape or the application.

28285. **comment:** On Windows, assume we're little endian. Even Itanium which has a * configurable endianness runs little endian in Windows.
label: code-design

28286. **comment:** Compiler specific hackery needed to force struct size to match alignment, * see e.g. `duk_hbuffer.h`. * * <http://stackoverflow.com/questions/11130109/c-struct-size-alignment> * http://stackoverflow.com/questions/10951039/specifying-64-bit-alignment
label: code-design

28287. Index values must have at least 32-bit signed range.

28288. **comment:** Workaround for GH-323: avoid inlining control when compiling from * multiple sources, as it causes compiler portability trouble.
label: code-design

28289. byte order

28290. Linux

28291. Based on 'make checkalign' there are no alignment requirements on * Linux SH4, but align by 4 is probably a good basic default.

28292. MSVC preprocessor defines: <http://msdn.microsoft.com/en-us/library/b0084kay.aspx> * `_MSC_FULL_VER` includes the build number, but it has at least two formats, see e.g. * `BOOST_MSVC_FULL_VER` in http://www.boost.org/doc/libs/1_52_0/boost/config/compiler/visualc.hpp

28293. ARM

28294. Fast variants of small integers, again for really fast paths like the * executor.

28295. **comment:** When C99 types are not available, we use heuristic detection to get * the basic 8, 16, 32, and (possibly) 64 bit types. The fast/least * types are then assumed to be exactly the same for now: these could * be improved per platform but C99 types are very often now available. * 64-bit types are not available on all platforms; this is OK at least * on 32-bit platforms. * * This detection code is necessarily a bit hacky and can provide `typedefs` * and defines that won't work correctly on some exotic platform.
label: code-design

28296. **comment:** * Cleanups (all statement parsing flows through here). * * Pop label site and reset labels. Reset 'next temp' to value at * entry to reuse temps.
label: code-design

28297. **comment:** * Reference counting helper macros. The macros take a thread argument * and must thus always be executed in a specific thread context. The * thread argument is needed for features like finalization. Currently * it is not required for INCREF, but it is included just in case. * * Note that 'raw' macros such as `DUK_HEAPHDR_GET_REFCOUNT()` are not * defined without `DUK_USE_REFERENCE_COUNTING`, so caller must `#ifdef` * around them.
label: code-design

28298. alloc external with size zero

28299. 'i' is the first entry we'll keep

28300. If blen <= 0xffffUL, clen is also guaranteed to be <= 0xffffUL.

28301. Set catcher regs: `idx_base+0 = value, idx_base+1 = lj_type`.

28302. resend state next time executor is about to run

28303. **comment:** XXX: avoid this check somehow
label: code-design

28304. Should match `Function.prototype.toString()`

28305. * Main struct

28306. '/'

28307. * Unwind debugger state. If we unwind while stepping * (either step over or step into), pause execution.

28308. `Duktape.modLoaded[]` module cache

28309. [obj handler trap]

28310. Not safe to use 'reg_varbind' as assignment expression * value, so go through a temp.

28311. **comment:** size for formatting buffers
label: code-design

28312. default case exists: go there if no case matches

28313. registers are mutable, non-deletable

28314. '{'

28315. If the debugger is active we need to force an interrupt so that * debugger breakpoints are rechecked. This is important for function * calls caused by side effects (e.g. when doing a `DUK_OP_GETPROP`), see * GH-303. Only needed for success path, error path always causes a * breakpoint recheck in the executor. It would be enough to set this * only when returning to an Ecmascript activation, but setting the flag * on every return should have no ill effect.

28316. get or set a range of flags; m=first bit number, n=number of bits

28317. [level obj func pc line]

28318. * Ecmascript compliant [[GetProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If `DUK_GETDESC_FLAG_PUSH_VALUE` is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with `DUK_GETDESC_FLAG_PUSH_VALUE` * set) * - Returns zero * * May cause arbitrary side effects and invalidate (most) `duk_tval` * pointers.

28319. [arg1 ... argN this]

28320. * Matching order: * * Punctuator first chars, also covers comments, regexps * LineTerminator * Identifier or reserved word, also covers null/true/false literals * NumericLiteral * StringLiteral * EOF * * The order does not matter as long as the longest match is * always correctly identified. There are order dependencies * in the clauses, so it's not trivial to convert to a switch.

28321. Allocator functions.

28322. next does not exist or next is not a letter

28323. * Create and push an error object onto the top of stack. * If a "double error" occurs, use a fixed error instance * to avoid further trouble.

28324. 'enum'

28325. `SameValue(NaN, NaN) = true`, regardless of NaN sign or extra bits

28326. just making sure

28327. needed for inf: causes mantissa to become zero, * and rounding to be skipped.

28328. String table assert check omitted from 1.x branch * backport.

28329. **comment:** XXX: string not shared because it is conditional
label: code-design

28330. carry

28331. Initial bytes for markers.

28332. queue back to heap_allocated

28333. XXX: We can't resize the value stack to a size smaller than the * current top, so the order of the resize and adjusting the stack * top depends on the current vs. final size of the value stack. * The operations could be combined to avoid this, but the proper * fix is to only grow the value stack on a function call, and only * shrink it (without throwing if the shrink fails) on function * return.

28334. print provider

28335. form: { DecimalDigits }, val1 = min count

28336. * Figure out the final, non-bound constructor, to get "prototype" * property.

28337. **comment:** * Note: * * - `duk_match_regexp()` is required not to `longjmp()` in ordinary "non-match" * conditions; a `longjmp()` will terminate the entire matching process. * * - Clearing `saved[]` is not necessary because backtracking does it * * - Backtracking also rewinds `ctx.recursion` back to zero, unless an * internal/limit error occurs (which causes a `longjmp()`) * * - If we supported anchored matches, we would break out here * unconditionally; however, Ecmascript regexps don't have anchored * matches. It might make sense to implement a fast bail-out if * the regexp begins with '^' and `sp` is not 0: currently we'll just * run through the entire input string, trivially failing the match * at every non-zero offset.
label: code-design

28338. `DUK_TOK_LPAREN`

28339. caller ensures; rom objects are never bufferobjects now

28340. Would be nice to bulk clear the allocation, but the context * is 1-2 kilobytes and nothing should rely on it being zeroed.

28341. 1 -> needs a PUTVAR
28342. * Post-increment/decrement will return the original value as its * result value. However, even that value will be coerced using * ToNumber() which is quite awkward. Specific bytecode opcodes * are used to handle these semantics. ** Note that post increment/decrement has a "no LineTerminator here" * restriction. This is handled by duk__expr_lbp(), which forcibly terminates * the previous expression if a LineTerminator occurs before '++'/'--'.
28343. leave out trailing 'unused' elements
28344. **comment:** TODO: implement Proxy object support here
 label: requirement
28345. * Multiply + add + carry for 32-bit components using only $16 \times 16 \rightarrow 32$ * multiplies and carry detection based on unsigned overflow. ** 1st mult, 32-bit: $(A \times 2^{16} + B) \times 2^{16}$ * 2nd mult, 32-bit: $(C \times 2^{16} + D) \times 2^{16}$ * 3rd add, 32-bit: $E \times 4^{\text{th}}$ add, 32-bit: $F \times (AC \times 2^{16} + B) \times (C \times 2^{16} + D) + E + F = AC \times 2^{32} + AD \times 2^{16} + BC \times 2^{16} + BD + E + F = AC \times 2^{32} + (AD + BC) \times 2^{16} + (BD + E + F) \times 2^{16} = AC \times 2^{32} + AD \times 2^{16} + BC \times 2^{16} + (BD + E + F)$
28346. args start at index 2
28347. * Various Unicode help functions for character classification predicates, * case conversion, decoding, etc.
28348. eat colon
28349. always set; points to a reserved valstack slot
28350. Track number of escapes: necessary for proper keyword * detection.
28351. * Node.js Buffer.prototype.write(string, [offset], [length], [encoding])
28352. IdentifierPart production with IdentifierStart, ASCII, and non-BMP excluded
28353. should never happen but default here
28354. 'env'
28355. Preliminaries, required by setjmp() handler. Must be careful not * to throw an unintended error here.
28356. -> [val]
28357. emitted
28358. resume delete to target
28359. -> [... obj]
28360. success
28361. +0.0
28362. replacer is a mutation risk
28363. parse key and value
28364. **comment:** * Arithmetic operations other than '+' have number-only semantics * and are implemented here. The separate switch-case here means a * "double dispatch" of the arithmetic opcode, but saves code space. ** E5 Sections 11.5, 11.5.1, 11.5.2, 11.5.3, 11.6, 11.6.1, 11.6.2, 11.6.
 label: code-design
28365. No duk_bw_remove_ensure_slice(), functionality would be identical.
28366. if boolean matches A, skip next inst
28367. $(\geq e 0) \text{ AND } (\neq f (\text{expt } b (- p 1))) \times \text{be} < - (\text{expt } b e) == b^e \times \text{be} 1 < - (\text{be } b) == (\text{expt } b (+ e 1)) == b^{(e+1)} \times r < - (\text{f be} 1 2) == 2 \times f \times b^{(e+1)}$ [if $b == 2 \rightarrow f * b^{(e+2)}$] * $s < - (\text{f } 2)$ [if $b == 2 \rightarrow 4$] * $m^+ < - \text{be} 1 == b^{(e+1)} \times m^- < - \text{be} == b^e \times k < - 0 \times B < - B \times \text{low_ok} < - \text{round} * \text{high_ok} < - \text{round}$
28368. Perform fixed-format rounding.
28369. If not within Ecmascript range, some integer time calculations * won't work correctly (and some asserts will fail), so bail out * if so. This fixes test-bug-date-insane-setyear.js. There is * a +/- 24h leeway in this range check to avoid a test262 corner * case documented in test-bug-date-timeval-edges.js.
28370. Failed to match the quantifier, restore lexer and parse * opening brace as a literal.
28371. [...] val]
28372. 'true'
28373. **comment:** If we don't have a jmpbuf_ptr, there is little we can do * except panic. The caller's expectation is that we never * return. ** With C++ exceptions we now just propagate an uncaught error * instead of invoking the fatal error handler. Because there's * a dummy jmpbuf for C++ exceptions now, this could be changed.
 label: code-design
28374. For now shared handler is fine.
28375. may be less, since DELETED entries are NULLed by rehash
28376. Handle single character fills as memset() even when * the fill data comes from a one-char argument.
28377. OK
28378. **comment:** * Bytecode dump/load ** The bytecode load primitive is more important performance-wise than the * dump primitive. ** Unlike most Duktape API calls, bytecode dump/load is not guaranteed to be * memory safe for invalid arguments - caller beware! There's little point * in trying to achieve memory safety unless bytecode instructions are also * validated which is not easy to do with indirect register references etc.
 label: code-design
28379. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
28380. maximum loopcount for peephole optimization
28381. cleared before entering finally
28382. flags for duk_hobject_get_own_propdesc() and variants
28383. **comment:** Execute finalizers before freeing the heap, even for reachable * objects, and regardless of whether or not mark-and-sweep is * enabled. This gives finalizers the chance to free any native * resources like file handles, allocations made outside Duktape, * etc. This is quite tricky to get right, so that all finalizer * guarantees are honored. ** XXX: this perhaps requires an execution time limit.
 label: code-design
28384. useful for patching jumps later
28385. avoid attempts to add/remove object keys
28386. The get/set pointers could be 16-bit pointer compressed but it * would make no difference on 32-bit platforms because duk_tval is * 8 bytes or more anyway.
28387. * Local defines
28388. **comment:** XXX: compression
 label: code-design
28389. already NULLed (by unwind)
28390. Don't allow actual chars after equal sign.
28391. -> [Object defineProperty undefined obj key desc]
28392. whether to use macros or helper function depends on call count
28393. DUK_USE_FASTINT
28394. 7: toISOString
28395. **comment:** XXX: "read only object" ?
 label: code-design
28396. no action
28397. **comment:** XXX: There is currently no support for writing buffer object * indexed elements here. Attempt to do so will succeed and * write a concrete property into the buffer object. This should * be fixed at some point but because buffers are a custom feature * anyway, this is relatively unimportant.
 label: code-design
28398. must hold DUK_VARARGS
28399. capture is 'undefined', always matches!
28400. **comment:** * XXX: could make this a lot faster if we create the double memory * representation directly. Feasible easily (must be uniform random).
 label: code-design
28401. 'raw'
28402. DUK_TOK_RPAREN
28403. request to create catch binding
28404. for object-bound identifiers
28405. **comment:** XXX: compression (as an option)

label: code-design

28406. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** This handling is now identical for C and Ecmascript functions. * C functions always have the 'NEWENV' flag set, so their * environment record initialization is delayed (which is good). ** Delayed creation (on demand) is handled in duk_js_var.c.

28407. **comment:** Declaration binding instantiation conceptually happens when calling a * function; for us it essentially means that function prologue. The * conceptual process is described in E5 Section 10.5. ** We need to keep track of all encountered identifiers to (1) create an * identifier-to-register map ("varmap"); and (2) detect duplicate * declarations. Identifiers which are not bound to registers still need * to be tracked for detecting duplicates. Currently such identifiers * are put into the varmap with a 'null' value, which is later cleaned up. ** To support functions with a large number of variable and function * declarations, registers are not allocated beyond a certain limit; * after that limit, variables and functions need slow path access. * Arguments are currently always register bound, which imposes a hard * (and relatively small) argument count limit. ** Some bindings in E5 are not configurable (=deletable) and almost all * are mutable (writable). Exceptions are: ** - The 'arguments' binding, established only if no shadowing argument * or function declaration exists. We handle 'arguments' creation * and binding through an explicit slow path environment record. ** - The "name" binding for a named function expression. This is also * handled through an explicit slow path environment record.

label: code-design

28408. first coerce to a plain value

28409. -> [... func varmap enum]

28410. * Replace lexical environment for global scope ** Create a new object environment for the global lexical scope. * We can't just reset the _Target property of the current one, * because the lexical scope is shared by other threads with the * same (initial) built-ins.

28411. force_global

28412. NB: use 's' as temp on purpose

28413. Not odd, or y == -Infinity

28414. * Global object built-ins

28415. A bunch of helpers (for size optimization) that combine duk__expr()/duk__exprtop() * and result conversions. ** Each helper needs at least 2-3 calls to make it worth while to wrap.

28416. Value stack: these are expressed as pointers for faster stack manipulation. * [valstack, valstack_top[is GC-reachable, [valstack_top, valstack_end[is * not GC-reachable but kept initialized as 'undefined'.

28417. Output a specified number of digits instead of using the shortest * form. Used for toPrecision() andtoFixed().

28418. [body formals], formals is comma separated list that needs to be parsed

28419. bytecode execution

28420. thisArg

28421. * Not found

28422. XXX: duk_uarridx_t?

28423. E5 Section 9.4, ToInteger()

28424. force_new

28425. duk_js_call.c is required to restore the stack reserve * so we only need to reset the top.

28426. * __FILE__ / __LINE__ entry, here 'pc' is line number directly. * Sometimes __FILE__ / __LINE__ is reported as the source for * the error (fileName, lineNumber), sometimes not.

28427. avoid dereference after potential callstack realloc

28428. * Pass 2

28429. compensate for eval() call

28430. set values into ret array

28431. * Pick an object from the head (don't remove yet).

28432. fast path

28433. **comment:** Fast exit if indices are identical. This is valid for a non-existent property, * for an undefined value, and almost always for ToString() coerced comparison of * arbitrary values (corner cases where this is not the case include e.g. a an * object with varying ToString() coercion). ** The specification does not prohibit "caching" of values read from the array, so * assuming equality for comparing an index with itself falls into the category of * "caching". ** Also, compareFn may be inconsistent, so skipping a call to compareFn here may * have an effect on the final result. The specification does not require any * specific behavior for inconsistent compare functions, so again, this fast path * is OK.

label: code-design

28434. **comment:** XXX: this could be a DUK__CONSTP instead

label: code-design

28435. [... this tracedata sep this]

28436. * After arguments, allocate special registers (like shuffling temps)

28437. [ToObject(this) item1 ... itemN arr item(i)]

28438. The implementation for computing of start_pos and end_pos differs * from the standard algorithm, but is intended to result in the exactly * same behavior. This is not always obvious.

28439. This behavior mostly mimics Node.js now.

28440. **comment:** XXX: typing (duk_hcompiledfunction has duk_uint32_t)

label: code-design

28441. **comment:** XXX: because we're dealing with 'own' properties of a fresh array, * the array initializer should just ensure that the array has a large * enough array part and write the values directly into array part, * and finally set 'length' manually in the end (as already happens now).

label: code-design

28442. This may happen even after the fast path check, if exponent is * not balanced (e.g. "0e1"). Remember to respect zero sign.

28443. Encode a fastint from duk_tval ptr, no value stack effects.

28444. Already declared, update value.

28445. 'has'

28446. DUK_TOK_COMMA

28447. 'void'

28448. bufwriter for code

28449. * Abandon array failed, need to decref keys already inserted * into the beginning of new_e_k before unwinding valstack.

28450. * UTF-8 / XUTF-8 / CESU-8 constants

28451. when going backwards, we decrement cpos 'early'; * 'p' may point to a continuation byte of the char * at offset 'cpos', but that's OK because we'll * backtrack all the way to the initial byte.

28452. DUK_TOK_IDENTIFIER

28453. * Regexp recursive matching function. ** Returns 'sp' on successful match (points to character after last matched one), * NULL otherwise. ** The C recursion depth limit check is only performed in this function, this * suffices because the function is present in all true recursion required by * regexp execution.

28454. **comment:** Use recursion_limit to ensure we don't overwrite js_ctx->visiting[] * array so we don't need two counter checks in the fast path. The * slow path has a much larger recursion limit which we'll use if * necessary.

label: code-design

28455. [... func varmap enum key value this]

28456. 1: toDateString

28457. **comment:** Fill offset handling is more lenient than in Node.js.

label: code-design

28458. Positive if local time ahead of UTC.

28459. With this check in place fast paths won't need read-only * object checks. This is technically incorrect if there are * setters that cause no writes to ROM objects, but current * built-ins don't have such setters.

28460. save stack top

28461. Insert an empty jump in the middle of code emitted earlier. This is * currently needed for compiling for-in.

28462. Treat like a debugger statement: ignore when not attached.
28463. 'import'
28464. Duktape/C API guaranteed entries (on top of args)
28465. avoid warning (unsigned)
28466. Helper which can be called both directly and with duk_safe_call().
28467. [... template]
28468. -> [buffer]
28469. **comment:** XXX: Here a "slice copy" would be useful.
 label: code-design
28470. Must ensure result is 64-bit (no overflow); a * simple and sufficient fast path is to allow only * 32-bit inputs. Avoid zero inputs to avoid * negative zero issues (-1 * 0 = -0, for instance).
28471. duk_unicode_ids_m_let_noa[]
28472. Don't intercept a DoubleError, we may have caused the initial double * fault and attempting to intercept it will cause us to be called * recursively and exhaust the C stack.
28473. **comment:** * Node.js Buffer.prototype.equals() * Node.js Buffer.prototype.compare() * Node.js Buffer.compare()
 label: requirement
28474. indirect allocs
28475. Lightfuncs are always considered strict.
28476. implies DUK_HOBJECT_IS_BUFFEROBJECT
28477. safe, because matched (NUL causes a break)
28478. * Packed tval sanity
28479. longjmp type
28480. should be first on 64-bit platforms
28481. **comment:** * XXX: duk_uint_fast32_t should probably be used in many places here.
 label: code-design
28482. [...] | (crud) errobj]
28483. expensive flag
28484. validation performed by duk_hexval
28485. non-strict equality from here on
28486. allocate catcher and populate it (should be atomic)
28487. since duk_abandon_array_checked() causes a resize, there should be no gaps in keys
28488. **comment:** DUK_TVAL_SET_TVAL_UPDREF() is used a lot in executor, property lookups, * etc, so it's very important for performance. Measure when changing. * * NOTE: the source and destination duk_tval pointers may be the same, and * the macros MUST deal with that correctly.
 label: code-design
28489. if 0, 'str16' used, if > 0, 'strlist16' used
28490. probe sequence (open addressing)
28491. token was preceded by a lineterm
28492. -> [...]
28493. * Store entry state.
28494. Perform an intermediate join when this many elements have been pushed * on the value stack.
28495. duk_unicode_ids_m_let_noabmp[]
28496. * Parse variant 3 or 4. * * For variant 3 (e.g. "for (A in C) D;") the code for A (except the * final property/variable write) has already been emitted. The first * instruction of that code is at pc_v34_lhs; a JUMP needs to be inserted * there to satisfy control flow needs. * * For variant 4, if the variable declaration had an initializer * (e.g. "for (var A = B in C) D;") the code for the assignment * (B) has already been emitted. * * Variables set before entering here: * * pc_v34_lhs: insert a "JUMP L2" here (see doc/compiler.rst example). * reg_temps + 0: iteration target value (written to LHS) * reg_temps + 1: enumerator object
28497. 'return'
28498. **comment:** * String table algorithm: fixed size string table with array chaining * * The top level string table has a fixed size, with each slot holding * either NULL, string pointer, or pointer to a separately allocated * string pointer list. * * This is good for low memory environments using a pool allocator: the * top level allocation has a fixed size and the pointer lists have quite * small allocation size, which further matches the typical pool sizes * needed by objects, strings, property tables, etc.
 label: code-design
28499. **comment:** XXX: remove this native function and map 'stack' accessor * to the toString() implementation directly.
 label: code-design
28500. * Create a RegExp instance (E5 Section 15.10.7). * * Note: the output stack left by duk_regexp_compile() is directly compatible * with the input here. * * Input stack: [escaped_source bytecode] (both as strings) * Output stack: [RegExp]
28501. **comment:** log level could be popped but that's not necessary
 label: code-design
28502. thread currently running (only one at a time)
28503. numargs
28504. already defined, good
28505. * Currently only allowed only if yielding thread has only * Ecmascript activations (except for the Duktape.Thread.yield() * call at the callstack top) and none of them constructor * calls. * * This excludes the 'entry' thread which will always have * a preventcount > 0.
28506. Copy term name until end or '/'.
28507. -> [...] trap handler]
28508. Non-BMP characters within valid UTF-8 range: encode as is. * They'll decode back into surrogate pairs if the escaped * output is decoded.
28509. [...] v1 v2 name filename str] -> [...] str v2 name filename]
28510. DUK_TOK_DEBUGGER
28511. Init newly allocated slots (only).
28512. a complex (new) atom taints the result
28513. [this value]
28514. See: test-bug-tailcall-preventyield-assert.c.
28515. check_object_coercible
28516. **comment:** XXX: source code property
 label: code-design
28517. **comment:** XXX: Would be nice to share the fast path loop from duk_hex_decode() * and avoid creating a temporary buffer. However, there are some * differences which prevent trivial sharing: * * - Pipe char detection * - EOF detection * - Unknown length of input and output * * The best approach here would be a bufwriter and a reasonably sized * safe inner loop (e.g. 64 output bytes at a time).
 label: code-design
28518. **comment:** XXX: unnecessary, handle in adjust
 label: code-design
28519. [...] key trap handler]
28520. **comment:** * XXX: As noted above, a protected API call won't be counted as a * catcher. This is usually convenient, e.g. in the case of a top- * level duk_pcall(), but may not always be desirable. Perhaps add an * argument to treat them as catchers?
 label: code-design
28521. const
28522. even for zero-length string
28523. eat 'if'
28524. 14: getDay

28525. Custom behavior: plain buffer is used as internal buffer * without making a copy (matches Duktape.Buffer).
28526. function: create an arguments object on function call
28527. -> [...] target]
28528. * Nice debug log.
28529. **comment:** XXX: copy flags using a mask
 label: code-design
28530. [... func/retval] -> [...]
28531. **comment:** XXX: optimize loops
 label: code-design
28532. up to 36 bit codepoints
28533. no fractions in internal time
28534. ditto
28535. [key] -> []
28536. Track number of escapes; count not really needed but directive * prologues need to detect whether there were any escapes or line * continuations or not.
28537. [buf? res]
28538. **comment:** Duktape 0.11.0 and prior tried to optimize the resize by not * counting the number of actually used keys prior to the resize. * This worked mostly well but also caused weird leak-like behavior * as in: test-bug-object-prop-alloc-unbounded.js. So, now we count * the keys explicitly to compute the new entry part size.
 label: code-design
28539. undefined -> skip (replaced with empty)
28540. eat opening quote on first loop
28541. **comment:** * Array built-ins * * Note that most Array built-ins are intentionally generic and work even * when the 'this' binding is not an Array instance. To ensure this, * Array algorithms do not assume "magical" Array behavior for the "length" * property, for instance. * * XXX: the "Throw" flag should be set for (almost?) all [[Put]] and * [[Delete]] operations, but it's currently false throughout. Go through * all put/delete cases and check throw flag use. Need a new API primitive * which allows throws flag to be specified. * * XXX: array lengths above 2G won't work reliably. There are many places * where one needs a full signed 32-bit range ([-0xffffffff, 0xffffffff], * i.e. -33- bits). Although array 'length' cannot be written to be outside * the unsigned 32-bit range (E5.1 Section 15.4.5.1 throws a RangeError if so) * some intermediate values may be above 0xffffffff and this may not be always * correctly handled now (duk_uint32_t is not enough for all algorithms). * * For instance, push() can legitimately write entries beyond length 0xffffffff * and cause a RangeError only at the end. To do this properly, the current * push() implementation tracks the array index using a 'double' instead of a * duk_uint32_t (which is somewhat awkward). See test-bi-array-push-maxlen.js. * * On using "put" vs. "def" prop * ===== * * Code below must be careful to use the appropriate primitive as it matters * for compliance. When using "put" there may be inherited properties in * Array.prototype which cause side effects when values are written. When * using "define" there are no such side effects, and many test262 test cases * check for this (for real world code, such side effects are very rare). * Both "put" and "define" are used in the E5.1 specification; as a rule, * "put" is used when modifying an existing array (or a non-array 'this' * binding) and "define" for setting values into a fresh result array. * * Also note that Array instance 'length' should be writable, but not * enumerable and definitely not configurable: even Duktape code internally * assumes that an Array instance will always have a 'length' property. * Preventing deletion of the property is critical.
 label: code-design
28542. no need to decref
28543. 18: getMinutes
28544. 'exports'
28545. * Unicode letter check.
28546. DUK_USE_MATH_BUILTIN
28547. No field needed when strings are in ROM.
28548. value was undefined/unsupported
28549. eat 'function'
28550. keep in valstack
28551. * Must guarantee all actually used array entries will fit into * new entry part. Add one growth step to ensure we don't run out * of space right away.
28552. step 11
28553. IdentifierPart production with IdentifierStart and ASCII excluded
28554. Internal timevalue is already NaN, so don't touch it.
28555. flag for later write
28556. GH-303
28557. Popping one element is called so often that when footprint is not an issue, * compile a specialized function for it.
28558. DUK_TOK_SEMICOLON
28559. return value from Duktape.Thread.yield()
28560. No need to re-lookup 'act' at present: no side effects.
28561. Causes a ToNumber() coercion, but doesn't break coercion order since * year is coerced first anyway.
28562. Parts are in local time, convert when setting.
28563. * E5 Section 7.4. If the multi-line comment contains a newline, * it is treated like a single line terminator for automatic * semicolon insertion.
28564. Note: if target_pc1 == i, we'll optimize a jump to itself. * This does not need to be checked for explicitly; the case * is rare and max iter breaks us out.
28565. Backup 'caller' property and update its value.
28566. assume exactly 1 arg, which is why * is forbidden; arg size still * depends on type though.
28567. Add function object.
28568. finalization
28569. [ToObject(this) item1 ... itemN arr]
28570. 0x50...0x5f
28571. Rule table: first matching rule is used to determine what to do next.
28572. CATCH flag may be enabled or disabled here; it may be enabled if * the statement has a catch block but the try block does not throw * an error.
28573. * Init the heap object
28574. Inherit from ROM-based global object: less RAM usage, less transparent.
28575. Note: 0xff != DUK_BC_C_MAX
28576. COMMA
28577. Node.js return value for noAssert out-of-bounds reads is * usually (but not always) NaN. Return NaN consistently.
28578. **comment:** function must not be tail called
 label: code-design
28579. stage 3: update length (done by caller), decide return code
28580. idx_space
28581. statement does not terminate directive prologue
28582. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func exports fresh_require exports module]
28583. **comment:** XXX: optimize for buffer inputs: no need to coerce to a string * which causes an unnecessary interning.
 label: code-design
28584. * Table for hex decoding ASCII hex digits
28585. function: create new environment when called (see duk_hcompiledfunction)
28586. pop regexp res_obj or match string
28587. P
28588. required if NAMEBINDING set
28589. slow path decode
28590. don't care value, year is mandatory

28591. really 'not applicable' anymore, should not be referenced after this
28592. skip jump conditionally
28593. * Flags parsing (see E5 Section 15.10.4.1).
28594. for updating (all are set to < 0 for virtual properties)
28595. * Run (object) finalizers in the "to be finalized" work list.
28596. 'warn'
28597. DUK_USE_FILE_IO
28598. [... constructor arg1 ... argN final_cons fallback]
28599. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur.
28600. 'Array' object, array length and index exotic behavior
28601. a statement following a label cannot be a source element * (a function declaration).
28602. 24: setMilliseconds
28603. '\xffVarenv'
28604. **comment:** DUK_USE_DATE_GET_LOCAL_TZOFFSET() needs to be called with a * time value computed from UTC parts. At this point we only * have 'd' which is a time value computed from local parts, so * it is off by the UTC-to-local time offset which we don't know * yet. The current solution for computing the UTC-to-local * time offset is to iterate a few times and detect a fixed * point or a two-cycle loop (or a sanity iteration limit), * see test-bi-date-local-parts.js and test-bi-date-tzoffset-basic-fi.js. * * E5.1 Section 15.9.1.9: * UTC(t) = t - LocalTZA - DaylightSavingTA(t - LocalTZA) * * For NaN/inf, DUK_USE_DATE_GET_LOCAL_TZOFFSET() returns 0.
label: code-design
28605. DUK_TOK_BREAK
28606. We're conceptually between two opcodes; act->pc indicates the next * instruction to be executed. This is usually the correct pc/line to * indicate in Status. (For the 'debugger' statement this now reports * the pc/line after the debugger statement because the debugger opcode * has already been executed.)
28607. DUK_TOK_MOD_EQ
28608. **comment:** * Write to 'length' of an array is a very complex case * handled in a helper which updates both the array elements * and writes the new 'length'. The write may result in an * unconditional RangeError or a partial write (indicated * by a return code). * * Note: the helper has an unnecessary writability check * for 'length', we already know it is writable.
label: code-design
28609. **comment:** * Allocation size for 'curr_alloc' is alloc_size. There is no * automatic NUL terminator for buffers (see above for rationale). * * 'curr_alloc' is explicitly allocated with heap allocation * primitives and will thus always have alignment suitable for * e.g. duk_tval and an IEEE double.
label: code-design
28610. * ctx->prev_token token to process with duk__expr_led() * ctx->curr_token updated by caller
28611. Note: target registers a and a+1 may overlap with DUK_REGP(b). * Careful here.
28612. * Mark unreachable, finalizable objects. * * Such objects will be moved aside and their finalizers run later. They have * to be treated as reachability roots for their properties etc to remain * allocated. This marking is only done for unreachable values which would * be swept later (refzero_list is thus excluded). * * Objects are first marked FINALIZABLE and only then marked as reachability * roots; otherwise circular references might be handled inconsistently.
28613. **comment:** XXX: accept any duk_hbufferobject type as an input also?
label: code-design
28614. Note: DST adjustment is determined using UTC time.
28615. caller must check
28616. 2
28617. **comment:** Fresh require: require.id is left configurable (but not writable) * so that is not easy to accidentally tweak it, but it can still be * done with Object.defineProperty(). * * XXX: require.id could also be just made non-configurable, as there * is no practical reason to touch it.
label: code-design
28618. log this with a normal debug level because this should be relatively rare
28619. * Ecmascript execution, support primitives.
28620. * replace()
28621. out_clamped=NULL, RangeError if outside range
28622. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC
28623. idx_args = idx_func + 2
28624. handle comma and closing bracket
28625. * Avoid a GC if GC is already running. See duk_heap_mem_alloc().
28626. Note: this allows creation of internal strings.
28627. DUK_FLD_DOUBLE
28628. **comment:** * Logging * * Current logging primitive is a sprintf-style log which is convenient * for most C code. Another useful primitive would be to log N arguments * from value stack (like the Ecmascript binding does).
label: code-design
28629. caller and arguments must use the same thrower, [[ThrowTypeError]]
28630. bound function 'length' property is interesting
28631. DUK_REPLACEMENTS_H_INCLUDED
28632. [... error func]
28633. ext 0x000 is zero/subnormal
28634. There is a caller; it MUST be an Ecmascript caller (otherwise it would * match entry level check)
28635. restore PC
28636. allow negative PCs, behave as a no-op
28637. '\xffValue'
28638. * Activation defines
28639. Step 15: insert itemCount elements into the hole made above
28640. No need to replace the 'enum_target' value in stack, only the * enum_target reference. This also ensures that the original * enum target is reachable, which keeps the proxy and the proxy * target reachable. We do need to replace the internal _Target.
28641. XXX: ARRAY_PART for Array prototype?
28642. Coerce value to a number before computing check_length, so that * the field type specific coercion below can't have side effects * that would invalidate check_length.
28643. string is a valid array index
28644. * Fast path: assume no mutation, iterate object property tables * directly; bail out if that assumption doesn't hold.
28645. 'Number'
28646. buffer is behind a pointer, dynamic or external
28647. * in
28648. allow caller to give a const number with the DUK_CONST_MARKER
28649. DUK_USE_JSON_DECNUMBER_FASTPATH
28650. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack (e.g. if * lval is already a string).
28651. XXX: use DUK_HSTRING_FLAG_INTERNAL?
28652. func support for [[HasInstance]] checked in the beginning of the loop
28653. %p
28654. weak refs should be handled here, but no weak refs for * any non-string objects exist right now.
28655. pop key
28656. implementation specific
28657. * Log level check

28658. **comment:** XXX: unify handling with native call.
label: code-design

28659. internal extra elements assumed on function entry, * always added to user-defined 'extra' for e.g. the * duk_check_stack() call.

28660. * Complex atom ** The original code is used as a template, and removed at the end * (this differs from the handling of simple quantifiers). ** NOTE: there is no current solution for empty atoms in complex * quantifiers. This would need some sort of a 'progress' instruction. ** XXX: impose limit on maximum result size, i.e. atom_code_len * atom_copies?

28661. Leading zero is not counted towards precision digits; not * in the integer part, nor in the fraction part.

28662. 39: setYear

28663. **comment:** * ToString() (E5 Section 9.8) ** ==> implemented in the API.
label: requirement

28664. E5.1 Section 15.9.1.6

28665. DUK_USE_JSON_QUOTESTRING_FASTPATH

28666. always in executor

28667. because of valstack init policy

28668. embed: func ptr

28669. local

28670. Sanity check

28671. * Object environment record. * Binding (target) object is an external, uncontrolled object. * Identifier may be bound in an ancestor property, and may be * an accessor. Target can also be a Proxy which we must support * here.

28672. * Manipulate callstack for the call.

28673. Currently the bytecode executor and executor interrupt * instruction counts are off because we don't execute the * interrupt handler when we're about to exit from the initial * user call into Duktape. ** If we were to execute the interrupt handler here, the counts * would match. You can enable this block manually to check * that this is the case.

28674. **comment:** Note: combining comparison ops must be done carefully because * of uncomparable values (NaN): it's not necessarily true that * (x >= y) === !(x < y). Also, evaluation order matters, and * although it would only seem to affect the compiler this is * actually not the case, because there are also run-time coercions * of the arguments (with potential side effects). ** XXX: can be combined; check code size.
label: code-design

28675. 'typeof'

28676. Execute bytecode until returned or longjmp().

28677. overflow not possible, buffer limits

28678. 'interface'

28679. zero everything unless requested not to do so

28680. E5 Section 15.12.3, main algorithm, step 4.b.ii steps 1-4.

28681. * Node.js Buffer.isBuffer()

28682. DUK_TOK_COLON

28683. B -> target register for next key * C -> enum register

28684. -> [... flags escaped_source bytecode]

28685. backref n -> saved indices [n*2, n*2+1]

28686. value1 -> error object

28687. remove key

28688. **comment:** * String hash computation (interning). ** String hashing is performance critical because a string hash is computed * for all new strings which are candidates to be added to the string table. * However, strings actually added to the string table go through a codepoint * length calculation which dominates performance because it goes through * every byte of the input string (but only for strings added). ** The string hash algorithm should be fast, but on the other hand provide * good enough hashes to ensure both string table and object property table * hash tables work reasonably well (i.e., there aren't too many collisions * with real world inputs). Unless the hash is cryptographic, it's always * possible to craft inputs with maximal hash collisions. ** NOTE: The hash algorithms must match src/dukutil.py:duk_heap_hashstring() * for RÖM string support!
label: code-design

28689. **comment:** XXX: should the API call handle this directly, i.e. attempt * to duk_push_hobject(ctx, null) would push a null instead? * (On the other hand 'undefined' would be just as logical, but * not wanted here.)
label: code-design

28690. **comment:** For short strings, use a fixed temp buffer.
label: code-design

28691. label limits

28692. marker values for hash part

28693. DUK_TOK_GE

28694. -> [... closure]

28695. NULL pointer is required to encode to zero, so memset is enough.

28696. else an array initializer element

28697. * E5 Section 7.3 ** A LineTerminatorSequence essentially merges <CR> <LF> sequences * into a single line terminator. This must be handled by the caller.

28698. 35: setUTCMonth

28699. [... retval]

28700. Input shuffling happens before the actual operation, while output * shuffling happens afterwards. Output shuffling decisions are still * made at the same time to reduce branch clutter; output shuffle decisions * are recorded into X_out variables.

28701. 'new' MemberExpression Arguments

28702. assume plain values

28703. duk_reconst_t is unsigned, so use 0 as dummy value (ignored by caller)

28704. [... val obj]

28705. -> [... re_obj input bc saved_buf lastIndex]

28706. **comment:** (advance << 8) + token_type, updated at function end, * init is unnecessary but suppresses "may be used uninitialized" warnings.
label: code-design

28707. first, parse regexp body roughly

28708. Duktape/C (nativefunction) object, exotic 'length'

28709. DUK_HOBJECT_FLAG_EXOTIC_ARRAY varies

28710. 'this' binding

28711. [... val key] -> [... key val]

28712. isArray

28713. Cannot be compressed as a heap pointer because may point to * an arbitrary address.

28714. skip opening slash on first loop

28715. transfer flags

28716. DUK_TOK_INTERFACE

28717. may be undefined

28718. accept anything, expect first value (EOF will be * caught by duk__dec_value() below.

28719. iteration statements allow continue

28720. Clamp so that values at 'clamp_top' and above are wiped and won't * retain reachable garbage. Then extend to 'nregs' because we're * returning to an Ecmascript function.

28721. 'while'

28722. **comment:** Checking this here rather than in memory alloc primitives * reduces checking code there but means a failed allocation * will go through a few retries before giving up. That's * fine because this only happens during debugging.

label: code-design

28723. **comment:** * Array abandon check; abandon if: * * new_used / new_size < limit * new_used < limit * new_size || limit is 3 bits fixed point * new_used < limit' / 8 * new_size || *8 * 8*new_used < limit' * new_size || :8 * new_used < limit' * (new_size / 8) * * Here, new_used = a_used, new_size = a_size. * * Note: some callers use approximate values for a_used and/or a_size * (e.g. dropping a '+1' term). This doesn't affect the usefulness * of the check, but may confuse debugging.

label: code-design

28724. Target is before source. Source offset is expressed as * a "before change" offset. Account for the memmove.

28725. object is constructable

28726. regexp support disabled

28727. This should be equivalent to match() algorithm step 8.f.iii.2: * detect an empty match and allow it, but don't allow it twice.

28728. assume keys are compacted

28729. currently implicitly also DUK_USE_DOUBLE_LINKED_HEAP

28730. advance, whatever the current token is; parse next token in regexp context

28731. *ToInt32(), ToUint32(), ToUint16() (E5 Sections 9.5, 9.6, 9.7)

28732. reg to allocate

28733. preallocated temporaries (2) for variants 3 and 4

28734. [regexp input]

28735. function: function object is strict

28736. DUK_TOK_ALSHIFT

28737. **comment:** XXX: what to do if _Formals is not empty but compiler has * optimized it away -- read length from an explicit property * then?

label: code-design

28738. work list for objects whose refcounts are zero but which have not been * "finalized"; avoids recursive C calls when refcounts go to zero in a * chain of objects.

28739. **comment:** context and locale specific rules which cannot currently be represented * in the caseconv bitstream: hardcoded rules in C

label: code-design

28740. * Heap string representation. * * Strings are byte sequences ordinarily stored in extended UTF-8 format, * allowing values larger than the official UTF-8 range (used internally) * and also allowing UTF-8 encoding of surrogate pairs (CESU-8 format). * Strings may also be invalid UTF-8 altogether which is the case e.g. with * strings used as internal property names and raw buffers converted to * strings. In such cases the 'clen' field contains an inaccurate value. * * Ecmascript requires support for 32-bit long strings. However, since each * 16-bit codepoint can take 3 bytes in CESU-8, this representation can only * support about 1.4G codepoint long strings in extreme cases. This is not * really a practical issue.

28741. call is a direct eval call

28742. varname is still reachable

28743. Leave 'setter' on stack

28744. seconds

28745. **comment:** * Ecmascript compiler. * * Parses an input string and generates a function template result. * Compilation may happen in multiple contexts (global code, eval * code, function code). * * The parser uses a traditional top-down recursive parsing for the * statement level, and an operator precedence based top-down approach * for the expression level. The attempt is to minimize the C stack * depth. Bytecode is generated directly without an intermediate * representation (tree), at the cost of needing two passes over each * function. * * The top-down recursive parser functions are named "duk_parse_XXX". * * Recursion limits are in key functions to prevent arbitrary C recursion: * function body parsing, statement parsing, and expression parsing. * * See doc/compiler.rst for discussion on the design. * * A few typing notes: * * - duk_regconst_t: unsigned, no marker value for "none" * - duk_reg_t: signed, < 0 = none * - PC values: duk_int_t, negative values used as markers

label: code-design

28746. Append a "(line NNN)" to the "message" property of any error * thrown during compilation. Usually compilation errors are * SyntaxErrors but they can also be out-of-memory errors and * the like.

28747. **comment:** XXX: any way to detect faster whether something needs to be closed? * We now look up _Callee and then skip the rest.

label: requirement

28748. DUK_USE_VARIADIC_MACROS

28749. * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * ieee value = 1.ffff... * 2^(e - 1023) (normal) * = 0.ffff... * 2^(-1022) (denormal) * * algorithm v = f * b^e

28750. key is now reachable in the valstack

28751. MakeDate

28752. Request callback should push values for reply to client onto valstack

28753. inserted jump

28754. [enum_target res]

28755. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize or a forced relocating realloc?

label: code-design

28756. For Ecmascript strings, this check can only match for * initial UTF-8 bytes (not continuation bytes). For other * strings all bets are off.

28757. Then reuse the unwound activation; callstack was not shrunk so there is always space

28758. DUK_USE_DEBUGGER_DUMPHEAP || DUK_USE_DEBUGGER_INSPECT

28759. for storing/restoring the varmap binding for catch variable

28760. r <- (2 * f) * b^(e+1)

28761. [...]

28762. **comment:** Careful here and with other duk_put_prop_xxx() helpers: the * target object and the property value may be in the same value * stack slot (unusual, but still conceptually clear).

label: code-design

28763. function is strict

28764. must work for nargs <= 0

28765. Ignore digits beyond a radix-specific limit, but note them * in expt_adj.

28766. don't run finalizers; leave finalizable objects in finalize_list for next round

28767. [this value] -> [value this]

28768. Update 'buffer_length' to be the effective, safe limit which * takes into account the underlying buffer. This value will be * potentially invalidated by any side effect.

28769. 2: toTimeString

28770. **comment:** No need for constants pointer (= same as data). * * When using 16-bit packing alignment to 4 is nice. 'funcs' will be * 4-byte aligned because 'constants' are duk_tvals. For now the * inner function pointers are not compressed, so that 'bytecode' will * also be 4-byte aligned.

label: code-design

28771. * Initialize function state for a zero-argument function

28772. Guaranteed by recursion_limit setup so we don't have to * check twice.

28773. line continuation

28774. IsGenericDescriptor(desc) == true; this means in practice that 'desc' * only has [[Enumerable]] or [[Configurable]] flag updates, which are * allowed at this point.

28775. '-Infinity'

28776. * Resume a thread. * * The thread must be in resumable state, either (a) new thread which hasn't * yet started, or (b) a thread which has previously yielded. This method * must be called from an Ecmascript function. * * Args: * - thread * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.

28777. type tag (public)

28778. **comment:** It'd be nice to do something like this - but it doesn't * work for closures created inside the catch clause.

label: code-design

28779. don't re-enter e.g. during Eval

28780. mp <- b^e

28781. special handling of fmt==NULL

28782. Ensure space for final configuration (idx_rebase + num_stack_rets) * and intermediate configurations.

28783. parse value
28784. **comment:** Note: not honoring round-to-even should work but now generates incorrect * results. For instance, 1e23 serializes to "a000...", i.e. the first digit * equals the radix (10). Scaling stops one step too early in this case. * Don't know why this is the case, but since this code path is unused, it * doesn't matter.
 label: code-design
28785. **comment:** The 'strict' flag is copied to get the special [[Get]] of E5.1 * Section 15.3.5.4 to apply when a 'caller' value is a strict bound * function. Not sure if this is correct, because the specification * is a bit ambiguous on this point but it would make sense.
 label: code-design
28786. function may call direct eval
28787. **comment:** _Varmap: omitted if function is guaranteed not to do slow path identifier * accesses or if it would turn out to be empty of actual register mappings * after a cleanup. When debugging is enabled, we always need the varmap to * be able to lookup variables at any point.
 label: code-design
28788. [array totalLength]
28789. Opcode interval for a Date-based status/peek rate limit check. Only * relevant when debugger is attached. Requesting a timestamp may be a * slow operation on some platforms so this shouldn't be too low. On the * other hand a high value makes Duktape react to a pause request slowly.
28790. -> [... key val val']
28791. [... re_obj input]-> [... re_obj input bc]
28792. valstack slot for 1st token value
28793. side effects, possibly errors
28794. NOTE: Multiple evaluation of 'ptr' in this macro.
28795. Overflow of the execution counter is fine and doesn't break * anything here.
28796. mark-and-sweep: finalized (on previous pass)
28797. curr is accessor, desc is data
28798. [func thisArg arg1 ... argN]
28799. XXX: the insert helpers should ensure that the bytecode result is not * larger than expected (or at least assert for it). Many things in the * bytecode, like skip offsets, won't work correctly if the bytecode is * larger than say 2G.
28800. this function
28801. **comment:** 'tv' becomes invalid
 label: code-design
28802. is an actual function (not global/eval code)
28803. Catch attempts to use out-of-range reg/const. Without this * check Duktape 0.12.0 could generate invalid code which caused * an assert failure on execution. This error is triggered e.g. * for functions with a lot of constants and a try-catch statement. * Shuffling or opcode semantics change is needed to fix the issue. * See: test-bug-trycatch-many-constants.js.
28804. 'callee'
28805. Initial stack size satisfies the stack spare constraints so there * is no need to require stack here.
28806. 'byteOffset'
28807. **comment:** duk_handle_call_xxx: constructor call (i.e. called as 'new Foo()')
 label: code-design
28808. no operators
28809. Strict: never allow function declarations outside top level.
28810. [... arg1 ... argN envobj]
28811. * Field access
28812. 0x30...0x3f
28813. value1 -> label number, pseudo-type to indicate a continue continuation (for ENDFIN)
28814. true
28815. **comment:** XXX: common reg allocation need is to reuse a sub-expression's temp reg, * but only if it really is a temp. Nothing fancy here now.
 label: code-design
28816. only use the highest bit
28817. Function values are handled completely above (including * coercion results):
28818. distinct bignums, easy mistake to make
28819. **comment:** XXX: this is now a very unoptimal implementation -- this can be * made very simple by direct manipulation of the object internals, * given the guarantees above.
 label: code-design
28820. Split time value into parts. The time value is assumed to be an internal * one, i.e. finite, no fractions. Possible local time adjustment has already * been applied when reading the time value.
28821. If flags != 0 (strict or SameValue), thr can be NULL. For loose * equals comparison it must be != NULL.
28822. DUK_TOK_INSTANCEOF
28823. time
28824. update entry, allocating if necessary
28825. A structure for 'snapshotting' a point for rewinding
28826. * Ecmascript call
28827. cannot be arguments exotic
28828. **comment:** this is not strictly necessary, but helps debugging
 label: code-design
28829. * Prototypes for built-in functions not automatically covered by the * header declarations emitted by genbuiltins.py.
28830. push initial function call to new thread stack; this is * picked up by resume().
28831. 31: setUTCHours
28832. Decode a one-bit flag, and if set, decode a value of 'bits', otherwise return * default value. Return value is signed so that negative marker value can be * used by caller as a "not present" value.
28833. format plus something to avoid just missing
28834. **comment:** XXX: post-checks (such as no duplicate keys)
 label: code-design
28835. if property already exists, overwrites silently
28836. **comment:** 'val' is never NaN, so no need to normalize
 label: code-design
28837. * Lowercase digits for radix values 2 to 36. Also doubles as lowercase * hex nybble table.
28838. may be reg or const
28839. decl name
28840. rnd_state for duk_util_tinyrandom.c
28841. Note: heap->heap_thread, heap->curr_thread, and heap->heap_object * are on the heap allocated list.
28842. jump is inserted here
28843. misc
28844. length' and other virtual properties are not * enumerable, but are included if non-enumerable * properties are requested.
28845. only flag now
28846. [... func this arg1 ... argN]
28847. record previous atom info in case next token is a quantifier
28848. ..."
28849. * Ecmascript compliant [[GetOwnProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero *

* If property is not found: * - 'out_desc' is left in untouched state (possibly garbage)
* - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE set)
* - Returns zero
** Notes:
* - Getting a property descriptor may cause an allocation (and hence GC) to take place, hence reachability and refcount of all related values matter. Reallocation of value stack, properties, etc may invalidate many duk_tval pointers (concretely, those which reside in memory areas subject to reallocation). However, heap object pointers are never affected (heap objects have stable pointers).
* - The value of a plain property is always reachable and has a non-zero reference count.
* - The value of a virtual property is not necessarily reachable from elsewhere and may have a refcount of zero. Hence we push it onto the valstack for the caller, which ensures it remains reachable while it is needed.
* - There are no virtual accessor properties. Hence, all getters and setters are always related to concretely stored properties, which ensures that the get/set functions in the resulting descriptor are reachable and have non-zero refcounts. Should there be virtual accessor properties later, this would need to change.

28850. not enumerable

28851. %f and %lf both consume a 'long'

28852. * Catcher defines

28853. never executed

28854. **comment:** At least 'magic' has a significant impact on function identity.

label: code-design

28855. 'c'

28856. important to do this *after* pushing, to make the thread reachable for gc

28857. [offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT

28858. Should not happen.

28859. init pointer fields to null

28860. * toString(), valueOf()

28861. success, fixup pointers

28862. **comment:** XXX: several pointer comparison issues here

label: code-design

28863. insert (DUK_REOP_WIPERANGE, start, count) in reverse order so the order ends up right

28864. Check that value is a duk_hbufferobject and return a pointer to it.

28865. Note: DUK_VALSTACK_INITIAL_SIZE must be >= DUK_VALSTACK_API_ENTRY_MINIMUM * + DUK_VALSTACK_INTERNAL_EXTRA so that the initial stack conforms to spare requirements.

28866. **comment:** XXX: may need a 'length' filter for forEach()

label: code-design

28867. Note: assumes that duk_util_probe_steps size is 32

28868. guaranteed when building arguments

28869. updates thread state, minimizes its allocations

28870. {'_inf':true'}

28871. **comment:** Without variadic macros resort to comma expression trickery to handle debug prints. This generates a lot of harmless warnings. These hacks are not needed normally because DUK_D() and friends will hide the entire debug log statement from the compiler.

label: code-design

28872. **comment:** TimeClip() should never be necessary

label: code-design

28873. 0x10-0x1f

28874. E5.1 Section B.2.2, step 7.

28875. Parser part masks.

28876. end switch (tok)

28877. negative -> complex atom

28878. -> [... val root ""]

28879. result array

28880. XXX: fast path for arrays?

28881. -> [... ToObject(this) ToUint32(length)]

28882. Continue checked execution if there are breakpoints or we're stepping. * Also use checked execution if paused flag is active - it shouldn't be * because the debug message loop shouldn't terminate if it was. Step out * is handled by callstack unwind and doesn't need checked execution. * Note that debugger may have detached due to error or explicit request * above, so we must recheck attach status.

28883. 0x00...0x0f

28884. coercion order matters

28885. 'jc'

28886. 1 if property found, 0 otherwise

28887. fixed-format

28888. not strictly necessary because of lookahead '}' above

28889. **comment:** XXX: must be able to represent -len

label: code-design

28890. DUK_USE_DPRINT_COLORS

28891. expect_eof

28892. borrowed reference to catch variable name (or NULL if none)

28893. **comment:** XXX: these could be implemented as macros calling an internal function directly. * XXX: same issue as with Duktape.fin: there's no way to delete the property now (just set it to undefined).

label: code-design

28894. **comment:** XXX: set magic directly here? (it could share the c_nargs arg)

label: code-design

28895. * Preliminary activation record and valstack manipulation. * The concrete actions depend on whether we're dealing with a tail call (reuse an existing activation), a resume, or a normal call. ** The basic actions, in varying order, are: ** - Check stack size for call handling * - Grow call stack if necessary (non-tail-calls) * - Update current activation (idx_retval) if necessary * (non-tail, non-resume calls) * - Move start of args (idx_args) to valstack bottom * (tail calls) ** Don't touch valstack_bottom or valstack_top yet so that Duktape API calls work normally.

28896. result index for filter()

28897. either nonzero value is ok

28898. bc

28899. * DELVAR * See E5 Sections: * 11.4.1 The delete operator * 10.2.1.1.5 DeleteBinding (N) [declarative environment record] * 10.2.1.2.5 DeleteBinding (N) [object environment record] ** Variable bindings established inside eval() are deletable (configurable), * other bindings are not, including variables declared in global level. * Registers are always non-deletable, and the deletion of other bindings * is controlled by the configurable flag. ** For strict mode code, the 'delete' operator should fail with a compile time SyntaxError if applied to identifiers. Hence, no strict mode run-time deletion of identifiers should ever happen. This function should never be called from strict mode code!

28900. * Mark refzero_list objects. ** Objects on the refzero_list have no inbound references. They might have outbound references to objects that we might free, which would invalidate any references held by the refzero objects. A refzero object might also be rescued by refcount finalization. Refzero objects are treated as reachability roots to ensure they (or anything they point to) are not freed in mark-and-sweep.

28901. Negative zero needs special handling in JX/JC because it would otherwise serialize to '0', not '-0'.

28902. can't resize below 'top'

28903. round up roughly to next 'grow step'

28904. **comment:** * E5 Section 7.6: ** IdentifierPart: * IdentifierStart * UnicodeCombiningMark * UnicodeDigit * UnicodeConnectorPunctuation * <ZWNJ> [U+200C] * <ZWJ> [U+200D] ** IdentifierPart production has one multi-character production as part of its IdentifierStart alternative. The '\ character of an escape sequence is not matched here, see discussion in duk_unicode_is_identifier_start(). ** To match non-ASCII characters (codepoints >= 0x80), a very slow linear range-by-range scan is used. The codepoint is first compared to the IdentifierStart ranges, and if it doesn't match, then to a set consisting of code points in

IdentifierPart but not in * IdentifierStart. This is done to keep the unicode range data small, * at the expense of speed. * * The ASCII fast path consists of: * * 0x0030 ... 0x0039 ['0' ... '9', UnicodeDigit] * 0x0041 ... 0x005a ['A' ... 'Z', IdentifierStart] * 0x0061 ... 0x007a ['a' ... 'z', IdentifierStart] * 0x0024 ['\$', IdentifierStart] * 0x005f['_', IdentifierStart and * UnicodeConnectorPunctuation] * * UnicodeCombiningMark has no code points <= 0x7f. * * The matching code reuses the "identifier start" tables, and then * consults a separate range set for characters in "identifier part" * but not in "identifier start". These can be extracted with the * "src/extract_chars.py" script. * * UnicodeCombiningMark -> categories Mn, Mc * UnicodeDigit -> categories Nd * UnicodeConnectorPunctuation -> categories Pc

label: code-design

28905. last array index explicitly initialized, +1

28906. Value would yield 'undefined', so skip key altogether. * Side effects have already happened.

28907. need side effects, not value

28908. * Check function name validity now that we know strictness. * This only applies to function declarations and expressions, * not setter/getter name. * * See: test-dev-strict-mode-boundary.js

28909. Target may be a Proxy or property may be an accessor, so we must * use an actual, Proxy-aware hasprop check here. * * out->holder is NOT set to the actual duk_hobject where the * property is found, but rather the object binding target object.

28910. Note: for srclen=0, src may be NULL

28911. never an empty match, so step 13.c.iii can't be triggered

28912. Outer executor with setjmp/longjmp handling.

28913. if B/C is >= this value, refers to a const

28914. if debugging disabled

28915. if true, the stack already contains the final result

28916. * E5 Section 7.2 specifies six characters specifically as * white space: * * 0009;<control>;Cc;0;S;;;;N;CHARACTER TABULATION;;;; * 000B;<control>;Cc;0;S;;;;N;LINE TABULATION;;;; * 000C;<control>;Cc;0;WS;;;;N;FORM FEED (FF);;; * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; * FEFF;ZERO WIDTH NO-BREAK SPACE;Cf;0;BN;;;;N;BYTE ORDER MARK;;;; * * It also specifies any Unicode category 'Zs' characters as white * space. These can be extracted with the "src/extract_chars.py" script. * Current result: * * RAW OUTPUT: * ===== * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; * 1680;OGHAM SPACE MARK;Zs;0;WS;;;;N;;;; * 180E;MONGOLIAN VOWEL SEPARATOR;Zs;0;WS;;;;N;;;; * 2000;EN QUAD;Zs;0;WS;2002;;;;N;;;; * 2001;EM QUAD;Zs;0;WS;2003;;;;N;;;; * 2002;EN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2003;EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2004;THREE-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2005;FOUR-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2006;SIX-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2007;FIGURE SPACE;Zs;0;WS;<noBreak> 0020;;;;N;;;; * 2008;PUNCTUATION SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2009;THIN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 200A;HAIR SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 202F;NARROW NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;;;; * 205F;MEDIUM MATHEMATICAL SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 3000;IDEOGRAPHIC SPACE;Zs;0;WS;<wide> 0020;;;;N;;;; * * RANGES: * ===== * 0x0020 * 0x00a0 * 0x1680 * 0x180e * 0x2000 ... 0x20aa * 0x202f * 0x205f * 0x3000 * * A manual decoder (below) is probably most compact for this.

28917. ToNumber(bool) is +1.0 or 0.0. Tagged boolean value is always 0 or 1.

28918. when nothing is running, API calls are in non-strict mode

28919. **comment:** String length is computed here to avoid multiple evaluation * of a macro argument in the calling side.

label: code-design

28920. see duk_js_executor.c

28921. Convert heap string index to a token (reserved words)

28922. **comment:** XXX: refactor and share with other code

label: code-design

28923. order must match constants in genbuiltins.py

28924. Buffer is kept as is, with the fixed/dynamic nature of the * buffer only changed if requested. An external buffer * is converted into a non-external dynamic buffer in a * duk_to_dynamic_buffer() call.

28925. * EcmaScript compliant [[Delete]](P, Throw).

28926. -> [... str]

28927. args go here in parens

28928. 0xb0...0xbf

28929. Write in big endian

28930. only need to guarantee 1 more slot, but allocation growth is in chunks

28931. **comment:** alloc function typedefs in duktape.h

label: code-design

28932. i >= 0 would always be true

28933. **comment:** XXX: inline into a prototype walking loop?

label: code-design

28934. Lightfunc handling by ToObject() coercion.

28935. DUK_BUFOBJ_UINT16ARRAY

28936. 'else'

28937. 'String' object, array index exotic behavior

28938. A -> target reg * B -> object reg/const (may be const e.g. in "foo'[1]" * C -> key reg/const

28939. There is no eval() special handling here: eval() is never * automatically converted to a lightfunc.

28940. **comment:** XXX: identify enumeration target with an object index (not top of stack)

label: code-design

28941. Write unsigned 32-bit integer.

28942. The underlying types for offset/length in duk_hbufferobject is * duk_uint_t; make sure argument values fit and that offset + length * does not wrap.

28943. [body formals source template closure]

28944. no refcount changes

28945. [arg1 ... argN-1 body] -> [body arg1 ... argN-1]

28946. First evaluate LHS fully to ensure all side effects are out.

28947. properties object

28948. Call handling and success path. Success path exit cleans * up almost all state.

28949. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer * stack[4] = regexp match OR match string

28950. object is extensible

28951. * The length comparisons are present to handle * strings like "use strict\u0000foo" as required.

28952. lookup name from an open declarative record's registers

28953. relookup (side effects)

28954. **comment:** XXX: make a common DUK_USE_ option, and allow custom fixed seed?

label: code-design

28955. x <- y * z

28956. * Note: assuming new_a_size == 0, and that entry part contains * no conflicting keys, refcounts do not need to be adjusted for * the values, as they remain exactly the same. * * The keys, however, need to be interned, incref'd, and be * reachable for GC. Any intern attempt may trigger a GC and * claim any non-reachable strings, so every key must be reachable * at all times. * * A longjmp must not occur here, as the new_p allocation would * be freed without these keys being decref'd, hence the messy * decref handling if intern fails.

28957. **comment:** * 'props' contains {key,value,flags} entries, optional array entries, and * an optional hash lookup table for non-array entries in a single 'sliced' * allocation. There are several layout options, which differ slightly in * generated code size/speed and alignment/padding; duk_features.h selects * the layout used. * * Layout 1 (DUK_USE_HOBJECT_LAYOUT_1): * * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xffffffffUL = deleted

** Layout 2 (DUK_USE_HOBJECT_LAYOUT_2): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_uint8_t) + pad bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted ** Layout 3 (DUK_USE_HOBJECT_LAYOUT_3): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) ** In layout 1, the 'e_next' count is rounded to 4 or 8 on platforms * requiring 4 or 8 byte alignment. This ensures proper alignment * for the entries, at the cost of memory footprint. However, it's * probably preferable to use another layout on such platforms instead. ** In layout 2, the key and value parts are swapped to avoid padding * the key array on platforms requiring alignment by 8. The flags part * is padded to get alignment for array entries. The 'e_next' count does * not need to be rounded as in layout 1. ** In layout 3, entry values and array values are always aligned properly, * and assuming pointers are at most 8 bytes, so are the entry keys. Hash * indices will be properly aligned (assuming pointers are at least 4 bytes). * Finally, flags don't need additional alignment. This layout provides * compact allocations without padding (even on platforms with alignment * requirements) at the cost of a bit slower lookups. ** Objects with few keys don't have a hash index; keys are looked up linearly, * which is cache efficient because the keys are consecutive. Larger objects * have a hash index part which contains integer indexes to the entries part. ** A single allocation reduces memory allocation overhead but requires more * work when any part needs to be resized. A sliced allocation for entries * makes linear key matching faster on most platforms (more locality) and * skimps on flags size (which would be followed by 3 bytes of padding in * most architectures if entries were placed in a struct). ** 'props' also contains internal properties distinguished with a non-BMP * prefix. Often used properties should be placed early in 'props' whenever * possible to make accessing them as fast as possible.

label: code-design

28958. Catches >0x100000000 and negative values.

28959. * Entry points

28960. * Error built-ins

28961. Packed or unpacked tval

28962. * PC-to-line constants

28963. top-down expression parser

28964. E5 Section 15.4.5.1, steps 4.e.i - 4.e.ii

28965. fall through

28966. x

28967. **comment:** XXX: write protect after flag? -> any chance of handling it here?

label: code-design

28968. Term was '!' and is eaten entirely (including dup slashes).

28969. fast path for ASCII

28970. num_stack_res

28971. %lx

28972. * Not found as concrete or virtual

28973. 'Buffer'

28974. [... env target target]

28975. DUK_TOK_LOR

28976. DUK_OP_DECLVAR flags in A; bottom bits are reserved for propdesc flags (DUK_PROPDESC_FLAG_XXX)

28977. Day-of-month is one-based in the API, but zero-based * internally, so fix here. Note that month is zero-based * both in the API and internally.

28978. The coercion order must match the ToPropertyDescriptor() algorithm * so that side effects in coercion happen in the correct order. * (This order also happens to be compatible with duk_def_prop(), * although it doesn't matter in practice.)

28979. triggers garbage digit check below

28980. relevant array index is non-configurable, blocks write

28981. **comment:** the NaN variant we use

label: code-design

28982. * Compact the function template.

28983. PRIMARY EXPRESSIONS

28984. * indexOf(), lastIndexOf()

28985. Sometimes this assert is not true right now; it will be true after * rounding. See: test-bug-numconv-mantissa-assert.js.

28986. Automatic error throwing, retval check.

28987. Fast path for the case where the register * is a number (e.g. loop counter).

28988. always push some string

28989. spare

28990. Note: new_e_next matches pushed temp key count, and nothing can * fail above between the push and this point.

28991. * Heap flags

28992. Object extra properties. ** There are some difference between function templates and functions. * For example, function templates don't have .length and nargs is * normally used to instantiate the functions.

28993. We intentionally ignore the duk_safe_call() return value and only * check the output type. This way we don't also need to check that * the returned value is indeed a string in the success case.

28994. * Arguments exotic behavior not possible for new properties: all * magically bound properties are initially present in the arguments * object, and if they are deleted, the binding is also removed from * parameter map.

28995. get_value

28996. clear array part flag only after switching

28997. '!'

28998. tv1 points to value storage

28999. Map DUK_HBUFFEROBJECT_ELEM_xxx to prototype object built-in index. * Sync with duk_hbufferobject.h.

29000. * Create "fallback" object to be used as the object instance, * unless the constructor returns a replacement value. * Its internal prototype needs to be set based on "prototype" * property of the constructor.

29001. Slightly modified "Bernstein hash" from: ** http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx ** Modifications: string skipping and reverse direction similar to * Lua 5.1.5, and different hash initializer. ** The reverse direction ensures last byte it always included in the * hash which is a good default as changing parts of the string are * more often in the suffix than in the prefix.

29002. ascii fast path: avoid decoding utf-8

29003. just in case

29004. entry_top + 4

29005. m- <- (* m- B)

29006. UTC

29007. no valstack space check

29008. End of input (NUL) goes through slow path and causes SyntaxError.

29009. [... enum_target res trap_result val]

29010. **comment:** XXX: unnecessary '%s' formatting here, but cannot use * 'msg' as a format string directly.

label: code-design

29011. ptr_curr_pc != NULL only when bytecode executor is active.

29012. Restore entry thread executor curr_pc stack frame pointer.

29013. Storing the entry top is cheaper here to ensure stack is correct at exit, * as there are several paths out.

29014. duk_hobject_set_length_zero(thr, func->h_funcs);

29015. fall thru

29016. see above

29017. * Struct defines

29018. [... name reg/null] -> [...]
29019. mandatory if du.d is a NaN
29020. duk_tval ptr for 'func' on stack (borrowed reference)
29021. For tv1 == tv2, this is a no-op (no explicit check needed).
29022. Use b[] to access the size of the union, which is strictly not * correct. Can't use fixed size unless there's feature detection * for pointer byte size.
29023. local copy to avoid relookup
29024. **comment:** XXX: this has some overlap with object inspection; remove this and make * DumpHeap return lists of heapptrs instead?
 label: code-design
29025. Return to the bytecode executor caller which will unwind stacks. * Return value is already on the stack top: [... retval].
29026. * Hobject Ecmascript [[Class]].
29027. Handle one slow path unit (or finish if we're done).
29028. **comment:** Copy the .buffer property, needed for TypedArray.prototype.subarray(). ** XXX: limit copy only for TypedArray classes specifically?
 label: code-design
29029. DUK_HEAPHDR_HAS_FINALIZED may or may not be set.
29030. enough to cover the whole mantissa
29031. DUK_TOK_BXOR_EQ
29032. **comment:** * Hobject enumeration support. ** Creates an internal enumeration state object to be used e.g. with for-in * enumeration. The state object contains a snapshot of target object keys * and internal control state for enumeration. Enumerator flags allow caller * to e.g. request internal/non-enumerable properties, and to enumerate only * "own" properties. ** Also creates the result value for e.g. Object.keys() based on the same * internal structure. ** This snapshot-based enumeration approach is used to simplify enumeration: * non-snapshot-based approaches are difficult to reconcile with mutating * the enumeration target, running multiple long-lived enumerators at the * same time, garbage collection details, etc. The downside is that the * enumerator object is memory inefficient especially for iterating arrays.
 label: code-design
29033. repl string scan
29034. [key result] -> [result]
29035. '\x0ffTracedata'
29036. ... and its 'message' from an instance property
29037. AUTHORS.rst
29038. Potential direct eval call detected, flag the CALL * so that a run-time "direct eval" check is made and * special behavior may be triggered. Note that this * does not prevent 'eval' from being register bound.
29039. prefer jump
29040. **comment:** * Arithmetic, binary, and logical helpers. ** Note: there is no opcode for logical AND or logical OR; this is on * purpose, because the evalution order semantics for them make such * opcodes pretty pointless: short circuiting means they are most * comfortably implemented as jumps. However, a logical NOT opcode * is useful. ** Note: careful with duk_tval pointers here: they are potentially * invalidated by any DECREF and almost any API call. It's still * preferable to work without making a copy but that's not always * possible.
 label: code-design
29041. automatic length update
29042. args
29043. 'String'
29044. With ROM-based strings, heap->strs[] and thr->strs[] are omitted * so nothing to initialize for strs[].
29045. * Note: prototype chain is followed BEFORE first comparison. This * means that the instanceof lval is never itself compared to the * rval.prototype property. This is apparently intentional, see E5 * Section 15.3.5.3, step 4.a. ** Also note: ** js> (function() {}) instanceof Function * true * js> Function instanceof Function * true ** For the latter, h_proto will be Function.prototype, which is the * built-in Function prototype. Because Function.[[Prototype]] is * also the built-in Function prototype, the result is true.
29046. not a vararg function
29047. Shouldn't happen but check anyway.
29048. E5 Sections 11.8.3, 11.8.5; x <= y --> not (x > y) --> not (y < x)
29049. Used by duk_emit*() calls so that we don't shuffle the loadints that * are needed to handle indirect opcodes.
29050. No strs[] pointer.
29051. **comment:** XXX: assertion that entries >= old_len are already unused
 label: code-design
29052. e.g. DUK_OP_PREINCR
29053. eat 'while'
29054. nuke values at idx_rebase to get the first retval (initially * at idx_rcbase) to idx_rebase
29055. input radix
29056. char: use int cast
29057. result
29058. exponent digit
29059. **comment:** This is a cleaner approach and also produces smaller code than * the other alternative. Use duk_require_string() for format * safety (although the source property should always exist).
 label: code-design
29060. defprop_flags
29061. s <- 2
29062. spaces (nargs - 1) + newline
29063. * Parse inner function
29064. if accessor without getter, return value is undefined
29065. **comment:** * For non-ASCII strings, we need to scan forwards or backwards * from some starting point. The starting point may be the start * or end of the string, or some cached midpoint in the string * cache. ** For "short" strings we simply scan without checking or updating * the cache. For longer strings we check and update the cache as * necessary, inserting a new cache entry if none exists.
 label: code-design
29066. XXX: len >= 0x80000000 won't work below because we need to be * able to represent -len.
29067. who resumed us (if any)
29068. out_clamped==NULL -> RangeError if outside range
29069. relookup after possible realloc
29070. [... regexp_object escaped_source bytecode]
29071. m+ <- (* m+ B)
29072. bytecode opcode (or extraop) for binary ops
29073. forget temp
29074. * Shared assignment expression handling ** args = (opcode << 8) + rbp * * If 'opcode' is DUK_OP_NONE, plain assignment without arithmetic. * Syntactically valid left-hand-side forms which are not accepted as * left-hand-side values (e.g. as in "f() = 1") must NOT cause a * SyntaxError, but rather a run-time ReferenceError. ** When evaluating X <op>= Y, the LHS (X) is conceptually evaluated * to a temporary first. The RHS is then evaluated. Finally, the * <op> is applied to the initial value of RHS (not the value after * RHS evaluation), and written to X. Doing so concretely generates * inefficient code so we'd like to avoid the temporary when possible. * See: <https://github.com/svaarala/duktape/pull/992>. ** The expression value (final LHS value, written to RHS) is * conceptually copied into a fresh temporary so that it won't * change even if the LHS/RHS values change in outer expressions. * For example, it'd be generally incorrect for the expression value * to be the RHS register binding, unless there's a guarantee that it * won't change during further expression evaluation. Using the * temporary concretely produces inefficient bytecode, so we try to * avoid the extra temporary for some known-to-be-safe cases. * Currently the only safe case we detect is a "top level assignment", * for example "x = y + z;", where the assignment expression value is * ignored. * See: test-dev-assign-expr.js and test-bug-assign-mutate-gh381.js.

29075. TRYCATCH, cannot emit now (not enough info)
29076. Value stack slot limits: these are quite approximate right now, and * because they overlap in control flow, some could be eliminated.
29077. absolute position for digit considered for rounding
29078. -> [... this timeval_new]
29079. TypeError if wrong; class check, see E5 Section 15.10.6
29080. DUK_HOBJECT_FLAG_EXOTIC_DUKFUNC: omitted here intentionally
29081. * Byte order. Important to self check, because on some exotic platforms * there is no actual detection but rather assumption based on platform * defines.
29082. Both inputs are zero; cases where only one is zero can go * through main algorithm.
29083. same thread
29084. 'Boolean'
29085. * Inside one or more 'with' statements fall back to slow path always. * (See e.g. test-stmt-with.js.)
29086. * Init the heap thread
29087. XXX: need a duk_require_func_or_lfunc_coerce()
29088. avoid referencing, invalidated
29089. Don't support "straddled" source now.
29090. * Object compaction. * * Compaction is assumed to never throw an error.
29091. [arg1 ... argN this loggerLevel loggerName]
29092. duk_tval intentionally skipped
29093. * Since built-ins are not often extended, compact them.
29094. * Compilation
29095. Push a new ArrayBuffer (becomes view .buffer)
29096. idx_start
29097. Note: we rely on the _Varmap having a bunch of nice properties, like: * - being compacted and unmodified during this process * - not containing an array part * - having correct value types
29098. **comment:** XXX: return indication of "terminalness" (e.g. a 'throw' is terminal)
label: code-design
29099. may be NULL if no constants or inner funcs
29100. Equivalent year for DST calculations outside [1970,2038[range, see * E5 Section 15.9.1.8. Equivalent year has the same leap-year-ness and * starts with the same weekday on Jan 1. * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
29101. empty expressions can be detected conveniently with nud/led counts
29102. There are at most 7 args, but we use 8 here so that also * DUK_DATE_IDX_WEEKDAY gets initialized (to zero) to avoid the potential * for any Valgrind gripes later.
29103. [... val callback thisArg val i obj]
29104. [... this tracedata sep this str1 ... strN]
29105. important chosen base types
29106. Digit count indicates number of fractions (i.e. an absolute * digit index instead of a relative one). Used together with * DUK_N2S_FLAG_FIXED_FORMAT for toFixed().
29107. must also check refzero_list
29108. MEMBER/NEW/CALL EXPRESSIONS
29109. **comment:** XXX: add flag to indicate whether caller cares about return value; this * affects e.g. handling of assignment expressions. This change needs API * changes elsewhere too.
label: code-design
29110. Fast jumps (which avoid longjmp) jump directly to the jump sites * which are always known even while the iteration/switch statement * is still being parsed. A final peephole pass "straightens out" * the jumps.
29111. [... func]
29112. * Function gets no new environment when called. This is the * case for global code, indirect eval code, and non-strict * direct eval code. There is no direct correspondence to the * E5 specification, as global/eval code is not exposed as a * function.
29113. Wrappers for calling standard math library methods. These may be required * on platforms where one or more of the math built-ins are defined as macros * or inline functions and are thus not suitable to be used as function pointers.
29114. **comment:** Throwing an error this deep makes the error rather vague, but * saves hundreds of bytes of code.
label: code-design
29115. This is not strictly necessary, but avoids compiler warnings; e.g. * gcc won't reliably detect that no uninitialized data is read below.
29116. Trigger at zero or below
29117. Custom coercion for API
29118. **comment:** Note: cannot read more than 24 bits without possibly shifting top bits out. * Fixable, but adds complexity.
label: code-design
29119. Almost all global level Function objects are constructable * but not all: Function.prototype is a non-constructable, * callable Function.
29120. source ends before dest starts
29121. * Statement terminator check, including automatic semicolon * handling. After this step, 'curr_tok' should be the first * token after a possible statement terminator.
29122. * Exposed string-to-number API * * Input: [string] * Output: [number] * * If number parsing fails, a NaN is pushed as the result. If number parsing * fails due to an internal error, an InternalError is thrown.
29123. 64-bit OK because always ≥ 0
29124. reg/const for switch value
29125. **comment:** When formatting an argument to a string, errors may happen from multiple * causes. In general we want to catch obvious errors like a toLogString() * throwing an error, but we don't currently try to catch every possible * error. In particular, internal errors (like out of memory or stack) are * not caught. Also, we expect Error.toString() to not throw an error.
label: code-design
29126. Each round of finalizer execution may spawn new finalizable objects * which is normal behavior for some applications. Allow multiple * rounds of finalization, but use a shrinking limit based on the * first round to detect the case where a runaway finalizer creates * an unbounded amount of new finalizable objects. Finalizer rescue * is not supported: the semantics are unclear because most of the * objects being finalized here are already reachable. The finalizer * is given a boolean to indicate that rescue is not possible. * * See discussion in: <https://github.com/svaarala/duktape/pull/473>
29127. no prototype, updated below
29128. unwind to 'resume' caller
29129. 0x0f-0xff
29130. **comment:** XXX: missing trap result validation for non-configurable target keys * (must be present), for non-extensible target all target keys must be * present and no extra keys can be present. * <http://www.ecma-international.org/ecma-262/6.0/#sec-proxy-object-internal-methods-and-internal-slots-ownpropertykeys>
label: code-design
29131. **comment:** NOTE: lightfuncs are coerced to full functions because * lightfuncs don't fit into a property value slot. This * has some side effects, see test-dev-lightfunc-accessor.js.
label: code-design
29132. IdentityEscape, with dollar added as a valid additional * non-standard escape (see test-regexp-identity-escape-dollar.js). * Careful not to match end-of-buffer (<0) here.
29133. 1st related value (type specific)
29134. The caller is responsible for being sure that bytecode being loaded * is valid and trusted. Invalid bytecode can cause memory unsafe * behavior directly during loading or later during bytecode execution * (instruction validation would be quite complex to implement). * * This signature check is the only sanity check for detecting * accidental invalid inputs. The initial 0xFF byte ensures no * ordinary string will be accepted by accident.
29135. Shared object part
29136. always throw ReferenceError for unresolvable

29137. * Mark roots, hoping that recursion limit is not normally hit. * If recursion limit is hit, run additional reachability rounds * starting from "temproots" until marking is complete. ** Marking happens in two phases: first we mark actual reachability * roots (and run "temproots" to complete the process). Then we * check which objects are unreachable and are finalizable; such * objects are marked as FINALIZABLE and marked as reachability * (and "temproots" is run again to complete the process). ** The heap finalize_list must also be marked as a reachability root. * There may be objects on the list from a previous round if the * previous run had finalizer skip flag.

29138. Shift to sign extend.

29139. '^'

29140. required_desc_flags

29141. **comment:** Stack indices for better readability
label: code-design

29142. * Try registers

29143. no prototype or class yet

29144. shadowed; update value

29145. E5 Section 8.6.2 + custom classes

29146. **comment:** Lookup current thread; use the stable 'entry_thread' for this to * avoid clobber warnings. Any valid, reachable 'thr' value would be * fine for this, so using 'entry_thread' is just to silence warnings.
label: code-design

29147. RELATIONAL EXPRESSION

29148. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property existence check right now.

29149. These are just convenience "wiping" of state. Side effects should * not be an issue here: thr->heap and thr->heap->lj have a stable * pointer. Finalizer runs etc capture even out-of-memory errors so * nothing should throw here.

29150. Zero escape (also allowed in non-strict mode)

29151. roughly 64 bytes

29152. XXX: this doesn't actually work properly for tail calls, so * tail calls are disabled when DUK_USE_NONSTD_FUNC_CALLER_PROPERTY * is in use.

29153. * Dragon4 slow path (binary) digit generation. * An extra digit is generated for rounding.

29154. this is a bit approximate (errors out before max is reached); this is OK

29155. **comment:** * Parse a function-body-like expression (FunctionBody or Program * in E5 grammar) using a two-pass parse. The productions appear * in the following contexts: * * - function expression * - function statement * - function declaration * - getter in object literal * - setter in object literal * - global code * - eval code * - Function constructor body ** This function only parses the statement list of the body; the argument * list and possible function name must be initialized by the caller. * For instance, for Function constructor, the argument names are originally * on the value stack. The parsing of statements ends either at an EOF or * a closing brace; this is controlled by an input flag. ** Note that there are many differences affecting parsing and even code * generation: * * - Global and eval code have an implicit return value generated * by the last statement; function code does not * * - Global code, eval code, and Function constructor body end in * an EOF, other bodies in a closing brace ('}') * * Upon entry, 'curr_tok' is ignored and the function will pull in the * first token on its own. Upon exit, 'curr_tok' is the terminating * token (EOF or closing brace).
label: code-design

29156. When torture not enabled, can just use the same helper because * 'reg' won't get spilled.

29157. Note: actual update happens once write has been completed * without error below. The write should always succeed * from a specification viewpoint, but we may e.g. run out * of memory. It's safer in this order.

29158. Convert buffer to result string.

29159. default prototype (Note: 'thr' must be reachable)

29160. we know these because enum objects are internally created

29161. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
label: code-design

29162. **comment:** Direct eval requires that there's a current * activation and it is an Ecmascript function. * When Eval is executed from e.g. cooperate API * call we'll need to do an indirect eval instead.
label: code-design

29163. 1 = num actual 'return values'

29164. should never happen

29165. '-' verbatim

29166. Regexp tokens

29167. avoid attempt to compact the current object

29168. DUK_TOK_TRUE

29169. idx_bottom and idx_retval are only used for book-keeping of * Ecmascript-initiated calls, to allow returning to an Ecmascript * function properly. They are duk_size_t to match the convention * that value stack sizes are duk_size_t and local frame indices * are duk_idx_t.

29170. **comment:** This loop is optimized for size. For speed, there should be * two separate loops, and we should ensure that memcmp() can be * used without an extra "will searchstring fit" check. Doing * the preconditioning for 'p' and 'p_end' is easy but cpos * must be updated if 'p' is wound back (backward scanning).
label: code-design

29171. min(incl)

29172. may be equal

29173. traceback depth ensures fits into 16 bits

29174. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack. ** The right hand side could be a light function (as they generally * behave like objects). Light functions never have a 'prototype' * property so E5.1 Section 15.3.5.3 step 3 always throws a TypeError. * Using duk_require_hobject() is thus correct (except for error msg).

29175. DUK_TOK_GET

29176. * Various

29177. Module has now evaluated to a wrapped module function. Force its * .name to match module.name (defaults to last component of resolved * ID) so that it is shown in stack traces too. Note that we must not * introduce an actual name binding into the function scope (which is * usually the case with a named function) because it would affect the * scope seen by the module and shadow accesses to globals of the same name. * This is now done by compiling the function as anonymous and then forcing * its .name without setting a "has name binding" flag.

29178. Characters outside BMP cannot be escape()'d. We could * encode them as surrogate pairs (for codepoints inside * valid UTF-8 range, but not extended UTF-8). Because * escape() and unescape() are legacy functions, we don't.

29179. trailer

29180. * Check whether we need to abandon an array part (if it exists)

29181. [... env callee]

29182. **comment:** These is not 100% because format would need to be non-portable "long long". * Also print out as doubles to catch cases where the "long" type is not wide * enough; the limits will then not be printed accurately but the magnitude * will be correct.
label: code-design

29183. an Ecmascript function

29184. **comment:** avoid pressure to add/remove strings, invalidation of call data argument, etc.
label: code-design

29185. end of bytecode

29186. key is 'length', cannot match argument exotic behavior

29187. * First check whether property exists; if not, simple case. This covers * steps 1-4.

29188. * Do-while statement is mostly trivial, but there is special * handling for automatic semicolon handling (triggered by the * DUK_ALLOW_AUTO_SEMI_ALWAYS) flag related to a bug filed at: * * https://bugs.ecmascript.org/show_bug.cgi?id=8 * * See doc/compiler.rst for details.

29189. must be found: was found earlier, and cannot be inherited

29190. Input value should be on stack top and will be coerced and * popped. Refuse to update an Array's 'length' to a value * outside the 32-bit range. Negative zero is accepted as zero.

29191. shift in zeroes

29192. free object and all auxiliary (non-heap) allocs

29193. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Returns NULL for a lightfunc.

29194. bytes in dest

29195. stack[0] = regexp * stack[1] = string

29196. flags

29197. Check for maximum buffer length.

29198. 'Thread'

29199. **comment:** Pointer to bytecode executor's 'curr_pc' variable. Used to copy * the current PC back into the topmost activation when activation * state is about to change (or "syncing" is otherwise needed). This * is rather awkward but important for performance, see execution.rst.
label: code-design

29200. The Quote(value) operation: quote a string. * * Stack policy: [] -> [].

29201. Output #2: last component name

29202. tc1 = false, tc2 = false

29203. 0

29204. Exponent without a sign or with a +/- sign is accepted * by all call sites (even JSON.parse()).

29205. 0x90-0x9f

29206. **comment:** XXX: Shuffling support could be implemented here so that LDINT+LDINTX * would only shuffle once (instead of twice). The current code works * though, and has a smaller compiler footprint.
label: code-design

29207. * Write to entry part

29208. To use the shared helper need the virtual index.

29209. * Slow delete, but we don't care as we're already in a very slow path. * The delete always succeeds: key has no exotic behavior, property * is configurable, and no resize occurs.

29210. LAYOUT 1

29211. For len == 0, i is initialized to len - 1 which underflows. * The condition (i < len) will then exit the for-loop on the * first round which is correct. Similarly, loop termination * happens by i underflowing.

29212. Raw internal valstack access macros: access is unsafe so call site * must have a guarantee that the index is valid. When that is the case, * using these macro results in faster and smaller code than duk_get_tval(). * Both 'ctx' and 'idx' are evaluated multiple times, but only for asserts.

29213. * Node.js Buffer.isEncoding()

29214. array entries are all plain values

29215. **comment:** Buffer values are encoded in (lowercase) hex to make the * binary data readable. Base64 or similar would be more * compact but less readable, and the point of JX/JC * variants is to be as useful to a programmer as possible.
label: code-design

29216. **comment:** XXX: the helper currently assumes stack top contains new * 'length' value and the whole calling convention is not very * compatible with what we need.
label: code-design

29217. Coerce to number before validating pointers etc so that the * number coercions in duk_hbufferobject_validated_write() are * guaranteed to be side effect free and not invalidate the * pointer checks we do here.

29218. Mark-and-sweep interval is relative to combined count of objects and * strings kept in the heap during the latest mark-and-sweep pass. * Fixed point .8 multiplier and .0 adder. Trigger count (interval) is * decreased by each (re)allocation attempt (regardless of size), and each * refzero processed object. ** 'SKIP' indicates how many (re)allocations to wait until a retry if * GC is skipped because there is no thread do it with yet (happens * only during init phases).

29219. XXX: assert 'c' is an enumerator

29220. two special escapes: '\n' and "", other printables as is

29221. for indexOf, ToInteger(undefined) would be 0, i.e. correct, but * handle both indexOf and lastIndexOf specially here.

29222. DUK_USE_AUGMENT_ERROR_THROW

29223. * Exponent notation for non-base-10 numbers isn't specified in ECMAScript * specification, as it never explicitly turns up: non-decimal numbers can * only be formatted with Number.prototype.toString([radix]) and for that, * behavior is not explicitly specified. ** Logical choices include formatting the exponent as decimal (e.g. binary * 100000 as 1e+5) or in current radix (e.g. binary 100000 as 1e+101). * The Dragon4 algorithm (in the original paper) prints the exponent value * in the target radix B. However, for radix values 15 and above, the * exponent separator 'e' is no longer easily parseable. Consider, for * instance, the number "1.faecee+1c".

29224. Template functions are not strictly constructable (they don't * have a "prototype" property for instance), so leave the * DUK_HOBJECT_FLAG_CONSTRUCTABLE flag cleared here.

29225. **comment:** Most practical strings will go here.
label: code-design

29226. non-strict eval: env is caller's env or global env (direct vs. indirect call) * global code: env is is global env

29227. **comment:** XXX: use the exactly same arithmetic function here as in executor
label: code-design

29228. -> [... lval new_rval]

29229. sep (even before first one)

29230. **comment:** XXX: use advancing pointers instead of index macros -> faster and smaller?
label: code-design

29231. 0x7F is special

29232. IEEE requires that zeros compare the same regardless * of their signed, so if both x and y are zeroes, they * are caught above.

29233. Signed integers always map to 4 bytes now.

29234. Resolve Proxy targets until Proxy chain ends. No explicit check for * a Proxy loop: user code cannot create such a loop without tweaking * internal properties directly.

29235. estimate is valid

29236. Note: it's nice if size is 2^N (not 4x4 = 16 bytes on 32 bit)

29237. always true, arg is unsigned

29238. ArrayBuffer: unlike any other argument variant, create * a view into the existing buffer.

29239. DUK_TOK_DO

29240. * Slow path: potentially requires function calls for coercion

29241. 'ArrayBuffer'

29242. Initialize index so that we skip internal control keys.

29243. flags have been already cleared

29244. * ArrayBuffer, DataView, and TypedArray constructors

29245. **comment:** * XXX: for now, indicate that an expensive catch binding * declarative environment is always needed. If we don't * need it, we don't need the const_varname either.
label: code-design

29246. -> [... this timeval_new timeval_new]

29247. DUK_TOK_CONST

29248. **comment:** XXX: unnecessary copying of values? Just set 'top' to * b + c, and let the return handling fix up the stack frame?
label: code-design

29249. Slot C

29250. **comment:** Array is dense and contains only strings, but ASIZE may * be larger than used part and there are UNUSED entries.

label: code-design

29251. At least: ... [err]
29252. wrapped
29253. whole or fraction digit
29254. * Constructor calls
29255. only 'length'
29256. DUK_USE_DEBUGGER_DUMPHEAP
29257. 'toISOString'
29258. 'res' contains expression value
29259. No other escape beginning with a digit in strict mode
29260. **comment:** Use Murmurhash2 directly for short strings, and use "block skipping" * for long strings: hash an initial part and then sample the rest of * the string with reasonably sized chunks. An initial offset for the * sampling is computed based on a hash of the initial part of the string; * this is done to (usually) avoid the case where all long strings have * certain offset ranges which are never sampled. * * Skip should depend on length and bound the total time to roughly * logarithmic. With current values: * * 1M string => 256 * 241 = 61696 bytes (0.06M) of hashing * 1G string => 256 * 16321 = 4178176 bytes (3.98M) of hashing * * XXX: It would be better to compute the skip offset more "smoothly" * instead of having a few boundary values.
label: code-design
29261. [key setter this val key] -> [key retval]
29262. point to start of 'reserved area'
29263. Shared helper for match() steps 3-4, search() steps 3-4.
29264. dummy
29265. Just call the "original" Object.defineProperties() to * finish up.
29266. Although these could be parsed as PatternCharacters unambiguously (here), * E5 Section 15.10.1 grammar explicitly forbids these as PatternCharacters.
29267. line terminator will be handled on next round
29268. * Debug dumping
29269. x in [2**31, 2**32[
29270. Error path.
29271. if a site already exists, nop: max one label site per statement
29272. Flag ORed to err_code to indicate __FILE__ / __LINE__ is not * blamed as source of error for error fileName / lineNumber.
29273. Start in paused state.
29274. **comment:** XXX: much to improve (code size)
label: code-design
29275. [regexp string res_arr]
29276. range conversion with a "skip"
29277. **comment:** Update function min/max line from current token. Needed to improve * function line range information for debugging, so that e.g. opening * curly brace is covered by line range even when no opcodes are emitted * for the line containing the brace.
label: code-design
29278. * Regular expression structs, constants, and bytecode defines.
29279. **comment:** should never be zero, because we (Duktape.Thread.yield) are on the stack
label: code-design
29280. **comment:** XXX: This will now return false for non-numbers, even though they would * coerce to NaN (as a general rule). In particular, duk_get_number() * returns a NaN for non-numbers, so should this function also return * true for non-numbers?
label: code-design
29281. **comment:** The handler is looked up with a normal property lookup; it may be an * accessor or the handler object itself may be a proxy object. If the * handler is a proxy, we need to extend the valstack as we make a * recursive proxy check without a function call in between (in fact * there is no limit to the potential recursion here). * * (For sanity, proxy creation rejects another proxy object as either * the handler or the target at the moment so recursive proxy cases * are not realized now.)
label: code-design
29282. Flag handling currently assumes that flags are consistent. This is OK * because the call sites are now strictly controlled.
29283. DUK_TOK_DEFAULT
29284. [key value] or [key undefined]
29285. Pad significand with "virtual" zero digits so that Dragon4 will * have enough (apparent) precision to work with.
29286. * Set lexer input position and reinitialize lookup window.
29287. nargs
29288. With lightfuncs, act 'func' may be NULL
29289. return (?)
29290. DUK_USE_REFZERO_FINALIZER_TORTURE
29291. next to use, highest used is top - 1
29292. **comment:** XXX: actually single step levels would work just fine, clean up
label: code-design
29293. * DUK_CALL_FLAG_IGNORE_RECLIMIT causes duk_handle_call() to ignore C * recursion depth limit (and won't increase it either). This is * dangerous, but useful because it allows the error handler to run * even if the original error is caused by C recursion depth limit. * * The heap level DUK_HEAP_FLAG_ERRHANDLER_RUNNING is set for the * duration of the error handler and cleared afterwards. This flag * prevents the error handler from running recursively. The flag is * heap level so that the flag properly controls even coroutines * launched by an error handler. Since the flag is heap level, it is * critical to restore it correctly. * * We ignore errors now: a success return and an error value both * replace the original error value. (This would be easy to change.)
29294. **comment:** XXX: remove heap->dbg_exec_counter, use heap->inst_count_interrupt instead?
label: code-design
29295. DUK_USE_JSON_DECSTRING_FASTPATH
29296. The actual detached_cb call is postponed to message loop so * we don't need any special precautions here (just skip to EOM * on the already closed connection).
29297. arg2 would be clobbered so reassign it to a temp.
29298. standard behavior, step 3.f.i
29299. avoid degenerate cases, so that (len - 1) won't underflow
29300. Note: %06d for positive value, %07d for negative value to include * sign and 6 digits.
29301. 'let'
29302. * Halt execution helper
29303. 1110 xxxx; 3 bytes
29304. ch1 = (r_increment << 8) + byte
29305. variable name reg/const, if variable not register-bound
29306. Cast converts magic to 16-bit signed value
29307. * Remove the object from the refzero list. This cannot be done * before a possible finalizer has been executed; the finalizer * may trigger a mark-and-sweep, and mark-and-sweep must be able * to traverse a complete refzero_list.
29308. Argument variants. When the argument is an ArrayBuffer a view to * the same buffer is created; otherwise a new ArrayBuffer is always * created.
29309. entry part size
29310. * Property not found in prototype chain.
29311. function executes in strict mode
29312. 'data' is reachable through every compiled function which * contains a reference.
29313. Caller must trigger recomputation of active breakpoint list. To * ensure stale values are not used if that doesn't happen, clear the * active breakpoint list here.
29314. * Longjmp types, also double as identifying continuation type for a rethrow (in 'finally')
29315. note: long live range

29316. * Object will be kept; queue object back to heap_allocated (to tail)
29317. no change
29318. DUK_USE_ASSERTIONS
29319. force the property to 'undefined' to create a slot for it
29320. Optimized for speed.
29321. **comment:** prevent multiple in-progress detaches
 label: requirement
29322. Set a high interrupt counter; the original executor * interrupt invocation will rewrite before exiting.
29323. DUK_TOK_THIS
29324. index
29325. * Expression parsing: duk__expr_nud(), duk__expr_led(), duk__expr_lbp(), and helpers. * * - duk__expr_nud(): ("null denotation"): process prev_token as a "start" of an expression (e.g. literal) * - duk__expr_led(): ("left denotation"): process prev_token in the "middle" of an expression (e.g. operator) * - duk__expr_lbp(): ("left-binding power"): return left-binding power of curr_token
29326. 3-letter log level strings
29327. **comment:** * Convert char offset to byte offset * * Avoid using the string cache if possible: for ASCII strings byte and * char offsets are equal and for short strings direct scanning may be * better than using the string cache (which may evict a more important * entry). * * Typing now assumes 32-bit string byte/char offsets (duk_uint_fast32_t). * Better typing might be to use duk_size_t.
 label: code-design
29328. Must prevent finalizers which may have arbitrary side effects.
29329. DUK_EXCEPTION_H_INCLUDED
29330. invalidates h_buf pointer
29331. Special case: original qmin was zero so there is nothing * to repeat. Emit an atom copy but jump over it here.
29332. **comment:** Higher footprint, less churn.
 label: code-design
29333. -> [... holder name val new_elem]
29334. NUL term or -1 (EOF), NUL check would suffice
29335. core property functions
29336. 25%, i.e. less than 25% used -> abandon
29337. Allow empty fraction (e.g. "123.")
29338. 0x0...0xff
29339. don't allow const
29340. this is just beneath bottom
29341. * Duktape.Buffer: constructor
29342. 'Uint8Array'
29343. * Helpers for writing multiple properties
29344. exhaustive
29345. refcounting requires direct heap frees, which in turn requires a dual linked heap
29346. lastIndexOf() needs to be a vararg function because we must distinguish * between an undefined fromIndex and a "not given" fromIndex; indexOf() is * made vararg for symmetry although it doesn't strictly need to be.
29347. [... error]
29348. XXX: share final setting code for value and flags? difficult because * refcount code is different. Share entry allocation? But can't allocate * until array index checked.
29349. Terminating conditions. For fixed width output, we just ignore the * terminating conditions (and pretend that tc1 == tc2 == false). The * the current shortcut for fixed-format output is to generate a few * extra digits and use rounding (with carry) to finish the output.
29350. Return value when returning to this activation (points to caller * reg, not callee reg); index is absolute (only set if activation is * not topmost). * * Note: idx_bottom is always set, while idx_retval is only applicable * for activations below the topmost one. Currently idx_retval for * the topmost activation is considered garbage (and it not initialized * on entry or cleared on return; may contain previous or garbage * values).
29351. Unlike non-obsolete String calls, substr() algorithm in E5.1 * specification will happily coerce undefined and null to strings * ("undefined" and "null").
29352. DecimalEscape, only \0 is allowed, no leading zeroes are allowed
29353. parse new token
29354. 1e9
29355. Negative value checked so that a "time jump" works * reasonably. * * Same interval is now used for status sending and * peeking.
29356. * Misc shared helpers.
29357. don't set 'n' at all, inherited value is used as name
29358. ;'
29359. switch cmd
29360. **comment:** XXX: could add a fast path to process chunks of input codepoints, * but relative benefit would be quite small.
 label: code-design
29361. Copy interrupt counter/init value state to new thread (if any). * It's OK for new_thr to be the same as curr_thr.
29362. User totalLength overrides a computed length, but we'll check * every copy in the copy loop. Note that duk_to_uint() can * technically have arbitrary side effects so we need to recheck * the buffers in the copy loop.
29363. **comment:** XXX: some overlapping code; cleanup
 label: code-design
29364. stack[0] = searchElement * stack[1] = fromIndex * stack[2] = object * stack[3] = length (not needed, but not popped above)
29365. from curr pc
29366. * Maximum size check
29367. isError flag for yield
29368. * callstack_top - 1 --> this function * callstack_top - 2 --> caller (may not exist) * * If called directly from C, callstack_top might be 1. If calling * activation doesn't exist, call must be indirect.
29369. no need to unwind callstack
29370. * Array part
29371. **comment:** * Number should already be in NaN-normalized form, * but let's normalize anyway.
 label: code-design
29372. [array totalLength bufres buf]
29373. When src_size == 0, src_data may be NULL (if source * buffer is dynamic), and dst_data may be NULL (if * target buffer is dynamic). Avoid zero-size memcpy() * with an invalid pointer.
29374. [r1,r2] is the range
29375. * NormalizePropertyDescriptor() related helper. * * Internal helper which validates and normalizes a property descriptor * represented as an EcmaScript object (e.g. argument to defineProperty()). * The output of this conversion is a set of defprop_flags and possibly * some values pushed on the value stack; some subset of: property value, * getter, setter. Caller must manage stack top carefully because the * number of values pushed depends on the input property descriptor. * * The original descriptor object must not be altered in the process.
29376. duk_handle_call / duk_handle_safe_call recursion depth limiting
29377. DUK_HOBJECT_FLAG_NEWENV: handled below
29378. A leading digit is not required in some cases, e.g. accept ".123". * In other cases (JSON.parse()) a leading digit is required. This * is checked for after the loop.
29379. common case, already closed, so skip
29380. ignore millisecond fractions after 3
29381. * unshift()
29382. E5 Section 10.4.3

29383. No built-in functions are constructable except the top * level ones (Number, etc).

29384. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed, * and we're in an infinite loop.

29385. Raw helper for getting a value from the stack, checking its tag. * The tag cannot be a number because numbers don't have an internal * tag in the packed representation.

29386. Bernstein hash init value is normally 5381

29387. * DECLVAR * * See E5 Sections: * 10.4.3 Entering Function Code * 10.5 Declaration Binding Instantiation * 12.2 Variable Statement * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * * Variable declaration behavior is mainly discussed in Section 10.5, * and is not discussed in the execution semantics (Sections 11-13). * * Conceptually declarations happen when code (global, eval, function) * is entered, before any user code is executed. In practice, register- * bound identifiers are 'declared' automatically (by virtue of being * allocated to registers with the initial value 'undefined'). Other * identifiers are declared in the function prologue with this primitive. * * Since non-register bindings eventually back to an internal object's * properties, the 'prop_flags' argument is used to specify binding type: * * - Immutable binding: set DUK_PROPDESC_FLAG_WRITABLE to false * - Non-deletable binding: set DUK_PROPDESC_FLAG_CONFIGURABLE to false * - The flag DUK_PROPDESC_FLAG_ENUMERABLE should be set, although it * doesn't really matter for internal objects * * All bindings are non-deletable mutable bindings except: * * - Declarations in eval code (mutable, deletable) * - 'arguments' binding in strict function code (immutable) * - Function name binding of a function expression (immutable) * * Declarations may go to declarative environment records (always * so for functions), but may also go to object environment records * (e.g. global code). The global object environment has special * behavior when re-declaring a function (but not a variable); see * E5.1 specification, Section 10.5, step 5.e. * * Declarations always go to the 'top-most' environment record, i.e. * we never check the record chain. It's not an error even if a * property (even an immutable or non-deletable one) of the same name * already exists. * * If a declared variable already exists, its value needs to be updated * (if possible). Returns 1 if a PUTVAR needs to be done by the caller; * otherwise returns 0.

29388. **comment:** duk_handle_call_xxx: call ignores C recursion limit (for errhandler calls)

label: code-design

29389. expt 0xffff is infinite/nan

29390. Augment the error if called as a normal function. __FILE__ and __LINE__ * are not desirable in this case.

29391. %s

29392. e.g. "x" < "xx"

29393. **comment:** XXX: for fastints, could use a variant which assumes a double duk_tval * (and doesn't need to check for fastint again).

label: code-design

29394. SameValue

29395. How much stack to require on entry to object/array encode

29396. * Parse code after the clause. Possible terminators are * 'case', 'default', and '}'. * * Note that there may be no code at all, not even an empty statement, * between case clauses. This must be handled just like an empty statement * (omitting seemingly pointless JUMPs), to avoid situations like * test-bug-case-fallthrough.js.

29397. ... name ': ' message

29398. to exit

29399. parenthesis count, 0 = top level

29400. * Useful Unicode codepoints * * Integer constants must be signed to avoid unexpected coercions * in comparisons.

29401. input string (may be a user pointer)

29402. * round_idx points to the digit which is considered for rounding; the * digit to its left is the final digit of the rounded value. If round_idx * is zero, rounding will be performed; the result will either be an empty * rounded value or if carry happens a '1' digit is generated.

29403. Careful with carry condition: * - If carry not added: 0x12345678 + 0 + 0xffffffff = 0x12345677 (< 0x12345678) * - If carry added: 0x12345678 + 1 + 0xffffffff = 0x12345678 (== 0x12345678)

29404. shared helper

29405. * Fast buffer writer with spare management.

29406. success/error path both do this

29407. DUK_TOK_MUL

29408. attempt to change from accessor to data property

29409. allowed ascii whitespace

29410. [... source? filename?]

29411. continue jump

29412. -> [... key val]

29413. Prepare value stack for a method call through an object property. * May currently throw an error e.g. when getting the property.

29414. no need to create environment record now; leave as NULL

29415. * Thread state check and book-keeping.

29416. "/=" and not in regexp mode

29417. for lastIndexOf, result may be -1 (mark immediate termination)

29418. fatal_func should be noreturn, but noreturn declarations on function * pointers has a very spotty support apparently so it's not currently * done.

29419. requested identifier

29420. Function expression. Note that any statement beginning with 'function' * is handled by the statement parser as a function declaration, or a * non-standard function expression/statement (or a SyntaxError). We only * handle actual function expressions (occurring inside an expression) here. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().

29421. * JSON.stringify() fast path * * Otherwise supports full JSON, JX, and JC features, but bails out on any * possible side effect which might change the value being serialized. The * fast path can take advantage of the fact that the value being serialized * is unchanged so that we can walk directly through property tables etc.

29422. **comment:** XXX: helper

label: code-design

29423. no return value -> don't replace created value

29424. 'Arguments' object and has arguments exotic behavior (non-strict callee)

29425. [requested_id require require.id resolved_id last_comp Duktape.modLoaded Duktape.modLoaded[id]]

29426. Note: left shift, should mask

29427. 31 to 36

29428. two level break

29429. don't allow an empty match at the end of the string

29430. 'thr' is now reachable

29431. Parse argument list. Arguments are written to temps starting from * "next temp". Returns number of arguments parsed. Expects left paren * to be already eaten, and eats the right paren before returning.

29432. Allow 'Infinity'

29433. Opcode slot C is used in a non-standard way, so shuffling * is not allowed.

29434. * Helpers for managing property storage size

29435. if-digit-else-ctrl

29436. Safe write calls which will ensure space first.

29437. having this as a separate function provided a size benefit

29438. code emission

29439. **comment:** * obj->props is intentionally left as NULL, and duk_hobject_props.c must deal * with this properly. This is intentional: empty objects consume a minimum * amount of memory. Further, an initial allocation might fail and cause * 'obj' to "leak" (require a mark-and-sweep) since it is not reachable yet.

label: code-design

29440. Term was '.', backtrack resolved name by one component. * q[-1] = previous slash (or beyond start of buffer) * q[-2] = last char of previous component (or beyond start of buffer)

29441. **comment:** This could also be thrown internally (set the error, goto check_longjmp), * but it's better for internal errors to bubble outwards so that we won't * infinite loop in this catchpoint.

label: code-design

29442. * Debugging related API calls
29443. -> [... func this arg1 ... argN _Args length]
29444. 'Float64Array'
29445. 1 1 1 <32 bits> * Encode in two parts to avoid bitencode 24-bit limitation
29446. -> [this]
29447. **comment:** Note: array entries are always writable, so the writability check * above is pointless for them. The check could be avoided with some * refactoring but is probably not worth it.
label: code-design
29448. retval indicates delete failed
29449. formatters always get one-based month/day-of-month
29450. continue matching, set neg_tzoffset flag
29451. bound function chain has already been resolved
29452. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property get right now.
29453. standard JSON; array gaps
29454. Note: array part values are [[Writable]], [[Enumerable]], * and [[Configurable]] which matches the required attributes * here.
29455. 0=before period/exp, * 1=after period, before exp * 2=after exp, allow '+' or '-' * 3=after exp and exp sign
29456. just check 'env'
29457. **comment:** XXX: primitive to make array from valstack slice
label: code-design
29458. **comment:** * Dragon4 setup. * * Convert double from IEEE representation for conversion; * normal finite values have an implicit leading 1-bit. The * slow path algorithm doesn't handle zero, so zero is special * cased here but still creates a valid nc_ctx, and goes * through normal formatting in case special formatting has * been requested (e.g. forced exponential format: 0 -> "0e+0").
label: code-design
29459. consistency requires this
29460. DUK_TOK_CASE
29461. Nothing to initialize, strsf[] is in ROM.
29462. currently 6 characters of lookup are actually needed (duk_lexer.c)
29463. index/length check guarantees
29464. left.x1 -> res.x1
29465. * Case conversion tables generated using src/extract_caseconv.py.
29466. is global code
29467. **comment:** * Macros to set a duk_tval and update refcount of the target (decref the * old value and incref the new value if necessary). This is both performance * and footprint critical; any changes made should be measured for size/speed.
label: code-design
29468. We don't duk_require_stack() here now, but rely on the caller having * enough space.
29469. * Regexp executor. * * Safety: the Ecmascript executor should prevent user from reading and * replacing regexp bytecode. Even so, the executor must validate all * memory accesses etc. When an invalid access is detected (e.g. a 'save' * opcode to invalid, unallocated index) it should fail with an internal * error but not cause a segmentation fault. * * Notes: * * - Backtrack counts are limited to unsigned 32 bits but should * technically be duk_size_t for strings longer than 4G chars. * This also requires a regexp bytecode change.
29470. Note: not a valid stack index if num_stack_args == 0
29471. * Convenience (independent of representation)
29472. no refcounting
29473. duk_handle_ecma_call_setup: setup for a resume()
29474. escape any non-ASCII characters
29475. if 0, 'str' used, if > 0, 'strlist' used
29476. These limits are based on bytecode limits. Max temps is limited * by duk_hcompiledfunction nargs/nregs fields being 16 bits.
29477. * pop(), push()
29478. 'DecEnv'
29479. number of elements guaranteed to be user accessible * (in addition to call arguments) on Duktape/C function entry.
29480. NOTE: The Array special behaviors are NOT invoked by duk_xdef_prop_index() * (which differs from the official algorithm). If no error is thrown, this * doesn't matter as the length is updated at the end. However, if an error * is thrown, the length will be unset. That shouldn't matter because the * caller won't get a reference to the intermediate value.
29481. Torture option to shake out finalizer side effect issues: * make a bogus function call for every finalizable object, * essentially simulating the case where everything has a * finalizer.
29482. [...] key]
29483. **comment:** This seems faster than emitting bytes one at a time and * then potentially rewinding.
label: code-design
29484. * Writability check
29485. may be changed by call
29486. DUK_USE_NONSTD_FUNC_CALLER_PROPERTY
29487. DUK_USE_JSON_STRINGIFY_FASTPATH
29488. Update all labels with matching label_id.
29489. guaranteed by duk_to_string()
29490. DUK_USE_TRACEBACKS
29491. **comment:** Formatting function pointers is tricky: there is no standard pointer for * function pointers and the size of a function pointer may depend on the * specific pointer type. This helper formats a function pointer based on * its memory layout to get something useful on most platforms.
label: code-design
29492. ensures callstack_top - 1 >= 0
29493. Count free operations toward triggering a GC but never actually trigger * a GC from a free. Otherwise code which frees internal structures would * need to put in NULLs at every turn to ensure the object is always in * consistent state for a mark-and-sweep.
29494. detached
29495. 'constructor'
29496. disabled for now
29497. idx_replacer
29498. type to represent a straight register reference, with <0 indicating none
29499. continue jump not patched, an INVALID opcode remains there
29500. -> [... enum_target res trap_result]
29501. current (next) array index
29502. fmin() with args -0 and +0 is not guaranteed to return * -0 as Ecmascript requires.
29503. 'const'
29504. all codepoints up to U+10FFFF
29505. **comment:** This is performance critical because it's needed for every DECREF. * Take advantage of the fact that the first heap allocated tag is 8, * so that bit 3 is set for all heap allocated tags (and never set for * non-heap-allocated tags).
label: code-design
29506. if no NaN handling flag, may still be NaN here, but not Inf
29507. thr argument only used for thr->heap, so specific thread doesn't matter
29508. Preshifted << 4. Must use 16-bit entry to allow negative value signalling.
29509. [0x00000000, 0xffffffff]

29510. fall through to error
29511. [... key_obj key key]
29512. Specification stripPrefix maps to DUK_S2N_FLAG_ALLOW_AUTO_HEX_INT. ** Don't autodetect octals (from leading zeroes), require user code to * provide an explicit radix 8 for parsing octal. See write-up from Mozilla: * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt#ECMAScript_5_Removes_Octal_Interpretation
29513. Process messages until we're no longer paused or we peek * and see there's nothing to read right now.
29514. XXX: callstack unwind may now throw an error when closing * scopes; this is a sandboxing issue, described in: * <https://github.com/svaarala/duktape/issues/476>
29515. since no recursive error handler calls
29516. **comment:** XXX: other properties of function instances; 'arguments', 'caller'.
 label: code-design
29517. numeric label_id (-1 reserved as marker)
29518. irrelevant when out->value == NULL
29519. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'd.
29520. **comment:** * Helper to sort array index keys. The keys are in the enumeration object * entry part, starting from DUK_ENUM_START_INDEX, and the entry part is dense. ** We use insertion sort because it is simple (leading to compact code,) * works nicely in-place, and minimizes operations if data is already sorted * or nearly sorted (which is a very common case here). It also minimizes * the use of element comparisons in general. This is nice because element * comparisons here involve re-parsing the string keys into numbers each * time, which is naturally very expensive. ** Note that the entry part values are all "true", e.g. * * "1" -> true, "3" -> true, "2" -> true * so it suffices to only work in the key part without exchanging any keys, * simplifying the sort. **
http://en.wikipedia.org/wiki/Insertion_sort * * (Compiles to about 160 bytes now as a stand-alone function.)
 label: code-design
29521. The function object is now reachable and refcounts are fine, * so we can pop off all the temporaries.
29522. Care must be taken to avoid pointer wrapping in the index * validation. For instance, on a 32-bit platform with 8-byte * duk_tval the index 0x20000000UL would wrap the memory space * once.
29523. Convert a duk_tval fastint (caller checks) to a 32-bit index.
29524. Insert bytes in the middle of the buffer from a slice already * in the buffer. Source offset is interpreted "before" the operation.
29525. DUK_USE_STRTAB_CHAIN
29526. Original function instance/template had NAMEBINDING. * Must create a lexical environment on loading to allow * recursive functions like 'function foo() { foo(); }'.
29527. **comment:** the message should be a compile time constant without formatting (less risk); * we don't care about assertion text size because they're not used in production * builds.
 label: code-design
29528. assume value is a number
29529. **comment:** combines steps 3, 6; step 7 is not needed
 label: requirement
29530. * Debugger support
29531. **comment:** XXX: spare handling, slow now
 label: code-design
29532. normal and constructor calls have identical semantics
29533. * A token is interpreted as any possible production of InputElementDiv * and InputElementRegExp, see E5 Section 7 in its entirety. Note that * the E5 "Token" production does not cover all actual tokens of the * language (which is explicitly stated in the specification, Section 7.5). * Null and boolean literals are defined as part of both ReservedWord * (E5 Section 7.6.1) and Literal (E5 Section 7.8) productions. Here, * null and boolean values have literal tokens, and are not reserved * words. ** Decimal literal negative/positive sign is -not- part of DUK_TOK_NUMBER. * The number tokens always have a non-negative value. The unary minus * operator in "-1.0" is optimized during compilation to yield a single * negative constant. ** Token numbering is free except that reserved words are required to be * in a continuous range and in a particular order. See genstrings.py.
29534. Same coercion behavior as for Number.
29535. the outer loop will recheck and exit
29536. **comment:** These are macros for now, but could be separate functions to reduce code * footprint (check call site count before refactoring).
 label: code-design
29537. x == -Infinity
29538. [... source? func_template]
29539. other call
29540. lastIndex already set up for next match
29541. * Macros to access the 'props' allocation.
29542. **comment:** * Struct size/alignment if platform requires it * * There are some compiler specific struct padding pragmas etc in use, this * selftest ensures they're correctly detected and used.
 label: code-design
29543. [... source? filename?] (depends on flags)
29544. ToUint16() coercion is mandatory in the E5.1 specification, but * this non-compliant behavior makes more sense because we support * non-BMP codepoints. Don't use CESU-8 because that'd create * surrogate pairs.
29545. match string
29546. [targetBuffer targetStart sourceStart sourceEnd]
29547. x = sign(x) * floor(abs(x)), i.e. truncate towards zero, keep sign
29548. **comment:** XXX: here again finalizer thread is the heap_thread which needs * to be coordinated with finalizer thread fixes.
 label: code-design
29549. at least one opcode emitted
29550. Slice offsets are element (not byte) offsets, which only matters * for TypedArray views, Node.js Buffer and ArrayBuffer have shift * zero so byte and element offsets are the same. Negative indices * are counted from end of slice, crossed indices are allowed (and * result in zero length result), and final values are clamped * against the current slice. There's intentionally no check * against the underlying buffer here.
29551. side effects, in theory (referenced by global env)
29552. * Object.seal() and Object.freeze() (E5 Sections 15.2.3.8 and 15.2.3.9) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. ** Note: virtual (non-concrete) properties which are non-configurable but * writable would pose some problems, but such properties do not currently * exist (all virtual properties are non-configurable and non-writable). * If they did exist, the non-configurability does NOT prevent them from * becoming non-writable. However, this change should be recorded somehow * so that it would turn up (e.g. when getting the property descriptor), * requiring some additional flags in the object.
29553. additional flags which are passed in the same flags argument as property * flags but are not stored in object properties.
29554. **comment:** XXX: Is this INCREF necessary? 'func' is always a borrowed * reference reachable through the value stack? If changed, stack * unwind code also needs to be fixed to match.
 label: code-design
29555. **comment:** XXX: shared decoding of 'b' and 'c'?
 label: code-design
29556. **comment:** The fast path for array property put is not fully compliant: * If one places conflicting number-indexed properties into * Array.prototype (for example, a non-writable Array.prototype[7]) * the fast path will incorrectly ignore them. ** This fast path could be made compliant by falling through * to the slow path if the previous value was UNUSED. This would * also remove the need to check for extensibility. Right now a * non-extensible array is slower than an extensible one as far * as writes are concerned. ** The fast path behavior is documented in more detail here: * tests/ecmascript/test-misc-array-fast-write.js
 label: code-design
29557. **comment:** XXX: because we unwind stacks above, thr->heap->curr_thread is at * risk of pointing to an already freed thread. This was indeed the * case in test-bug-multithread-valgrind.c, until duk_handle_call() * was fixed to restore thr->heap->curr_thread before rethrowing an * uncaught error.
 label: code-design

29558. Allow naked fraction (e.g. ".123")
29559. * Rate limit check for sending status update or peeking into * the debug transport. Both can be expensive operations that * we don't want to do on every opcode. *
* Making sure the interval remains reasonable on a wide variety * of targets and bytecode is difficult without a timestamp, so * we use a Date-provided timestamp
for the rate limit check. * But since it's also expensive to get a timestamp, a bytecode * counter is used to rate limit getting timestamps.
29560. >>> struct.unpack('>d', '4000112233445566'.decode('hex')) * (2.008366013071895.)
29561. Other longjmp types are handled by executor before propagating * the error here.
29562. * E5 Section 11.8.6 describes the main algorithm, which uses * [[HasInstance]]. [[HasInstance]] is defined for only * function objects: * * - Normal functions:
E5 Section 15.3.5.3 * - Functions established with Function.prototype.bind(): * E5 Section 15.3.4.5.3 ** For other objects, a TypeError is thrown. * * Limited
Proxy support: don't support 'getPrototypeOf' trap but * continue lookup in Proxy target if the value is a Proxy.
29563. * Ecmascript/native function call or lightfunc call
29564. Halt execution and enter a debugger message loop until execution is resumed * by the client. PC for the current activation may be temporarily decremented * so
that the "current" instruction will be shown by the client. This helper * is callable from anywhere, also outside bytecode executor.
29565. computed live
29566. **comment:** Load a function from bytecode. The function object returned here must * match what is created by duk_js_push_closure() with respect to its flags, *
properties, etc. * * NOTE: there are intentionally no input buffer length / bound checks. * Adding them would be easy but wouldn't ensure memory safety as
untrusted * or broken bytecode is unsafe during execution unless the opcodes themselves * are validated (which is quite complex, especially for indirect opcodes).
label: code-design
29567. * For an external string, the NUL-terminated string data is stored * externally. The user must guarantee that data behind this pointer * doesn't change while it's
used.
29568. Note: masking of 'x' is not necessary because of * range check and shifting -> no bits overlapping * the marker should be set.
29569. **comment:** * Object.defineProperty() related helper (E5 Section 15.2.3.6) * * Inlines all [[DefineOwnProperty]] exotic behaviors. * * Note: Ecmascript compliant
[[DefineOwnProperty]](P, Desc, Throw) is not * implemented directly, but Object.defineProperty() serves its purpose. * We don't need the [[DefineOwnProperty]]
internally and we don't have a * property descriptor with 'missing values' so it's easier to avoid it * entirely. * * Note: this is only called for actual objects, not
primitive values. * This must support virtual properties for full objects (e.g. Strings) * but not for plain values (e.g. strings). Lightfuncs, even though * primitive in
a sense, are treated like objects and accepted as target * values.
label: code-design
29570. callstack limits
29571. A...P
29572. Module table: * - module.exports: initial exports table (may be replaced by user) * - module.id is non-writable and non-configurable, as the CommonJS * spec
suggests this if possible * - module.filename: not set, defaults to resolved ID if not explicitly * set by modSearch() (note capitalization, not .fileName, matches
Node.js) * - module.name: not set, defaults to last component of resolved ID if * not explicitly set by modSearch()
29573. String constructor needs to distinguish between an argument not given at all * vs. given as 'undefined'. We're a vararg function to handle this properly.
29574. Explicit length is only needed if it differs from 'nargs'.
29575. Force restart caused by a function return; must recheck * debugger breakpoints before checking line transitions, * see GH-303. Restart and then handle
interrupt_counter * zero again.
29576. Decode 'bits' bits from the input stream (bits must be 1...24). * When reading past bitstream end, zeroes are shifted in. The result * is signed to match
duk_bd_decode_flagged.
29577. (?):
29578. Must be restored here to handle e.g. yields properly.
29579. The base ID of the current require() function, resolution base
29580. non-Reference deletion is always 'true', even in strict mode
29581. jump to next loop, using reg_v34_iter as iterated value
29582. Scan error: this shouldn't normally happen; it could happen if * string is not valid UTF-8 data, and clen/blen are not consistent * with the scanning algorithm.
29583. max possible
29584. punctuators (unlike the spec, also includes "/" and "/=")
29585. DUK_BUFOBJ_FLOAT32ARRAY
29586. [...] ptr]
29587. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property delete right now.
29588. 12: getDate
29589. * Helper for valstack space * * Caller of DUK_ASSERT_VALSTACK_SPACE() estimates the number of free stack entries * required for its own use, and any
child calls which are not (a) Duktape API calls * or (b) Duktape calls which involve extending the valstack (e.g. getter call).
29590. 'modLoaded'
29591. * Assertions before
29592. **comment:** * Buffer writer (dynamic buffer only) * * Helper for writing to a dynamic buffer with a concept of a "spare" area * to reduce resizes. You can ensure
there is enough space beforehand and * then write for a while without further checks, relying on a stable data * pointer. Spare handling is automatic so call sites
only indicate how * much data they need right now. * * There are several ways to write using bufwriter. The best approach * depends mainly on how much
performance matters over code footprint. * The key issues are (1) ensuring there is space and (2) keeping the * pointers consistent. Fast code should ensure space
for multiple writes * with one ensure call. Fastest inner loop code can temporarily borrow * the 'p' pointer but must write it back eventually. * * Be careful to
ensure all macro arguments (other than static pointers like * 'thr' and 'bw_ctx') are evaluated exactly once, using temporaries if * necessary (if that's not possible,
there should be a note near the macro). * Buffer write arguments often contain arithmetic etc so this is * particularly important here.
label: code-design
29593. Ordering should not matter (E5 Section 11.8.5, step 3.a) but * preserve it just in case.
29594. for zero size, don't push anything on valstack
29595. Here we'd have the option of decoding unpadded base64 * (e.g. "xxxxyy" instead of "xxxxyy==". Currently not * accepted.
29596. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
29597. * Conceptually, we look for the identifier binding by starting from * 'env' and following to chain of environment records (represented * by the prototype chain). * *
If 'env' is NULL, the current activation does not yet have an * allocated declarative environment record; this should be treated * exactly as if the environment
record existed but had no bindings * other than register bindings. * * Note: we assume that with the DUK_HOBJECT_FLAG_NEWENV cleared * the
environment will always be initialized immediately; hence * a NULL 'env' should only happen with the flag set. This is the * case for: (1) function calls, and (2)
strict, direct eval calls.
29598. heap level "stash" object (e.g., various reachability roots)
29599. * For global or eval code this is straightforward. For functions * created with the Function constructor we only get the source for * the body and must manufacture
the "function ..." part. * * For instance, for constructed functions (v8): * * > a = new Function("foo", "bar", "print(foo)"); * [Function] * > a.toString() * function
anonymous(foo,bar) {\nprint(foo)\n} ** Similarly for e.g. getters (v8): * * > x = { get a(foo,bar) { print(foo); } } * { a: [Getter] } * >
Object.getOwnPropertyDescriptor(x, 'a').get.toString() * 'function a(foo,bar) { print(foo); }'
29600. c recursion check
29601. XXX: optimize by adding the token numbers directly into the * always interned duk_hstring objects (there should be enough * flag bits free for that)?
29602. Continue parsing after padding, allows concatenated, * padded base64.
29603. Check that 'this' is a duk_hbufferobject and return a pointer to it * (NULL if not).
29604. idx_step is +1 for indexOf, -1 for lastIndexOf
29605. is a setter/getter
29606. env created above to stack top
29607. cannot be e.g. arguments exotic, since exotic 'traits' are mutually exclusive
29608. DUK_TOK_LNOT
29609. XXX: sufficient to check 'strict', assert for 'is function'
29610. check final character validity
29611. * Init lexer context
29612. * Inref and decref functions. * * Decref may trigger immediate refzero handling, which may free and finalize * an arbitrary number of objects. *

29613. Generate pc2line data for an instruction sequence, leaving a buffer on stack top.
29614. a dummy undefined value is pushed to make valstack * behavior uniform for caller
29615. idx_step is +1 for reduce, -1 for reduceRight
29616. PatternCharacter, all excluded characters are matched by cases above
29617. Note: -nargs alone would fail for nargs == 0, this is OK
29618. curr and desc are accessors
29619. * Throw the error in the resumed thread's context; the * error value is pushed onto the resumee valstack. ** Note: the callstack of the target may be empty in this case * too (i.e. the target thread has never been resumed). The * value stack will contain the initial function in that case, * which we simply ignore.
29620. For string-to-number, pretend we never have the lowest mantissa as there * is no natural "precision" for inputs. Having lowest_mantissa == 0, we'll * fall into the base cases for both e >= 0 and e < 0.
29621. Number built-in accepts a plain number or a Number object (whose * internal value is operated on). Other types cause TypeError.
29622. Encode to extended UTF-8; 'out' must have space for at least * DUK_UNICODE_MAX_XUTF8_LENGTH bytes. Allows encoding of any * 32-bit (unsigned) codepoint.
29623. func.prototype.constructor = func
29624. Accept any pointer-like value; for 'object' dvalue, read * and ignore the class number.
29625. **comment:** Clone the properties of the ROM-based global object to create a * fully RAM-based global object. Uses more memory than the inherit * model but more compliant.
label: code-design
29626. prev value can be garbage, no decref
29627. **comment:** * For/for-in statement is complicated to parse because * determining the statement type (three-part for vs. a * for-in) requires potential backtracking. ** See the helper for the messy stuff.
label: code-design
29628. Stepped? Step out is handled by callstack unwind.
29629. Copy values, the copy method depends on the arguments. ** Copy mode decision may depend on the validity of the underlying * buffer of the source argument; there must be no harmful side effects * from there to here for copy_mode to still be valid.
29630. fills window
29631. Caller must check character offset to be inside the string.
29632. valstack will be unbalanced, which is OK
29633. conservative
29634. unconditional
29635. DUK_USE_FASTINT && DUK_USE_PACKED_TVAL
29636. * NEXTEMUM checks whether the enumerator still has unenumerated * keys. If so, the next key is loaded to the target register * and the next instruction is skipped. Otherwise the next instruction * will be executed, jumping out of the enumeration loop.
29637. **comment:** Providing access to e.g. act->lex_env would be dangerous: these * internal structures must never be accessible to the application. * Duktape relies on them having consistent data, and this consistency * is only asserted for, not checked for.
label: code-design
29638. Duktape.modSearch
29639. Set .buffer to the argument ArrayBuffer.
29640. * Parse a function-like expression: ** - function expression * - function declaration * - function statement (non-standard) * - setter/getter ** Adds the function to comp_ctx->curr_func function table and returns the * function number. ** On entry, curr_token points to: ** - the token after 'function' for function expression/declaration/statement * - the token after 'set' or 'get' for setter/getter
29641. * Match loop. ** Try matching at different offsets until match found or input exhausted.
29642. flags: ""
29643. -> [res_obj]
29644. unvalidated
29645. nbytes * <-----> * [... | p | x | x | q] * => [... | q | p | x | x]
29646. eat '{' on entry
29647. This native function is also used for Date.prototype.getTime() * as their behavior is identical.
29648. resume write to target
29649. if new_size < L * old_size, resize without abandon check; L = 3-bit fixed point, e.g. 9 -> 9/8 = 112.5%
29650. Note: we could try to stuff a partial hash (e.g. 16 bits) into the * shared heap header. Good hashing needs more hash bits though.
29651. Else functionality is identical for function call and constructor * call.
29652. inclusive
29653. Check that 'this' is a duk_hbufferobject and return a pointer to it.
29654. utf-8 continuation bytes have the form 10xx xxxx
29655. This is based on V8 EquivalentYear() algorithm (see src/genequivyear.py): * <http://code.google.com/p/v8/source/browse/trunk/src/date.h#146>
29656. 'buf' contains the string to write, 'sz_buf' contains the length * (which may be zero).
29657. '-'
29658. hobject management functions
29659. Convert day from one-based to zero-based (internal). This may * cause the day part to be negative, which is OK.
29660. 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx [26 bits]
29661. There is no need to check the first character specially here * (i.e. reject digits): the caller only accepts valid initial * characters and won't call us if the first character is a digit. * This also ensures that the plain string won't be empty.
29662. '\xffBytecode'
29663. Catch a double-to-int64 cast issue encountered in practice.
29664. * DumpHeap command
29665. **comment:** * Lexer for source files, ToNumber() string conversions, RegExp expressions, * and JSON. ** Provides a stream of Ecmascript tokens from an UTF-8/CESU-8 buffer. The * caller can also rewind the token stream into a certain position which is * needed by the compiler part for multi-pass scanning. Tokens are * represented as duk_token structures, and contain line number information. * Token types are identified with DUK_TOK_* defines. ** Characters are decoded into a fixed size lookup window consisting of * decoded Unicode code points, with window positions past the end of the * input filled with an invalid codepoint (-1). The tokenizer can thus * perform multiple character lookups efficiently and with few sanity * checks (such as access outside the end of the input), which keeps the * tokenization code small at the cost of performance. ** Character data in tokens, such as identifier names and string literals, * is encoded into CESU-8 format on-the-fly while parsing the token in * question. The string data is made reachable to garbage collection by * placing the token-related values in value stack entries allocated for * this purpose by the caller. The characters exist in Unicode code point * form only in the fixed size lookup window, which keeps character data * expansion (of especially ASCII data) low. ** Token parsing supports the full range of Unicode characters as described * in the E5 specification. Parsing has been optimized for ASCII characters * because ordinary Ecmascript code consists almost entirely of ASCII * characters. Matching of complex Unicode codepoint sets (such as in the * IdentifierStart and IdentifierPart productions) is optimized for size, * and is done using a linear scan of a bit-packed list of ranges. This is * very slow, but should never be entered unless the source code actually * contains Unicode characters. ** Ecmascript tokenization is partially context sensitive. First, * additional future reserved words are recognized in strict mode (see E5 * Section 7.6.1.2). Second, a forward slash character ('/') can be * recognized either as starting a RegExp literal or as a division operator, * depending on context. The caller must provide necessary context flags * when requesting a new token. ** Future work: * * * Make line number tracking optional, as it consumes space. * * * Add a feature flag for disabling UTF-8 decoding of input, as most * source code is ASCII. Because of Unicode escapes written in ASCII, * this does not allow Unicode support to be removed from e.g. * duk_unicode_is_identifier_start() nor does it allow removal of CESU-8 * encoding of e.g. string literals. * * * Add a feature flag for disabling Unicode compliance of e.g. identifier * names. This allows for a build more than a kilobyte smaller, because * Unicode ranges needed by duk_unicode_is_identifier_start() and * duk_unicode_is_identifier_part() can be dropped. String literals * should still be allowed to contain escaped Unicode, so this still does * not allow removal of CESU-8 encoding of e.g. string literals. * * * Character lookup tables for codepoints above BMP could be stripped. * * * Strictly speaking, E5 specification requires that source code consists * of 16-bit code units, and if not, must be conceptually converted to * that format first. The current lexer processes Unicode code points * and allows characters outside the BMP. These should be converted to * surrogate pairs while reading the source characters into the window, * not after tokens have been formed (as is done now).

However, the fix * is not trivial because two characters are decoded from one codepoint. *** Optimize for speed as well as size. Large if-else ladders are (at * least potentially) slow.

label: code-design

29666. [... buf loop]

29667. **comment:** * Heap object representation. ** Heap objects are used for Ecmascript objects, arrays, and functions, * but also for internal control like declarative and object environment * records. Compiled functions, native functions, and threads are also * objects but with an extended C struct. ** Objects provide the required Ecmascript semantics and exotic behaviors * especially for property access. ** Properties are stored in three conceptual parts: * 1. A linear 'entry part' contains ordered key-value-attributes triples * and is the main method of string properties. * 2. An optional linear 'array part' is used for array objects to store a * (dense) range of [0,N[array indexed entries with default attributes * (writable, enumerable, configurable). If the array part would become * sparse or non-default attributes are required, the array part is * abandoned and moved to the 'entry part'. * 3. An optional 'hash part' is used to optimize lookups of the entry * part; it is used only for objects with sufficiently many properties * and can be abandoned without loss of information. ** These three conceptual parts are stored in a single memory allocated area. * This minimizes memory allocation overhead but also means that all three * parts are resized together, and makes property access a bit complicated.

label: code-design

29668. x <- y + z

29669. get array index related to string (or return DUK_HSTRING_NO_ARRAY_INDEX); * avoids helper call if string has no array index value.

29670. [... re_obj input bc]

29671. Keep the error as the result (coercing it might fail below, * but we don't catch that now).

29672. * Note: although arguments object variable mappings are only established * for non-strict functions (and a call to a non-strict function created * the arguments object in question), an inner strict function may be doing * the actual property write. Hence the throw_flag applied here comes from * the property write call.

29673. push before advancing to keep reachable

29674. * duk_hthread allocation and freeing.

29675. [... pattern flags escaped_source bytecode]

29676. * Helper for updating callee 'caller' property.

29677. eat 'do'

29678. saves a few instructions to have this wrapper (see comment on duk_heap_mem_alloc)

29679. [source template result]

29680. 25 user flags

29681. also stash it before constructor, * in case we need it (as the fallback value)

29682. omit print

29683. Unary plus is used to force a fastint check, so must include * downgrade check.

29684. DUK_FLD_16BIT

29685. **comment:** should never happen, but be robust

label: code-design

29686. _Formals: omitted if function is guaranteed not to need a (non-strict) arguments object

29687. **comment:** XXX: comes out as signed now

label: code-design

29688. flags

29689. **comment:** If message is undefined, the own property 'message' is not set at * all to save property space. An empty message is inherited anyway.

label: code-design

29690. writable but not deletable

29691. Decode UTF-8 codepoint from a sequence of hex escapes. The * first byte of the sequence has been decoded to 't'. ** Note that UTF-8 validation must be strict according to the * specification: E5.1 Section 15.1.3, decode algorithm step * 4.d.vii.8. URIError from non-shortest encodings is also * specifically noted in the spec.

29692. [... formals]

29693. **comment:** avoid tag 0xffff0, no risk of confusion with negative infinity

label: code-design

29694. See: tests/ecmascript/test-spec-bound-constructor.js

29695. guaranteed to succeed

29696. * Define new property * * Note: this may fail if the holder is not extensible.

29697. used elements (includes DELETED)

29698. **comment:** XXX: since the enumerator may be a memory expensive object, * perhaps clear it explicitly here? If so, break jump must * go through this clearing operation.

label: code-design

29699. -> [... key val replacer holder key]

29700. as is

29701. Step 1 is not necessary because duk_call_method() will take * care of it.

29702. * Code emission helpers * * Some emission helpers understand the range of target and source reg/const * values and automatically emit shuffling code if necessary. This is the * case when the slot in question (A, B, C) is used in the standard way and * for opcodes the emission helpers explicitly understand (like DUK_OP_CALL). * * The standard way is that: * - slot A is a target register * - slot B is a source register/constant * - slot C is a source register/constant * * If a slot is used in a non-standard way the caller must indicate this * somehow. If a slot is used as a target instead of a source (or vice * versa), this can be indicated with a flag to trigger proper shuffling * (e.g. DUK_EMIT_FLAG_B_IS_TARGET). If the value in the slot is not * register/const related at all, the caller must ensure that the raw value * fits into the corresponding slot so as to not trigger shuffling. The * caller must set a "no shuffle" flag to ensure compilation fails if * shuffling were to be triggered because of an internal error. * * For slots B and C the raw slot size is 9 bits but one bit is reserved for * the reg/const indicator. To use the full 9-bit range for a raw value, * shuffling must be disabled with the DUK_EMIT_FLAG_NO_SHUFFLE_{B,C} flag. * Shuffling is only done for A, B, and C slots, not the larger BC or ABC slots. * * There is call handling specific understanding in the A-B-C emitter to * convert call setup and call instructions into indirect ones if necessary.

29703. **comment:** XXX: make this an internal helper

label: code-design

29704. assumed to bottom relative

29705. intentionally empty

29706. Compute time value from (double) parts. The parts can be either UTC * or local time; if local, they need to be (conceptually) converted into * UTC time. The parts may represent valid or invalid time, and may be * wildly out of range (but may cancel each other and still come out in * the valid Date range).

29707. aaaaabbbccccccdddddd

29708. [[SetPrototypeOf]] standard behavior, E6 9.1.2

29709. **comment:** XXX: could unwind catchstack here, so that call handling * didn't need to do that?

label: code-design

29710. 27: setUTCSeconds

29711. Lightfunc virtual properties are non-configurable, so * reject if match any of them.

29712. **comment:** init is unnecessary but suppresses "may be used uninitialized" warnings

label: code-design

29713. no need to unwind catchstack

29714. 'Proxy' object

29715. 'errCreate'

29716. coerce lval with ToString()

29717. see algorithm

29718. * reverse()

29719. index is above internal buffer length -> property is fully normal

29720. ensure never re-entered until rescue cycle complete
29721. **comment:** order: most frequent to least frequent
 label: code-design
29722. **comment:** not very useful, used for debugging
 label: code-design
29723. catch stack depth
29724. [...] res_obj]
29725. * Lexer defines.
29726. A token value. Can be memcpy()'d, but note that slot1/slot2 values are on the valstack. * Some fields (like num, str1, str2) are only valid for specific token types and may have * stale values otherwise.
29727. DUK_BUFOBJ_NODEJS_BUFFER
29728. XXX: if 'len' is low, may want to ensure array part is kept: * the caller is likely to want a dense array.
29729. XXX: version specific array format instead?
29730. * Process incoming debug requests ** Individual request handlers can push temporaries on the value stack and * rely on duk_debug_process_message() to restore the value stack top * automatically.
29731. roughly 2 kB
29732. Must avoid duk_pop() in exit path
29733. basic types from duk_features.h
29734. XXX: byte offset
29735. * instanceof
29736. escript compiler limits
29737. token type
29738. -> [...] key val replacer]
29739. call_flags
29740. * Comparisons (x >= y, x > y, x <= y, x < y) ** E5 Section 11.8.5: implement 'x < y' and then use negate and eval_left_first * flags to get the rest.
29741. Negative indices are always within allocated stack but * must not go below zero index.
29742. Executor interrupt default interval when nothing else requires a * smaller value. The default interval must be small enough to allow * for reasonable execution timeout checking but large enough to keep * impact on execution performance low.
29743. * Object internal value ** Returned value is guaranteed to be reachable / incref'd, caller does not need * to incref OR decref. No proxies or accessors are invoked, no prototype walk.
29744. Exponent
29745. general temp register
29746. 0xe0...0xef
29747. Maximum write size: XUTF8 path writes max DUK_UNICODE_MAX_XUTF8_LENGTH, * percent escape path writes max two times CESU-8 encoded BMP length.
29748. [arg1 ... argN this loggerLevel loggerName buffer]
29749. If not present in finalize_list or refzero_list, the pointer * must be either in heap_allocated or the string table.
29750. rc_varname and reg_varbind are ignored here
29751. duplicate zeroing, expect for (possible) NULL inits
29752. get as double to handle huge numbers correctly
29753. enum_index
29754. XXX: clarify on when and where DUK_CONST_MARKER is allowed
29755. * Error helpers
29756. * Data following the header depends on the DUK_HBUFFER_FLAG_DYNAMIC * flag. ** If the flag is clear (the buffer is a fixed size one), the buffer * data follows the header directly, consisting of 'size' bytes. ** If the flag is set, the actual buffer is allocated separately, and * a few control fields follow the header. Specifically: ** - a "void *" pointing to the current allocation * - a duk_size_t indicating the full allocated size (always >= 'size') ** If DUK_HBUFFER_FLAG_EXTERNAL is set, the buffer has been allocated * by user code, so that Duktape won't be able to resize it and won't * free it. This allows buffers to point to e.g. an externally * allocated structure such as a frame buffer. ** Unlike strings, no terminator byte (NUL) is guaranteed after the * data. This would be convenient, but would pad aligned user buffers * unnecessarily upwards in size. For instance, if user code requested * a 64-byte dynamic buffer, 65 bytes would actually be allocated which * would then potentially round upwards to perhaps 68 or 72 bytes.
29757. 'new'
29758. evaluate to plain value, no forced register (temp/bound reg both ok)
29759. Keep heap->finalize_list up-to-date during the list walk. * This has no functional impact, but does matter e.g. for * duk_push_heapptr() asserts when assertions are enabled.
29760. DUK_USE_INTERRUPT_COUNTER
29761. Fast path is triggered for no exponent and also for balanced exponent * and fraction parts, e.g. for "1.23e2" == "123". Remember to respect * zero sign.
29762. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property put right now (putprop protects * against it internally).
29763. **comment:** * Parser control values for tokens. The token table is ordered by the * DUK_TOK_XXX defines. ** The binding powers are for lbp() use (i.e. for use in led() context). * Binding powers are positive for typing convenience, and bits at the * top should be reserved for flags. Binding power step must be higher * than 1 so that binding power "lbp - 1" can be used for right associative * operators. Currently a step of 2 is used (which frees one more bit for * flags).
 label: code-design
29764. * Array part ** Note: ordering between array and entry part must match 'abandon array' * behavior in duk_hobject_props.c: key order after an array is abandoned * must be the same.
29765. **comment:** XXX: could be eliminated with valstack adjust
 label: code-design
29766. [key val]
29767. DUK_TOK_CATCH
29768. Note: we rely on duk_safe_call() to fix up the stack for the caller, * so we don't need to pop stuff here. There is no return value; * caller determines rescued status based on object refcount.
29769. * Case conversion helper, with context/local sensitivity. * For proper case conversion, one needs to know the character * and the preceding and following characters, as well as * locale/language.
29770. [...] arg1 ... argN envobj argobj]
29771. flags:nonstrict
29772. pass2 allocation handles this
29773. * Call the wrapped module function. ** Use a protected call so that we can update Duktape.modLoaded[resolved_id] * even if the module throws an error.
29774. advance manually
29775. **comment:** XXX: awkward
 label: code-design
29776. target out-of-bounds (but positive)
29777. free some memory
29778. constrained by string length
29779. stable; precalculated for faster lookups
29780. * Thread defines
29781. same handling for identifiers and strings
29782. -> [...] lval rval rval.prototype]
29783. 'implements'
29784. '='
29785. XXX: Not sure what the best return value would be in the API. * Return a boolean for now. Note that rc == 0 is success (true).

29786. Current convention is to use duk_size_t for value stack sizes and global indices, * and duk_idx_t for local frame indices.
29787. **comment:** XXX: where to release temp regs in intermediate expressions? * e.g. 1+2+3 -> don't inflate temp register count when parsing this. * that particular expression temp regs can be forced here.
label: code-design
29788. negative top-relative indices not allowed now
29789. 'instanceof'
29790. avoid attempt to compact any objects
29791. expression parsing helpers
29792. return ToObject(this)
29793. one token
29794. flags field: LLLLLLFT, L = label (24 bits), F = flags (4 bits), T = type (4 bits)
29795. not protected, respect reclimit, is a constructor call
29796. DUK_TOK_BNOT
29797. E5 Section 8.6.1
29798. indicates a deleted string; any fixed non-NULL, non-hstring pointer works
29799. val is unsigned so >= 0
29800. **comment:** NOTE: we try to minimize code size by avoiding unnecessary pops, * so the stack looks a bit cluttered in this function. DUK_ASSERT_TOP() * assertions are used to ensure stack configuration is correct at each * step.
label: code-design
29801. must have start and end
29802. Check for breakpoints only on line transition. * Breakpoint is triggered when we enter the target * line from a different line, and the previous line * was within the same function. * * This condition is tricky: the condition used to be * that transition to -or across- the breakpoint line * triggered the breakpoint. This seems intuitively * better because it handles breakpoints on lines with * no emitted opcodes; but this leads to the issue * described in:
<https://github.com/svaaraala/duktape/issues/263>.
29803. env and act may be NULL
29804. * Process yield * * After longjmp(), processing continues in bytecode executor longjmp * handler, which will e.g. update thr->resumer to NULL.
29805. -> [... enum key enum_target key]
29806. **comment:** XXX: store 'bcode' pointer to activation for faster lookup?
label: code-design
29807. * Unicode tables
29808. [... obj]
29809. Note: space must cater for both JX and JC.
29810. Needs to be inserted; scan backwards, since we optimize * for the case where elements are nearly in order.
29811. Normal non-bound function.
29812. Eating a left curly; regexp mode is allowed by left curly * based on duk_token_lbp[] automatically.
29813. A validated read() is always a number, so it's write coercion * is always side effect free and won't invalidate pointers etc.
29814. current and previous token for parsing
29815. probe sequence
29816. message is empty -> return name
29817. NaN timevalue: we need to coerce the arguments, but * the resulting internal timestamp needs to remain NaN. * This works but is not pretty: parts and dparts will * be partially uninitialized, but we only write to them.
29818. **comment:** It might seem that replacing 'thr->heap' with just 'heap' below * might be a good idea, but it increases code size slightly * (probably due to unnecessary spilling) at least on x64.
label: code-design
29819. Currently nothing to free; 'data' is a heap object
29820. 'prototype'
29821. 'jx'
29822. default prototype (Note: 'obj' must be reachable)
29823. **comment:** XXX: currently NULL allocations are not supported; remove if later allowed
label: code-design
29824. Unwind the topmost callstack entry before reusing it
29825. internal define property: skip write silently if exists
29826. borrowed, no refcount
29827. allow e.g. '0x0009' and '00077'
29828. DUK_BUFOBJ_DATAVIEW
29829. * Compact an object. Minimizes allocation size for objects which are * not likely to be extended. This is useful for internal and non- * extensible objects, but can also be called for non-extensible objects. * May abandon the array part if it is computed to be too sparse. * * This call is relatively expensive, as it needs to scan both the * entries and the array part. * * The call may fail due to allocation error.
29830. find elements to swap
29831. currently, always the case
29832. * Detected label
29833. **comment:** XXX: an array can have length higher than 32 bits; this is not handled * correctly now.
label: code-design
29834. * Helper macros
29835. **comment:** XXX: Simplify this algorithm, should be possible to come up with * a shorter and faster algorithm by inspecting IEEE representation * directly.
label: code-design
29836. * CheckObjectCoercible() (E5 Section 9.10) * * Note: no API equivalent now.
29837. constants are strings or numbers now
29838. '\n'
29839. Must behave like a no-op with NULL and any pointer returned from * malloc/realloc with zero size.
29840. if (is_regexp)
29841. [thisArg arg1 ... argN func boundFunc]
29842. -Infinity
29843. Undefine local defines
29844. * Top-level include file to be used for all (internal) source files. * * Source files should not include individual header files, as they * have not been designed to be individually included.
29845. * Init built-in strings
29846. * Two's complement arithmetic.
29847. [... constructor arg1 ... argN]
29848. no need to decref previous value
29849. 'finally'
29850. Zero size compare not an issue with DUK_MEMCMP.
29851. [... closure template val]
29852. YIELDED: Ecmascript activation + Duktape.Thread.yield() activation
29853. may be NULL (but only if size is 0)
29854. write to entry part
29855. * Ecmascript bytecode executor. * * Resume execution for the current thread from its current activation. * Returns when execution would return from the entry level activation, * leaving a single return value on top of the stack. Function calls * and thread resumptions are handled internally. If an error occurs, * a longjmp()

with type DUK_LJ_TYPE_THROW is called on the entry level * setjmp() jmpbuf. ** Ecmascript function calls and coroutine resumptions are handled * internally (by the outer executor function) without recursive C calls. * Other function calls are handled using duk_handle_call(), increasing * C recursion depth. * * Abrupt completions (= long control transfers) are handled either * directly by reconfiguring relevant stacks and restarting execution, * or via a longjmp. Longjmp-free handling is preferable for performance * (especially Emscripten performance), and is used for: break, continue, * and return. ** For more detailed notes, see doc/execution.rst. ** Also see doc/code-issues.rst for discussion of setjmp(), longjmp(), * and volatile.

29856. * Debug connection message write helpers

29857. We know _Formals is dense and all entries will be in the * array part. GC and finalizers shouldn't affect _Formals * so side effects should be fine.

29858. don't want an intermediate exposed state with func == NULL

29859. **comment:** * (Re)try handling the longjmp. ** A longjmp handler may convert the longjmp to a different type and * "virtually" rethrow by goto'ing to 'check_longjmp'. Before the goto, * the following must be updated: * - the heap 'lj' state * - 'thr' must reflect the "throwing" thread

label: code-design

29860. XXX=

29861. separators: space, space, colon

29862. **comment:** Format of magic, bits: * 0..1: field type; 0:uint8, 1:uint16, 2:uint32, 3=float, 4=double, 5=unused, 6=unused, 7=unused * 3: endianness: 0=little, 1=big * 4: signed: 1=yes, 0=no * 5: typedarray: 1=yes, 0=no

label: code-design

29863. **comment:** Note: reuse 'curr' as a temp propdesc

label: code-design

29864. Slow path

29865. Lightweight function: never bound, so terminate.

29866. unconditionally; is_global==0

29867. Node.js Buffer: return offset + #bytes written (i.e. next * write offset).

29868. string limits

29869. **comment:** function makes one or more slow path accesses

label: code-design

29870. DUK_USE_AUGMENT_ERROR_CREATE

29871. No need for a NULL/zero-size check because new_size > 0)

29872. Assume value stack sizes (in elements) fits into duk_idx_t.

29873. **comment:** Relookup in case duk_js_tointeger() ends up e.g. coercing an object.

label: code-design

29874. '>'

29875. * Debug connection skip primitives

29876. valstack index of 'func' and retval (relative to entry valstack_bottom)

29877. XXX: array internal?

29878. Assertion compatible inside a comma expression, evaluates to void. * Currently not compatible with DUK_USE_PANIC_HANDLER() which may have * a statement block.

29879. avoid key quotes when key is an ASCII Identifier

29880. embed: nothing

29881. **comment:** bit mask which indicates that a regconst is a constant instead of a register

label: code-design

29882. * Memory allocation handling.

29883. current assertion is quite strong: decref's and set to unused

29884. Coerce an duk_ivalue to a 'plain' value by generating the necessary * arithmetic operations, property access, or variable access bytecode. * The duk_ivalue argument ('x') is converted into a plain value as a * side effect.

29885. element type

29886. parsing in "scanning" phase (first pass)

29887. regexp res_obj is at offset 4

29888. Flags for duk_js_compare_helper().

29889. **comment:** * Eval ** Eval needs to handle both a "direct eval" and an "indirect eval". * Direct eval handling needs access to the caller's activation so that its * lexical environment can be accessed. A direct eval is only possible from * Ecmascript code; an indirect eval call is possible also from C code. * When an indirect eval call is made from C code, there may not be a * calling activation at all which needs careful handling.

label: code-design

29890. no fractions

29891. 'Function'

29892. 29: setUTCMinutes

29893. * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written.

29894. accept comma, expect new value

29895. -> [... reviver holder name val]

29896. [... pattern flags escaped_source buffer]

29897. reg

29898. empty match -> bump and continue

29899. regexp execution limits

29900. SCANBUILD: "Dereference of null pointer", normal

29901. catch or with binding is currently active

29902. Note: unbalanced stack on purpose

29903. Get a borrowed duk_tval pointer to the current 'this' binding. Caller must * make sure there's an active callstack entry. Note that the returned pointer * is unstable with regards to side effects.

29904. expt values [0x001,0x7fe] = normal

29905. Receiver: Proxy object

29906. **comment:** * Dump/load helpers, xxx_raw() helpers do no buffer checks

label: code-design

29907. **comment:** * Push readable string summarizing duk_tval. The operation is side effect * free and will only throw from internal errors (e.g. out of memory). * This is used by e.g. property access code to summarize a key/base safely, * and is not intended to be fast (but small and safe).

label: code-design

29908. * Type error thrower, E5 Section 13.2.3.

29909. **comment:** Corner case: see test-numconv-parse-mant-carry.js. We could * just bump the exponent and update bitstart, but it's more robust * to recompute (but avoid rounding twice).

label: code-design

29910. The JA(value) operation: encode array. ** Stack policy: [array] -> [array].

29911. XXX: init or assert catch depth etc -- all values

29912. happens when hash part dropped

29913. 1x heap size

29914. no need for decref/incref because value is a number

29915. **comment:** XXX: this may now fail, and is not handled correctly

label: code-design

29916. [key undefined] -> [key]

29917. * join(), toLocaleString() ** Note: checking valstack is necessary, but only in the per-element loop. ** Note: the trivial approach of pushing all the elements on the value stack * and then calling duk_join() fails when the array contains a large number * of elements. This problem can't be offloaded to duk_join() because the

* elements to join must be handled here and have special handling. Current * approach is to do intermediate joins with very large number of elements. * There is no fancy handling; the prefix gets re-joined multiple times.

29918. stack top contains 'false'

29919. LAYOUT 2

29920. If 16-bit hash is in use, stuff it into duk_heapdr_string flags.

29921. * Other cases, use C recursion. * * If a tail call was requested we ignore it and execute a normal call. * Since Duktape 0.11.0 the compiler emits a RETURN opcode even after * a tail call to avoid test-bug-tailcall-thread-yield-resume.js. * * Direct eval call: (1) call target (before following bound function * chain) is the built-in eval() function, and (2) call was made with * the identifier 'eval'.

29922. (

29923. NOTE! Caller must ensure that any side effects from the * coercions below are safe. If that cannot be guaranteed * (which is normally the case), caller must coerce the * argument using duk_to_number() before any pointer * validations; the result of duk_to_number() always coerces * without side effects here.

29924. **comment:** Compared to duk_handle_call(): * - protected call: never * - ignore recursion limit: never

label: code-design

29925. Note: we ask for one return value from duk_safe_call to get this * error debugging here.

29926. -> [sep ToObject(this) len str]

29927. covered by comparison

29928. Setting "no shuffle A" is covered by the assert, but it's needed * with DUK_USE_SHUFFLE_TORTURE.

29929. **comment:** XXX: .code = err_code disabled, not sure if useful

label: code-design

29930. For now, restrict result array into 32-bit length range.

29931. coerce in-place

29932. DUK_ERR_ASSERTION_ERROR: no macros needed

29933. directive prologue status at entry

29934. * ToNumber() (E5 Section 9.3) * * Value to convert must be on stack top, and is popped before exit. * * See: <http://www.cs.indiana.edu/~burger/FP-Printing-PLDI96.pdf> * <http://www.cs.indiana.edu/~burger/fp/index.html> * * Notes on the conversion: * * - There are specific requirements on the accuracy of the conversion * through a "Mathematical Value" (MV), so this conversion is not * trivial. * * - Quick rejects (e.g. based on first char) are difficult because * the grammar allows leading and trailing white space. * * - Quick reject based on string length is difficult even after * accounting for white space; there may be arbitrarily many * decimal digits. * * - Standard grammar allows decimal values ("123"), hex values * ("0x123") and infinities * * - Unlike source code literals, ToNumber() coerces empty strings * and strings with only whitespace to zero (not NaN).

29935. [... num]

29936. arrays MUST have a 'length' property

29937. Using an explicit 'ASCII' flag has larger footprint (one call site * only) but is quite useful for the case when there's no explicit * 'clen' in duk_hstring.

29938. **comment:** XXX: try to optimize to 8 (would now be possible, <200 used)

label: code-design

29939. * Arguments handling helpers (argument map mainly). * * An arguments object has exotic behavior for some numeric indices. * Accesses may translate to identifier operations which may have * arbitrary side effects (potentially invalidating any duk_tval * pointers).

29940. Undefined/null are considered equal (e.g. "null == undefined" -> true).

29941. * String value of 'blen+1' bytes follows (+1 for NUL termination * convenience for C API). No alignment needs to be guaranteed * for strings, but fields above should guarantee alignment-by-4 * (but not alignment-by-8).

29942. **comment:** XXX: Could be improved by coercing to a readable duk_tval (especially string escaping)

label: code-design

29943. **comment:** Inner executor, performance critical.

label: code-design

29944. significant from precision perspective

29945. [... Logger clog logfunc clog]

29946. No assertions for offset or length; in particular, \ * it's OK for length to be longer than underlying \ * buffer. Just ensure they don't wrap when added. \

29947. fits into duk_small_int_t

29948. Caller doesn't need to check exotic proxy behavior (but does so for * some fast paths).

29949. is eval code

29950. * Iterate the bitstream (line diffs) until PC is reached

29951. varmap is already in comp_ctx->curr_func.varmap_idx

29952. * Convert binary digits into an IEEE double. Need to handle * denormals and rounding correctly.

29953. [key setter this val] -> [key retval]

29954. DUK_TOK LAND

29955. **comment:** * E5 Section 15.3.4.2 places few requirements on the output of * this function: * * - The result is an implementation dependent representation * of the function; in particular * * - The result must follow the syntax of a FunctionDeclaration. * In particular, the function must have a name (even in the * case of an anonymous function or a function with an empty * name). * * - Note in particular that the output does NOT need to compile * into anything useful.

label: code-design

29956. Note that multiple catchstack entries may refer to the same * callstack entry.

29957. (-Number.MAX_VALUE).toString(2).length == 1025, + spare

29958. DUK_HNATIVEFUNCTION_H_INCLUDED

29959. check pointer at end

29960. **comment:** XXX: any way to avoid decoding magic bit; there are quite * many function properties and relatively few with magic values.

label: code-design

29961. chain jumps for 'fall-through' * after a case matches.

29962. duk_unicode_ids_noa[]

29963. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).

label: code-design

29964. reset longjmp

29965. non-strict equality for buffers compares contents

29966. all codepoints up to U+FFFF

29967. recursion tracking happens here only

29968. roughly 128 bytes

29969. default clause matches next statement list (if any)

29970. * E5 Sections 11.8.7, 8.12.6. * * Basically just a property existence check using [[HasProperty]].

29971. new value

29972. MULTIPLICATIVE EXPRESSION

29973. **comment:** * typeof * * E5 Section 11.4.3. * * Very straightforward. The only question is what to return for our * non-standard tag / object types. * * There is an unfortunate string constant define naming problem with * typeof return values for e.g. "Object" and "object"; careful with * the built-in string defines. The LC_XXX defines are used for the * lowercase variants now.

label: code-design

29974. -> [... func this]

29975. **comment:** XXX: should this happen in the callee's activation or after unwinding?

label: code-design

29976. **comment:** * Four possible outcomes: * * 1. A 'finally' in the same function catches the 'return'. * It may continue to propagate when 'finally' is finished, * or it may be neutralized by 'finally' (both handled by * ENDFIN). * * 2. The return happens at the entry level of the bytecode * executor, so return from the executor (in C stack). * * 3. There is a calling (Ecmascript) activation in the call * stack => return to it, in the same executor instance. * * 4. There is no calling activation, and the

thread is * terminated. There is always a resumer in this case, * which gets the return value similarly to a 'yield' * (except that the current thread can no longer be * resumed).

label: code-design

29977. caller checks

29978. t has high mantissa

29979. * Flags ** Fixed buffer: 0 * Dynamic buffer: DUK_HBUFFER_FLAG_DYNAMIC * External buffer: DUK_HBUFFER_FLAG_DYNAMIC | DUK_HBUFFER_FLAG_EXTERNAL

29980. DUK_TOK_LCURLY

29981. **comment:** * Identifier access and function closure handling. ** Provides the primitives for slow path identifier accesses: GETVAR, * PUTVAR, DELVAR, etc. The fast path, direct register accesses, should * be used for most identifier accesses. Consequently, these slow path * primitives should be optimized for maximum compactness. ** Ecmascript environment records (declarative and object) are represented * as internal objects with control keys. Environment records have a * parent record ("outer environment reference") which is represented by * the implicit prototype for technical reasons (in other words, it is a * convenient field). The prototype chain is not followed in the ordinary * sense for variable lookups. ** See identifier-handling.rst for more details on the identifier algorithms * and the internal representation. See function-objects.rst for details on * what function templates and instances are expected to look like. ** Care must be taken to avoid duk_tval pointer invalidation caused by * e.g. value stack or object resizing. ** TODO: properties for function instances could be initialized much more * efficiently by creating a property allocation for a certain size and * filling in keys and values directly (and INCREFing both with "bulk incref" * primitives. ** XXX: duk_hobject_getprop() and duk_hobject_putprop() calls are a bit * awkward (especially because they follow the prototype chain); rework * if "raw" own property helpers are added.

label: code-design

29982. If 'day' is NaN, returns NaN.

29983. * New length not lower than old length => no changes needed * (not even array allocation).

29984. * API oriented helpers

29985. [enum_target res key true]

29986. **comment:** XXX: we could save space by using _Target OR _This. If _Target, assume * this binding is undefined. If _This, assumes this binding is _This, and * target is also _This. One property would then be enough.

label: code-design

29987. ADDITIVE EXPRESSION

29988. macros

29989. default case does not exist, or no statements present * after default case: finish case evaluation

29990. use temp_next for tracking register allocations

29991. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer

29992. DUK_REGEXP_H_INCLUDED

29993. **comment:** XXX: add fastint support?

label: requirement

29994. FOUND

29995. * Additional control information is placed into the object itself * as internal properties to avoid unnecessary fields for the * majority of functions. The compiler tries to omit internal * control fields when possible. ** Function templates: ** { * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * } ** Function instances: ** { * length: 2, * prototype: { constructor: <func> }, * caller: <thrower>, * arguments: <thrower>, * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * _Varenv: <variable environment of closure>, * _Lexenv: <lexical environment of closure (if differs from _Varenv)> * } ** More detailed description of these properties can be found * in the documentation.

29996. initialize for debug prints, needed if sce==NULL

29997. start of valstack allocation

29998. Allocate a new duk_hbuffer of a certain type and return a pointer to it * (NULL on error). Write buffer data pointer to 'out_bufdata' (only if * allocation successful).

29999. Lookup 'key' from arguments internal 'map', delete mapping if found. * Used in E5 Section 10.6 algorithm for [[Delete]]. Note that the * variable/argument itself (where the map points) is not deleted.

30000. -> [... enum_target res trap handler target]

30001. derived types

30002. Awkward inclusion condition: drop out of compilation if not needed by any * call site: object hash part or probing stringtable.

30003. next instruction to execute (points to 'func' bytecode, stable pointer), NULL for native calls

30004. [offset littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)

30005. Must be a "pointer object", i.e. class "Pointer"

30006. ES6 proxy

30007. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array * stack[4] = regexp res_obj (if is_regexp)

30008. value of dbg_exec_counter when we last did a Date-based check

30009. idx_retval unsigned

30010. no negative sign for zero

30011. **comment:** Two value cycle, see e.g. test-bi-date-tzoffset-basic-fi.js. * In these cases, favor a higher tzoffset to get a consistent * result which is independent of iteration count. Not sure if * this is a generically correct solution.

label: code-design

30012. 'thr' is the current thread, as no-one resumes except us and we * switch 'thr' in that case.

30013. lastIndex must be ignored for non-global regexps, but get the * value for (theoretical) side effects. No side effects can * really occur, because lastIndex is a normal property and is * always non-configurable for RegExp instances.

30014. clamp anything above nargs

30015. **comment:** XXX: this could be more compact by accessing the internal properties * directly as own properties (they cannot be inherited, and are not * externally visible).

label: code-design

30016. Get/set the current user visible size, without accounting for a dynamic * buffer's "spare" (= usable size).

30017. looping should never happen

30018. Validate byte read/write for virtual 'offset', i.e. check that the * offset, taking into account h->offset, is within the underlying * buffer size. This is a safety check which is needed to ensure * that even a misconfigured duk_hbufferobject never causes memory * unsafe behavior (e.g. if an underlying dynamic buffer changes * after being setup). Caller must ensure 'buf' != NULL.

30019. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.

30020. digit position is absolute, not relative

30021. c

30022. **comment:** Same test with volatiles

label: test

30023. There's overlap: the desired end result is that * conceptually a copy is made to avoid "trampling" * of source data by destination writes. We make * an actual temporary copy to handle this case.

30024. **comment:** XXX: just use toString() for now; permitted although not recommended. * nargs==1, so radix is passed to toString().

label: code-design

30025. %d; only 16 bits are guaranteed

30026. 'this' value: * E5 Section 6.b.i ** The only (standard) case where the 'this' binding is non-null is when * (1) the variable is found in an object environment record, and * (2) that object environment record is a 'with' block. *

30027. This is important to ensure dynamic buffer data pointer is not * NULL (which is possible if buffer size is zero), which in turn * causes portability issues with e.g. memmove() and memcpy().

30028. Since we already started writing the reply, just emit nothing.

30029. * For top-level objects, 'length' property has the following * default attributes: non-writable, non-enumerable, non-configurable * (E5 Section 15). ** However, 'length' property for Array.prototype has attributes * expected of an Array instance which are different: writable, * non-enumerable, non-configurable (E5 Section 15.4.5.2). ** This is currently determined implicitly based on class; there are * no attribute flags in the init data.

30030. -> [... errhandler undefined(= this) errval]

30031. Reconfigure value stack for return to an Ecmascript function at 'act_idx'.

30032. * First pass. ** Gather variable/function declarations needed for second pass. * Code generated is dummy and discarded.

30033. valid pc range is [0, length[

30034. Current tracedata contains 2 entries per callstack entry.

30035. [... errval]

30036. **comment:** XXX: there is a small risk here: because the ISO 8601 parser is * very loose, it may end up parsing some datetime values which * would be better parsed with a platform specific parser.
label: code-design

30037. * Return (t - LocalTime(t)) in minutes: ** t - LocalTime(t) = t - (t + LocalTZA + DaylightSavingTA(t)) * = -(LocalTZA + DaylightSavingTA(t)) ** where DaylightSavingTA() is checked for time 't'. ** Note that the sign of the result is opposite to common usage, * e.g. for EE(S)T which normally is +2h or +3h from UTC, this * function returns -120 or -180. *

30038. **comment:** XXX: no need for indirect call
label: code-design

30039. XXX: the implementation now assumes "chained" bound functions, * whereas "collapsed" bound functions (where there is ever only * one bound function which directly points to a non-bound, final * function) would require a "collapsing" implementation which * merges argument lists etc here.

30040. The current implementation of localeCompare() is simply a codepoint * by codepoint comparison, implemented with a simple string compare * because UTF-8 should preserve codepoint ordering (assuming valid * shortest UTF-8 encoding). ** The specification requires that the return value must be related * to the sort order: e.g. negative means that 'this' comes before * 'that' in sort order. We assume an ascending sort order.

30041. valstack

30042. not protected, respect reclimit, not constructor

30043. Push the resulting view object and attach the ArrayBuffer.

30044. **comment:** XXX: this should be optimized to be a raw query and avoid valstack * operations if possible.
label: code-design

30045. unbalanced stack

30046. value from hook

30047. * If object has been marked finalizable, move it to the * "to be finalized" work list. It will be collected on * the next mark-and-sweep if it is still unreachable * after running the finalizer.

30048. * Struct defs

30049. nbytes zero size case * <-----> * [... | p | x | x | q] [... | p==q] * => [... | x | x | q] [...]

30050. **comment:** not really necessary
label: code-design

30051. [... target] -> [... target keys]

30052. Leading zero.

30053. ;

30054. Make underlying buffer compact to match DUK_BW_GET_SIZE().

30055. first heap allocated, match bit boundary

30056. **comment:** XXX: more specific error classes?
label: code-design

30057. 25: setUTCSeconds

30058. * Intermediate value helpers

30059. Limits

30060. Encoded as surrogate pair, each encoding to 3 bytes for * 6 bytes total. Codepoints above U+10FFFF encode as 6 bytes * too, see duk_unicode_encode_cesu8().

30061. * Setup value stack: clamp to 'nargs', fill up to 'nregs' * Value stack may either grow or shrink, depending on the * number of func registers and the number of actual arguments. * If nregs >= 0, func wants args clamped to 'nargs'; else it * wants all args (= 'num_stack_args').

30062. [sep ToObject(this) len sep result]

30063. * Variant 2 or 4

30064. Lightfunc, always success.

30065. step 11.c is relevant only if non-strict (checked in 11.c.ii)

30066. XX==

30067. **comment:** A plain buffer coerces to a Duktape.Buffer because it's the * object counterpart of the plain buffer value. But it might * still make more sense to produce an ArrayBuffer here?
label: code-design

30068. Force exponential format. Used for toExponential().

30069. Cannot wrap: each object is at least 8 bytes so count is * at most 1/8 of that.

30070. wipe the capture range made by the atom (if any)

30071. Anything else is not constructable.

30072. attrs in E5 Section 15.3.5.1

30073. * Use the index in the header to find the right starting point

30074. Tolerate act_caller->func == NULL which happens in * some finalization cases; treat like unknown caller.

30075. **comment:** XXX: remove DUK_CALL_FLAG_IGNORE_RECLIMIT flag: there's now the * reclimit bump?
label: code-design

30076. maximum length of standard format tag that we support

30077. buffer size is >= 1

30078. ignore encoding for now

30079. enable manually for dumping

30080. (?=

30081. require to be safe

30082. Parser part count.

30083. for side effects, result ignored

30084. Now we can check offset validity.

30085. Milliseconds between status notify and transport peeks.

30086. Matching separator index is used in the control table

30087. rehash even if old and new sizes are the same to get rid of * DELETED entries.

30088. * Two pass approach to processing the property descriptors. * On first pass validate and normalize all descriptors before * any changes are made to the target object. On second pass * make the actual modifications to the target object. ** Right now we'll just use the same normalize/validate helper * on both passes, ignoring its outputs on the first pass.

30089. * Stringify implementation.

30090. ToNumber() for a double is a no-op.

30091. default: char escape (two chars)

30092. * Hobject allocation. ** Provides primitive allocation functions for all object types (plain object, * compiled function, native function, thread). The object return is not yet * in "heap allocated" list and has a refcount of zero, so caller must careful.

30093. Check actual underlying buffers for validity and that they * cover the copy. No side effects are allowed after the check * so that the validity status doesn't change.
30094. * Special parser for character classes; calls callback for every * range parsed and returns the number of ranges present.
30095. negate result
30096. * Lightfunc
30097. **comment:** * Refzero handling is skipped entirely if (1) mark-and-sweep is * running or (2) execution is paused in the debugger. The objects * are left in the heap, and will be freed by mark-and-sweep or * eventual heap destruction. * * This is necessary during mark-and-sweep because refcounts are also * updated during the sweep phase (otherwise objects referenced by a * swept object would have incorrect refcounts) which then calls here. * This could be avoided by using separate decref macros in * mark-and-sweep; however, mark-and-sweep also calls finalizers which * would use the ordinary decref macros anyway and still call this * function. * * This check must be enabled also when mark-and-sweep support has been * disabled: the flag is also used in heap destruction when running * finalizers for remaining objects, and the flag prevents objects from * being moved around in heap linked lists.
label: code-design
30098. XXX: len >= 0x80000000 won't work below because a signed type * is needed by qsort.
30099. Special handling for year sign.
30100. [arg1 ... argN obj length new_length]
30101. The algorithm in E5.1 Section 15.9.1.12 normalizes month, but * does not normalize the day-of-month (nor check whether or not * it is finite) because it's not necessary for finding the day * number which matches the (year,month) pair. * * We assume that duk__day_from_year() is exact here. * * Without an explicit infinity / NaN check in the beginning, * day_num would be a bogus integer here. * * It's possible for 'year' to be out of integer range here. * If so, we need to return NaN without integer overflow. * This fixes test-bug-setyear-overflow.js
30102. * Node.js Buffer.prototype.slice([start], [end]) * ArrayBuffer.prototype.slice(begin, [end]) * TypedArray.prototype.slice(begin, [end]) * * The API calls are almost identical; negative indices are counted from end * of buffer, and final indices are clamped (allowing crossed indices). Main * differences: * * - Copy/view behavior; Node.js .slice() and TypedArray .subarray() create * views, ArrayBuffer .slice() creates a copy * * - Resulting object has a different class and prototype depending on the * call (or 'this' argument) * * - TypedArray .subarray() arguments are element indices, not byte offsets
30103. **comment:** When alloc_size == 0 the second allocation may not * actually exist.
label: code-design
30104. statements go here (if any) on next loop
30105. 'yield'
30106. eat trailing comma
30107. pruned
30108. * Declarative environment record. * * Identifiers can never be stored in ancestors and are * always plain values, so we can use an internal helper * and access the value directly with an duk_tval ptr. * * A closed environment is only indicated by it missing * the "book-keeping" properties required for accessing * register-bound variables.
30109. For anything other than an Error instance, we calculate the * error location directly from the current activation if one * exists.
30110. Note: there is no requirement that: 'thr->callstack_preventcount == 1' * like for yield.
30111. lookup results is ignored
30112. **comment:** * ToPrimitive() (E5 Section 9.1) * * ==> implemented in the API.
label: requirement
30113. resume caller must be an ecmascript func
30114. e.g. a = -4, b = 5 --> -4 - 5 + 1 / 5 --> -8 / 5 --> -1 * a = -5, b = 5 --> -5 - 5 + 1 / 5 --> -9 / 5 --> -1 * a = -6, b = 5 --> -6 - 5 + 1 / 5 --> -10 / 5 --> -2
30115. stack contains value (if requested), 'out_desc' is set
30116. no issues with empty memcmp()
30117. [str] -> [substr]
30118. func is NULL for lightfunc
30119. property attributes for identifier (relevant if value != NULL)
30120. Default function to format objects. Tries to use toLogString() but falls * back to toString(). Any errors are propagated out without catching.
30121. don't allow continue
30122. respect reclimit, not constructor
30123. * Notation: double underscore used for internal properties which are not * stored in the property allocation (e.g. '__valstack').
30124. 0x60-0x6f
30125. DUK_BUFOBJ_INT32ARRAY
30126. Caller has already eaten the first char so backtrack one byte.
30127. controls for minimum entry part growth
30128. temp accumulation buffer
30129. Unwind.
30130. * Object built-ins
30131. callstack index
30132. **comment:** XXX: The ES5/5.1/6 specifications require that the key in 'key in obj' * must be string coerced before the internal HasProperty() algorithm is * invoked. A fast path skipping coercion could be safely implemented for * numbers (as number-to-string coercion has no side effects). For ES6 * proxy behavior, the trap 'key' argument must be in a string coerced * form (which is a shame).
label: code-design
30133. take last entry
30134. curr is data, desc is accessor
30135. **comment:** XXX: The incref macro takes a thread pointer but doesn't * use it right now.
label: code-design
30136. An object may be in heap_allocated list with a zero * refcount also if it is a temporary object created by * a finalizer; because finalization now runs inside * mark-and-sweep, such objects will not be queued to * refzero_list and will thus appear here with refcount * zero.
30137. Accept 32-bit decimal integers, no leading zeroes, signs, etc. * Leading zeroes are not accepted (zero index "0" is an exception * handled above).
30138. **comment:** Set: currently a finalizer is disabled by setting it to * undefined; this does not remove the property at the moment. * The value could be type checked to be either a function * or something else; if something else, the property could * be deleted.
label: code-design
30139. * Bitstream decoder
30140. statement id allocation (running counter)
30141. 'public'
30142. single match always
30143. Note: 'curr' refers to 'length' propdesc
30144. Use double parts, they tolerate unnormalized time. * * Note: DUK_DATE_IDX_WEEKDAY is initialized with a bogus value (DUK__PL_TZHOUR) * on purpose. It won't be actually used by duk_bi_date_get_timeval_from_dparts(), * but will make the value initialized just in case, and avoid any * potential for Valgrind issues.
30145. Steps 8-10 have been merged to avoid a "partial" variable.
30146. Used during heap destruction: don't actually run finalizers * because we're heading into forced finalization. Instead, * queue finalizable objects back to the heap_allocated list.
30147. duk_safe_call discipline
30148. **comment:** * Unicode tables containing ranges of Unicode characters in a * packed format. These tables are used to match non-ASCII * characters of complex productions by resorting to a linear * range-by-range comparison. This is very slow, but is expected * to be very rare in practical Ecmascript source code, and thus * compactness is most important. * * The tables are matched using uni_range_match() and the format * is described in src/extract_chars.py.
label: code-design
30149. **comment:** XXX: encoding is ignored now.
label: code-design
30150. use 'undefined' for value automatically

30151. [... obj ...]

30152. **comment:** Function pointers do not always cast correctly to void * * (depends on memory and segmentation model for instance), * so they coerce to NULL.
label: code-design

30153. Buffer length is bounded to 0xffff automatically, avoid compile warning.

30154. Integer division which floors also negative values correctly.

30155. consequence of above

30156. * Self tests to ensure execution environment is sane. Intended to catch * compiler/platform problems which cannot be detected at compile time.

30157. DUK_TAG_NUMBER would logically go here, but it has multiple 'tags'

30158. backtrack (safe)

30159. DUK_FLD_32BIT

30160. * Parse a RegExp token. The grammar is described in E5 Section 15.10. * Terminal constructions (such as quantifiers) are parsed directly here. * * 0xffffffffU is used as a marker for "infinity" in quantifiers. Further, * DUK_MAX_RE_QUANT_DIGITS limits the maximum number of digits that * will be accepted for a quantifier.

30161. **comment:** Note: reuse 'curr'
label: code-design

30162. * Convert and push final string.

30163. evaluate to final form (e.g. coerce GETPROP to code), throw away temp

30164. * Selftest code

30165. use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to the existing one)

30166. duk_tval ptr for 'func' on stack (borrowed reference) or tv_func_copy

30167. * Encoding and decoding basic formats: hex, base64. * * These are in-place operations which may allow an optimized implementation. * * Base-64: <https://tools.ietf.org/html/rfc4648#section-4>

30168. Run mark-and-sweep a few times just in case (unreachable object * finalizers run already here). The last round must rescue objects * from the previous round without running any more finalizers. This * ensures rescued objects get their FINALIZED flag cleared so that * their finalizer is called once more in forced finalization to * satisfy finalizer guarantees. However, we don't want to run any * more finalizer because that'd required one more loop, and so on.

30169. required to keep recursion depth correct

30170. [... buf loop (proplist) (gap) holder ""]

30171. Get default hash part size for a certain entry part size.

30172. Maximum number of digits generated.

30173. [... func buf] -> [... buf]

30174. **comment:** XXX: could share code with duk_js_ops.c, duk_js_compare_helper
label: code-design

30175. Math.pow(+0,y) should be Infinity when y<0. NetBSD pow() * returns -Infinity instead when y is <0 and finite. The * if-clause also catches y == -Infinity (which works even * without the fix).

30176. Free strings in the stringtable and any allocations needed * by the stringtable itself.

30177. e.g. DUK_OP_PREINCV

30178. If neutered must return 0; offset is zeroed during * neutering.

30179. array of declarations: [name1, val1, name2, val2, ...] * valN = (typeN) | (fnum << 8), where fnum is inner func number (0 for vars) * record function and variable declarations in pass 1

30180. breakpoints: [0,breakpoint_count[gc reachable

30181. **comment:** XXX: inefficient loop
label: code-design

30182. * Insert a jump offset at 'offset' to complete an instruction * (the jump offset is always the last component of an instruction). * The 'skip' argument must be computed relative to 'offset', * -without- taking into account the skip field being inserted. * * ... A B C ins X Y Z ... (ins may be a JUMP, SPLIT1/SPLIT2, etc) * => ... A B C ins SKIP X Y Z * * Computing the final (adjusted) skip value, which is relative to the * first byte of the next instruction, is a bit tricky because of the * variable length UTF-8 encoding. See doc/regexp.rst for discussion.

30183. * NFY <int: 5> <int: fatal> <str: msg> <str: filename> <int: linenumber> EOM

30184. Note: not necessary to check p against re_ctx->input_end: * the memory access is checked by duk_inp_get_cp(), while * valid compiled regexps cannot write a saved[] entry * which points to outside the string.

30185. Strings and ROM objects are never placed on the heap allocated list.

30186. [... val root ""] -> [... val val!]

30187. **comment:** * JSON built-ins. * * See doc/json.rst. * * Codepoints are handled as duk_uint_fast32_t to ensure that the full * unsigned 32-bit range is supported. This matters to e.g. JX. * * Input parsing doesn't do an explicit end-of-input check at all. This is * safe: input string data is always NUL-terminated (0x00) and valid JSON * inputs never contain plain NUL characters, so that as long as syntax checks * are correct, we'll never read past the NUL. This approach reduces code size * and improves parsing performance, but it's critical that syntax checks are * indeed correct!
label: code-design

30188. stack type fits into 16 bits

30189. 'set'

30190. low to high

30191. required, NULL implies detached

30192. right associative

30193. covers +Infinity

30194. no size check is necessary

30195. r <- (* f 2) * s <- (* (expt b (- e)) 2) == b^(-e) * 2 [if b==2 -> b^(1-e)] * m+ <- 1 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round

30196. **comment:** XXX: best behavior for real world compatibility?
label: code-design

30197. 34: setMonth

30198. 20: getSeconds

30199. refcount macros not defined without refcounting, caller must #ifdef now

30200. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is never NULL.

30201. DUK_MEMCMP() is guaranteed to return zero (equal) for zero length * inputs so no zero length check is needed.

30202. **comment:** XXX: shared parsing
label: code-design

30203. Note: reject negative zero.

30204. Delete entry in Duktape.modLoaded[] and rethrow.

30205. already closed

30206. '\xffFormals'

30207. 1

30208. XXX: Multiple tv_func lookups are now avoided by making a local * copy of tv_func. Another approach would be to compute an offset * for tv_func from valstack bottom and recomputing the tv_func * pointer quickly as valstack + offset instead of calling duk_get_tval().

30209. DUK_TOK_NEQ

30210. **comment:** XXX: would be nice to omit this jump when the jump is not * reachable, at least in the obvious cases (such as the case * ending with a 'break'. * * Perhaps duk_parse_stmt() could provide some info on whether * the statement is a "dead end"? * * If implemented, just set pc_prevstmt to -1 when not needed.
label: code-design

30211. no shrink

30212. XXX: duk_to_int() ensures we'll get 8 lowest bits as * as input is within duk_int_t range (capped outside it).

30213. * Ecmascript bytecode

30214. Current require() function
30215. [... errobj]
30216. [... arr]
30217. -> [... escaped_source]
30218. func
30219. ignore fclose() error
30220. DUK_TOK_RETURN
30221. * <https://github.com/svaarala/duktape/issues/127#issuecomment-77863473>
30222. **comment:** Note: not initializing all bytes is normally not an issue: Duktape won't * read or use the uninitialized bytes so valgrind won't issue warnings. * In some special cases a harmless valgrind warning may be issued though. * For example, the DumpHeap debugger command writes out a compiled function's * 'data' area as is, including any uninitialized bytes, which causes a * valgrind warning.
label: code-design
30223. XXX: to util
30224. Byte length would overflow.
30225. register declarations in first pass
30226. implicit this value always undefined for * declarative environment records.
30227. match_caps == 0 without regexps
30228. Value coercion (in stack): ToInteger(), E5 Section 9.4 * API return value coercion: custom
30229. thr->heap->jmpbuf_ptr is checked by duk_err_longjmp() so we don't * need to check that here. If the value is NULL, a panic occurs because * we can't return.
30230. * Object finalizer
30231. **comment:** never shrinks; auto-adds DUK_VALSTACK_INTERNAL_EXTRA, which is generous
label: code-design
30232. -> [... proplist enum_obj key]
30233. elems in source and dest
30234. E5 Section 15.4.5.1, step 4
30235. **comment:** * Handle special cases (NaN, infinity, zero).
label: code-design
30236. DUK_USE_ROM_OBJECTS
30237. stats for current expression being parsed
30238. **comment:** maintain highest 'used' temporary, needed to figure out nregs of function
label: code-design
30239. DUK_IVAL_XXX
30240. **comment:** Serialize uncovered backing buffer as a null; doesn't * really matter as long we're memory safe.
label: code-design
30241. * Error, fatal, and panic handling.
30242. A number can be loaded either through a constant, using * LDINT, or using LDINT+LDINTX. LDINT is always a size win, * LDINT+LDINTX is not if the constant is used multiple times. * Currently always prefer LDINT+LDINTX over a double constant.
30243. guaranteed by string limits
30244. Slot A
30245. terminates expression; e.g. post-increment/-decrement
30246. ignore arguments, return undefined (E5 Section 15.3.4)
30247. Thread state.
30248. g...v
30249. [... |] or [... | errobj (M * undefined)] where M = num_stack_rets - 1
30250. omit
30251. always >= 0
30252. Fixed buffer, no zeroing because we'll fill all the data.
30253. Ecma-to-ecma call possible, may or may not be a tail call. * Avoid C recursion by being clever.
30254. terminal type: no depth check
30255. track cpos while scanning
30256. DUK_TOK_SEQ
30257. curr_pc synced by duk_handle_ecma_call_setup()
30258. **comment:** XXX: could accept numbers larger than 32 bits, e.g. up to 53 bits?
label: code-design
30259. accessor flag not encoded explicitly
30260. For now only needed by the debugger.
30261. * Bytecode instruction representation during compilation * * Contains the actual instruction and (optionally) debug info.
30262. remove value
30263. * Forward declarations
30264. mm <- mp
30265. **comment:** LDINTX is not necessarily in FASTINT range, so * no fast path for now. * * XXX: perhaps restrict LDINTX to fastint range, wider * range very rarely needed.
label: code-design
30266. the entries [new_e_next, new_e_size_adjusted[are left uninitialized on purpose (ok, not gc reachable)
30267. rethrow
30268. * Accessor macros for function specific data areas
30269. Argument validation and func/args offset.
30270. day number for Jan 1 since epoch
30271. DUK_HOBJECT_FLAG_NATIVEFUNCTION varies
30272. compact
30273. **comment:** XXX: could clear FINALIZED already here; now cleared in * next mark-and-sweep.
label: code-design
30274. **comment:** XXX: this generates quite large code - perhaps select the error * class based on the code and then just use the error 'name'?
label: code-design
30275. require a short (8-bit) reg/const which fits into bytecode B/C slot
30276. **comment:** Function.prototype.bind() should never let this happen, * ugly error message is enough.
label: code-design
30277. "123." is allowed in some formats
30278. Apply ToNumber() to specified index; if ToInteger(val) in [0,99], add * 1900 and replace value at idx_val.
30279. NB: must accept reserved words as property name
30280. no array part
30281. * Misc
30282. error message doesn't matter, ignored anyway
30283. Resolve 'res' directly into the LHS binding, and use * that as the expression value if safe. If not safe, * resolve to a temp/const and copy to LHS.
30284. * Found existing own (non-inherited) plain property. * Do an access control check and update in place.
30285. Currently nothing to free
30286. **comment:** Convenience copies from heap/vm for faster access.
label: code-design

30287. Reply with tvals pushed by request callback
30288. If out of catchstack, cat = thr->catchstack - 1; * new_cat_top will be 0 in that case.
30289. **comment:** XXX: Could just lookup .toJSON() and continue in fast path, * as it would almost never be defined.
label: code-design
30290. **comment:** * Raw write/read macros for big endian, unaligned basic values. * Caller ensures there's enough space. The macros update the pointer * argument automatically on resizes. The idiom seems a bit odd, but * leads to compact code.
label: code-design
30291. **comment:** * "arguments" and "caller" must be mapped to throwers for strict * mode and bound functions (E5 Section 15.3.5). * * XXX: This is expensive to have for every strict function instance. * Try to implement as virtual properties or on-demand created properties.
label: code-design
30292. 'index'
30293. index is above internal string length -> property is fully normal
30294. LF line terminator; CR LF and Unicode lineterms are handled in slow path
30295. * Misc helpers
30296. expt== -1023 -> bitstart=0 (leading 1); * expt== -1024 -> bitstart=-1 (one left of leading 1), etc
30297. bound chain resolved
30298. base
30299. Line format: <time> <entryLev> <loggerName>: <msg>
30300. NEXTENUM jump slot: executed when enum finished
30301. **comment:** prototype should be last, for readability
label: code-design
30302. **comment:** XXX: direct implementation
label: requirement
30303. Error codes.
30304. multi-character sets not allowed as part of ranges, see * E5 Section 15.10.2.15, abstract operation CharacterRange.
30305. * Property lookup
30306. Constants for duk_hashstring().
30307. roughly 1 kB
30308. advance by one character (code point) and one char_offset
30309. no need to coerce
30310. remove key and value
30311. A regexp token value.
30312. large context; around 2kB now
30313. lexer character window helpers
30314. at index 1
30315. trivial cases
30316. duplicate/invalid key checks; returns 1 if syntax error
30317. Here we could remove references to built-ins, but it may not be * worth the effort because built-ins are quite likely to be shared * with another (unterminated) thread, and terminated threads are also * usually garbage collected quite quickly. Also, doing DECREFs * could trigger finalization, which would run on the current thread * and have access to only some of the built-ins. Garbage collection * deals with this correctly already.
30318. Name will be filled from function expression, not by caller. * This case is used by Function constructor and duk_compile() * API with the DUK_COMPILE_FUNCTION option.
30319. original target at entry_top - 1
30320. not emitted
30321. **comment:** XXX: duk_ssize_t would be useful here
label: code-design
30322. comp_ctx->lex.input and comp_ctx->lex.input_length filled by caller
30323. **comment:** XXX: pre-checks (such as no duplicate keys)
label: code-design
30324. To handle deeper indents efficiently, make use of copies we've * already emitted. In effect we can emit a sequence of 1, 2, 4, * 8, etc copies, and then finish the last run. Byte counters * avoid multiply with gap_len on every loop.
30325. **comment:** XXX: TRYCATCH handling should be reworked to avoid creating * an explicit scope unless it is actually needed (e.g. function * instances or eval is executed inside the catch block). This * rework is not trivial because the compiler doesn't have an * intermediate representation. When the rework is done, the * opcode format can also be made more straightforward.
label: code-design
30326. Because new_size != 0, if condition doesn't need to be * (p != NULL || new_size == 0).
30327. * Init lexer
30328. '%'
30329. * This test fails on an exotic ARM target; double-to-uint * cast is incorrectly clamped to -signed- int highest value. * *
<https://github.com/svaarala/duktape/issues/336>
30330. DUK_TOK_NUMBER
30331. IEEE exp without bias
30332. 31 bits
30333. call handling
30334. not exposed
30335. No need to check for size of bp_active list, * it's always larger than maximum number of * breakpoints.
30336. **comment:** * Restart execution by reloading thread state. * * Note that 'thr' and any thread configuration may have changed, * so all local variables are suspect and we need to reinitialize. * * The number of local variables should be kept to a minimum: if * the variables are spilled, they will need to be loaded from * memory anyway. * * Any 'goto restart_execution;' code path in opcode dispatch must * ensure 'curr_pc' is synced back to act->curr_pc before the goto * takes place. * * The interpreter must be very careful with memory pointers, as * many pointers are not guaranteed to be 'stable' and may be * reallocated and relocated on-the-fly quite easily (e.g. by a * memory allocation or a property access). * * The following are assumed to have stable pointers: * - the current thread * - the current function * - the bytecode, constant table, inner function table of the * current function (as they are a part of the function allocation) * * The following are assumed to have semi-stable pointers: * - the current activation entry: stable as long as callstack * is not changed (reallocated by growing or shrinking), or * by any garbage collection invocation (through finalizers) * - Note in particular that ANY DECREF can invalidate the * activation pointer, so for the most part a fresh lookup * is required * * The following are not assumed to have stable pointers at all: * - the value stack (registers) of the current thread * - the catch stack of the current thread * * See execution.rst for discussion.
label: code-design
30337. For internal use: get array part value
30338. 10xx xxxx -> invalid
30339. Compiler is responsible for selecting property flags (configurability, * writability, etc).
30340. _Varmap is dense
30341. DUK_TOK_BAND
30342. **comment:** 0x00 ... 0x7f: as is * 0x80: escape generically * 0x81: slow path * 0xa0 ... 0xff: backslash + one char
label: code-design
30343. * Exposed matcher function which provides the semantics of RegExp.prototype.exec(). * * RegExp.prototype.test() has the same semantics as exec() but does not return the * result object (which contains the matching string and capture groups). Currently * there is no separate test() helper, so a temporary result object is created and * discarded if test() is needed. This is intentional, to save code space. * * Input stack: [... re_obj input] * Output stack: [... result]
30344. * Buffers have no internal references. However, a dynamic * buffer has a separate allocation for the buffer. This is * freed by duk_heap_free_heapdr_raw().

30345. indirect eval
30346. Numbers half-way between integers must be rounded towards +Infinity, * e.g. -3.5 must be rounded to -3 (not -4). When rounded to zero, zero * sign must be set appropriately. E5.1 Section 15.8.2.15. ** Note that ANSI C round() is "round to nearest integer, away from zero", * which is incorrect for negative values. Here we make do with floor().
30347. * Some assertions (E5 Section 13.2).
30348. Shared lenient buffer length clamping helper. No negative indices, no * element/byte shifting.
30349. **comment:** XXX: some code might benefit from DUK_SETTEMP_IFTEMP(ctx,x)
 label: code-design
30350. preincrement and predecrement
30351. Value would normally be omitted, replace with 'null'.
30352. e.g. require('/foo'), empty terms not allowed
30353. string is internal
30354. * Parse regexp Disjunction. Most of regexp compilation happens here. ** Handles Disjunction, Alternative, and Term productions directly without * recursion. The only constructs requiring recursion are positive/negative * lookaheads, capturing parentheses, and non-capturing parentheses. ** The function determines whether the entire disjunction is a 'simple atom' * (see doc/regexp.rst discussion on 'simple quantifiers') and if so, * returns the atom character length which is needed by the caller to keep * track of its own atom character length. A disjunction with more than one * alternative is never considered a simple atom (although in some cases * that might be the case). ** Return value: simple atom character length or < 0 if not a simple atom. * Appends the bytecode for the disjunction matcher to the end of the temp * buffer. ** Regexp top level structure is: ** Disjunction = Term* * | Term* | Disjunction * * Term = Assertion * | Atom * | Atom Quantifier * * An empty Term sequence is a valid disjunction alternative (e.g. //|c||/). ** Notes: ** Tracking of the 'simple-ness' of the current atom vs. the entire * disjunction are separate matters. For instance, the disjunction * may be complex, but individual atoms may be simple. Furthermore, * simple quantifiers are used whenever possible, even if the * disjunction as a whole is complex. ** The estimate of whether an atom is simple is conservative now, * and it would be possible to expand it. For instance, captures * cause the disjunction to be marked complex, even though captures * -can- be handled by simple quantifiers with some minor modifications. ** Disjunction 'tainting' as 'complex' is handled at the end of the * main for loop collectively for atoms. Assertions, quantifiers, * and '| tokens need to taint the result manually if necessary. * Assertions cannot add to result char length, only atoms (and * quantifiers) can; currently quantifiers will taint the result * as complex though.
30355. Get a pointer to the current buffer contents (matching current allocation * size). May be NULL for zero size dynamic/external buffer.
30356. success: leave varname in stack
30357. safe
30358. [... closure template len_value]
30359. Note: negative pc values are ignored when patching jumps, so no explicit checks needed
30360. [... func this (crud) retval]
30361. DUK_USE_DATE_PRS_STRPTIME
30362. **comment:** Global case is more complex.
 label: code-design
30363. max, excl. varargs marker
30364. should be a safe way to compute this
30365. not an error
30366. prev_token slot2
30367. side effects, perform in-place
30368. **comment:** XXX: turkish / azeri
 label: code-design
30369. current variable environment (may be NULL if delayed)
30370. May happen in some out-of-memory corner cases.
30371. * Store lexer position for a later rewind
30372. -> [... this timeval]
30373. Avoid using RegExp.prototype methods, as they're writable and * configurable and may have been changed.
30374. Index validation is strict, which differs from duk_equals(). * The strict behavior mimics how instanceof itself works, e.g. * it is a TypeError if rval is not a - callable- object. It would * be somewhat inconsistent if rval would be allowed to be * non-existent without a TypeError.
30375. prev_token slot1
30376. Note1
30377. **comment:** When DUK_USE_HEAPPTR16 (and DUK_USE_REFCOUNT16) is in use, the * struct won't align nicely to 4 bytes. This 16-bit extra field * is added to make the alignment clean; the field can be used by * heap objects when 16-bit packing is used. This field is now * conditional to DUK_USE_HEAPPTR16 only, but it is intended to be * used with DUK_USE_REFCOUNT16 and DUK_USE_DOUBLE_LINKED_HEAP; * this only matter to low memory environments anyway.
 label: code-design
30378. DUK_TOK_IMPORT
30379. * Native call.
30380. input offset tracking
30381. -> [... closure template newobj closure]
30382. fixed arg count: value
30383. chain jumps for case * evaluation and checking
30384. Called for handling both a longjmp() with type DUK_LJ_TYPE_YIELD and * when a RETURN opcode terminates a thread and yields to the resumer.
30385. function call
30386. full 3-byte -> 4-char conversions
30387. regexp compilation limits
30388. **comment:** Current PC, accessed by other functions through thr->ptr_to_curr_pc. * Critical for performance. It would be safest to make this volatile, * but that eliminates performance benefits; aliasing guarantees * should be enough though.
 label: code-design
30389. 0x00: slow path * other: as is
30390. **comment:** XXX: this could be optimized; there is only one call site now though
 label: code-design
30391. reject 'in' token (used for for-in)
30392. Data area, fixed allocation, stable data ptrs.
30393. list of conversion specifiers that terminate a format tag; * this is unfortunately guesswork.
30394. Preliminaries for __proto__ and setPrototypeOf (E6 19.1.2.18 steps 1-4); * magic: 0=setter call, 1=Object.setPrototypeOf
30395. DUK_LEXER_H_INCLUDED
30396. type bit mask: types which certainly produce 'undefined'
30397. XXX: for threads, compact value stack, call stack, catch stack?
30398. **comment:** Lightfuncs are currently supported by coercing to a temporary * Function object; changes will be allowed (the coerced value is * extensible) but will be lost.
 label: code-design
30399. second, parse flags
30400. * Not found: write to global object (non-strict) or ReferenceError * (strict); see E5 Section 8.7.2, step 3.
30401. strict mode restrictions (E5 Section 12.2.1)
30402. **comment:** Forget the previous allocation, setting size to 0 and alloc to * NULL. Caller is responsible for freeing the previous allocation. * Getting the allocation and clearing it is done in the same API * call to avoid any chance of a realloc.
 label: code-design
30403. [regexp]

30404. '\xffFinalizer'

30405. **comment:** The entries can either be register numbers or 'null' values. * Thus, no need to DECREF them and get side effects. DECREF'ing * the keys (strings) can cause memory to be freed but no side * effects as strings don't have finalizers. This is why we can * rely on the object properties not changing from underneath us.

label: code-design

30406. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

30407. For all combinations: $+0 < +0, +0 < -0, -0 < +0, -0 < -0$, * steps e, f, and g.

30408. output is never longer than input during resolution

30409. **comment:** XXX: make fast paths optional for size minimization?

label: code-design

30410. **comment:** XXX: handling for array part missing now; this doesn't affect * compliance but causes array entry writes using defineProperty() * to always abandon array part.

label: code-design

30411. Fast path the binary case

30412. * Unicode helpers

30413. **comment:** XXX: zero assumption

label: code-design

30414. * Assertions after

30415. $r < r * s <- (* s B) * m+ <- m+ * m- <- m- * k <- (+ k 1)$

30416. 'Math'

30417. 1e8: protects against deeply nested inner functions

30418. insert ranges instruction, range count patched in later

30419. * Validate and convert argument property descriptor (an Ecmascript * object) into a set of defprop_flags and possibly property value, * getter, and/or setter values on the value stack. * * Lightfunc set/get values are coerced to full Functions.

30420. tc1 = false, tc2 = true

30421. after this, return paths should 'goto finished' for decrement

30422. obj_index

30423. zero-width non-joiner

30424. activation has active breakpoint(s)

30425. Constants for built-in string data unpacking.

30426. Need to set curr_token.t because lexing regexp mode depends on current * token type. Zero value causes "allow regexp" mode.

30427. start (inl) and end (excl) of trimmed part

30428. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.

30429. interpret e.g. '09' as '0', not NaN

30430. * splice()

30431. no argument given -> leave components untouched

30432. **comment:** This fast path is pretty marginal in practice. * XXX: candidate for removal.

label: code-design

30433. character represents itself

30434. eat 'return'

30435. **comment:** If thr != NULL, the thr may still be in the middle of * initialization. * XXX: Improve the thread viability test.

label: code-design

30436. [value offset end]

30437. Relies on NULL encoding to zero.

30438. **comment:** XXX: the current implementation works but is quite clunky; it compiles * to almost 1,4kB of x86 code so it needs to be simplified (better approach, * shared helpers, etc). Some ideas for refactoring: * * - a primitive to convert a string into a regexp matcher (reduces matching * code at the cost of making matching much slower) * - use replace() as a basic helper for match() and split(), which are both * much simpler * - API call to get_prop and to_boolean

label: code-design

30439. * Compilation and evaluation

30440. Note: Identifier rejects reserved words

30441. -> [obj trap_result]

30442. * Unicode letter is now taken to be the categories: * * Lu, Ll, Lt, Lm, Lo * * (Not sure if this is exactly correct.) * * The ASCII fast path consists of: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z']

30443. caller ensures

30444. * Not found (in registers or record objects). Declare * to current variable environment.

30445. **comment:** 'this' binding is not needed here

label: requirement

30446. no need to check callable; duk_call() will do that

30447. mark finalizer work list as reachability roots

30448. Initial '{' has been checked and eaten by caller.

30449. '\xffMap'

30450. The directive prologue flag is cleared by default so that it is * unset for any recursive statement parsing. It is only "revived" * if a directive is detected. (We could also make directives only * allowed if 'allow_source_elem' was true.)

30451. 'NaN'

30452. unsigned 31-bit range

30453. DUK_USE_EXPLICIT_NULL_INIT

30454. valgrind whine without this

30455. **comment:** * Helper for calling a user error handler. * * 'thr' must be the currently active thread; the error handler is called * in its context. The valstack of 'thr' must have the error value on * top, and will be replaced by another error value based on the return * value of the error handler. * * The helper calls duk_handle_call() recursively in protected mode. * Before that call happens, no longjmps should happen; as a consequence, * we must assume that the valstack contains enough temporary space for * arguments and such. * * While the error handler runs, any errors thrown will not trigger a * recursive error handler call (this is implemented using a heap level * flag which will "follow" through any coroutines resumed inside the * error handler). If the error handler is not callable or throws an * error, the resulting error replaces the original error (for Duktape * internal errors, duk_error_throw.c further substitutes this error with * a DoubleError which is not ideal). This would be easy to change and * even signal to the caller. * * The user error handler is stored in 'Duktape.errCreate' or * 'Duktape.errThrow' depending on whether we're augmenting the error at * creation or throw time. There are several alternatives to this approach, * see doc/error-objects.rst for discussion. * * Note: since further longjmp(s) may occur while calling the error handler * (for many reasons, e.g. a labeled 'break' inside the handler), the * caller can make no assumptions on the thr->heap->lj state after the * call (this affects especially duk_error_throw.c). This is not an issue * as long as the caller writes to the lj state only after the error handler * finishes.

label: code-design

30456. * Exposed calls

30457. key encountered as a setter

30458. Don't need to sync curr_pc here; duk_new() will do that * when it augments the created error.

30459. one i_step over

30460. Shared prefix for all buffer types.

30461. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. * * Delayed creation (on demand) is handled in duk_js_var.c.

30462. Reject a proxy object as the target because it would need * special handler in property lookups. (ES6 has no such restriction)

30463. must restore reliably before returning
30464. * Basic context initialization. * * Some init values are read from the bytecode header * whose format is (UTF-8 codepoints): * * uint flags * uint nsaved (even, 2n+2 where n = num captures)
30465. return 'res_arr' or 'null'
30466. **comment:** This would be nice, but parsing is faster without resetting the * value slots. The only side effect is that references to temporary * string values may linger until lexing is finished; they're then * freed normally.
label: code-design
30467. break jump
30468. "123"
30469. Note: the only yield-preventing call is Duktape.Thread.yield(), hence check for 1, not 0
30470. range check not necessary because all 4-bit values are mapped
30471. floor(1.15 * (1 << 10))
30472. expect_eof
30473. -- p_insert .-- p_curr * v v * | ... | insert | ... | curr
30474. * Stack constants
30475. call: this(fmt(arg)
30476. Combined step 11 (empty string special case) and 14-15.
30477. w/o refcounting
30478. Example: d=3.5, t=0.5 -> ret = (3 + 1) & 0xfe = 4 & 0xfe = 4 * Example: d=4.5, t=0.5 -> ret = (4 + 1) & 0xfe = 5 & 0xfe = 4
30479. Write fully.
30480. Assume IEEE round-to-even, so that shorter encoding can be used * when round-to-even would produce correct result. By removing * this check (and having low_ok == high_ok == 0) the results would * still be accurate but in some cases longer than necessary.
30481. object established using Function.prototype.bind()
30482. * Heap object refcount finalization. * * When an object is about to be freed, all other objects it refers to must * be decref'd. Refcount finalization does NOT free the object or its inner * allocations (mark-and-sweep shares these helpers), it just manipulates * the refcounts. * * Note that any of the decref's may cause a refcount to drop to zero, BUT * it will not be processed inline; instead, because refzero is already * running, the objects will just be queued to refzero list and processed * later. This eliminates C recursion.
30483. output radix
30484. resume check from proxy target
30485. native function properties
30486. 1e8
30487. Lightfunc flags packing and unpacking.
30488. Helpers exposed for internal use
30489. * Hash function duk_util_hashbytes(). * * Currently, 32-bit MurmurHash2. * * Don't rely on specific hash values; hash function may be endianness * dependent, for instance.
30490. **comment:** NULL always indicates alloc failure because * new_alloc_size > 0.
label: code-design
30491. The final object may be a normal function or a lightfunc. * We need to re-lookup tv_func because it may have changed * (also value stack may have been resized). Loop again to * do that; we're guaranteed not to come here again.
30492. **comment:** For objects JSON.stringify() only looks for own, enumerable * properties which is nice for the fast path here. * * For arrays JSON.stringify() uses [[Get]] so it will actually * inherit properties during serialization! This fast path * supports gappy arrays as long as there's no actual inherited * property (which might be a getter etc). * * Since recursion only happens for objects, we can have both * recursion and loop checks here. We use a simple, depth-limited * loop check in the fast path because the object-based tracking * is very slow (when tested, it accounted for 50% of fast path * execution time for input data with a lot of small objects!).
label: code-design
30493. assertion omitted
30494. -> [... obj finalizer]
30495. does not fit into an uchar
30496. need to normalize, may even cancel to 0
30497. Callee/caller are throwers and are not deletable etc. They * could be implemented as virtual properties, but currently * there is no support for virtual properties which are accessors * (only plain virtual properties). This would not be difficult * to change in duk_hobject_props, but we can make the throwers * normal, concrete properties just as easily. * * Note that the specification requires that the *same* thrower * built-in object is used here! See E5 Section 10.6 main * algorithm, step 14, and Section 13.2.3 which describes the * thrower. See test case test-arguments-throwers.js.
30498. used for array, stack, and entry indices
30499. [...] name reg_bind]
30500. Select copy mode. Must take into account element * compatibility and validity of the underlying source * buffer.
30501. invalid value which never matches
30502. Arrays never have other exotic behaviors.
30503. 0xxx xxxx [7 bits]
30504. [...] env target]
30505. Setup builtins from ROM objects. All heaps/threads will share * the same readonly objects.
30506. **comment:** alloc size in elements
label: code-design
30507. * Init remaining result fields * * 'nregs' controls how large a register frame is allocated. * * 'nargs' controls how many formal arguments are written to registers: * r0, ... r(nargs-1). The remaining registers are initialized to * undefined.
30508. note: caller 'sp' is intentionally not updated here
30509. Underlying buffer (refcounted), may be NULL.
30510. actual chars dropped (not just NUL term)
30511. free-form
30512. formals for 'func' (may be NULL if func is a C function)
30513. **comment:** bytes of indent available for copying
label: code-design
30514. * Valstack manipulation for results.
30515. [obj key desc value get set curr_value varname]
30516. * Add ._Tracedata to an error on the stack top.
30517. force executor restart to recheck breakpoints; used to handle function returns (see GH-303)
30518. Lexer context. Same context is used for Ecmascript and Regexp parsing.
30519. **comment:** XXX: spilling
label: code-design
30520. Note: rely on index ordering
30521. * Get enumerated keys in an Ecmascript array. Matches Object.keys() behavior * described in E5 Section 15.2.3.14.
30522. NB: duk_lexer_getpoint() is a macro only
30523. dangerous: must only lower (temp_max not updated)
30524. x <- x + y, use t as temp
30525. * Init argument related properties
30526. add BC*2^16
30527. initial stringtable size, must be prime and higher than DUK_UTIL_MIN_HASH_PRIME
30528. char format: use int

30529. Clamping only necessary for 32-bit ints.
30530. maximum size check is handled by callee
30531. -> [... val root val]
30532. Note: start_offset/end_offset can still be < 0 here.
30533. cat_idx catcher is kept, even for finally
30534. "null", "true", and "false" are always reserved words. * Note that "get" and "set" are not!
30535. AssignmentExpression
30536. DUK_TOK_MOD
30537. **comment:** XXX: this is pointless here because pass 1 is throw-away
 label: code-design
30538. Possibly truncated; there is no explicit truncation * marker so this is the best we can do.
30539. 'fatal'
30540. * Plain, boring reachable object.
30541. value of re_ctx->captures at start of atom
30542. * Find an existing key from entry part either by linear scan or by * using the hash index (if it exists). * * Sets entry index (and possibly the hash index) to output variables, * which allows the caller to update the entry and hash entries in-place. * If entry is not found, both values are set to -1. If entry is found * but there is no hash part, h_idx is set to -1.
30543. pc_label
30544. 'configurable'
30545. * Object prototype
30546. 'this' binding exists
30547. XXX: direct manipulation, or duk_replace_tval()
30548. fill window
30549. **comment:** * Helper for handling a "bound function" chain when a call is being made. * * Follows the bound function chain until a non-bound function is found. * Prepends the bound arguments to the value stack (at idx_func + 2), * updating 'num_stack_args' in the process. The 'this' binding is also * updated if necessary (at idx_func + 1). Note that for constructor calls * the 'this' binding is never updated by [[BoundThis]]. * * XXX: bound function chains could be collapsed at bound function creation * time so that each bound function would point directly to a non-bound * function. This would make call time handling much easier.
 label: code-design
30550. * Constructor arguments are currently somewhat compatible with * (keep it that way if possible): * * http://nodejs.org/api/buffer.html * * Note that the ToBuffer() coercion (duk_to_buffer()) does NOT match * the constructor behavior.
30551. * Emit compiled regexp header: flags, ncaptures * (insertion order inverted on purpose)
30552. * Logging support
30553. [key result]
30554. unchanged by Duktape.Thread.resume()
30555. arguments object would be accessible; note that shadowing * bindings are arguments or function declarations, neither * of which are deletable, so this is safe.
30556. Return 1: got EOM
30557. include removed: duk_internal.h
30558. NOTE: length may be zero
30559. SCANBUILD: NULL pointer dereference, doesn't actually trigger, * asserted above.
30560. Alignment guarantee
30561. #DUK_USE_PREFER_SIZE
30562. Note1: the specification doesn't require matching a time form with * just hours ("HH"), but we accept it here, e.g. "2012-01-02T12Z". * * Note2: the specification doesn't require matching a timezone offset * with just hours ("HH"), but accept it here, e.g. "2012-01-02T03:04:05+02"
30563. * Mark the heap.
30564. silence scan-build warning
30565. varname is popped by above code
30566. Estimating the result size beforehand would be costly, so * start with a reasonable size and extend as needed.
30567. Decode failed.
30568. * Init stringcache
30569. [... constructor arg1 ... argN final_cons]
30570. **comment:** XXX: lithuanian, explicit dot rules
 label: code-design
30571. XXX: similar coercion issue as in DUK_TOK_PERIOD
30572. **comment:** Wipe capture range and save old values for backtracking. * * XXX: this typically happens with a relatively small idx_count. * It might be useful to handle cases where the count is small * (say <= 8) by saving the values in stack instead. This would * reduce memory churn and improve performance, at the cost of a * slightly higher code footprint.
 label: code-design
30573. source out-of-bounds (but positive)
30574. 'with' binding has no catch clause, so can't be here unless a normal try-catch
30575. second incref for the entry reference
30576. **comment:** XXX: declvar takes an duk_tval pointer, which is awkward and * should be reworked.
 label: code-design
30577. Get.
30578. DUK_TOK_PRIVATE
30579. randomized pivot selection
30580. * Check whether the property already exists in the prototype chain. * Note that the actual write goes into the original base object * (except if an accessor property captures the write).
30581. These are not needed when only Duktape.Buffer is supported.
30582. duk_hobject specific fields.
30583. [val] -> []
30584. Map DUK_HBUFFEROBJECT_ELEM_xxx to duk_hobject class number. * Sync with duk_hbufferobject.h and duk_hobject.h.
30585. * Shared exit path
30586. force_exponential
30587. As an initial implementation, write flush after every EOM (and the * version identifier). A better implementation would flush only when * Duktape is finished processing messages so that a flush only happens * after all outbound messages are finished on that occasion.
30588. DUK_INVALID_INDEX won't be accepted as a valid index.
30589. * Heap thread object representation. * * duk_hthread is also the 'context' (duk_context) for exposed APIs * which mostly operate on the topmost frame of the value stack.
30590. DUK_USE_BUFFEROBJECT_SUPPORT
30591. As an initial implementation, read flush after exiting the message * loop. If transport is broken, this is a no-op (with debug logs).
30592. * Figure out how generated digits match up with the mantissa, * and then perform rounding. If mantissa overflows, need to * recompute the exponent (it is bumped and may overflow to * infinity). * * For normal numbers the leading '1' is hidden and ignored, * and the last bit is used for rounding: * * rounding pt * <-----52---->| * 1 x x x x ... x x x x y ==> x x x x ... x x x x * * For denormals, the leading '1' is included in the number, * and the rounding point is different: * * rounding pt * <-52 or less--->| * 1 x x x x ... x x x x y ==> 0 0 ... 1 x x ... x x * * The largest denormals will have a mantissa beginning with * a '1' (the explicit leading bit); smaller denormals will * have leading zero bits. * * If the exponent would become too high, the result becomes * Infinity. If the exponent is so small that the entire * mantissa becomes zero, the result becomes zero. * * Note: the Dragon4 'k' is off-by-one with respect to the IEEE * exponent. For instance, k==0 indicates that the leading '1' * digit is at the first binary fraction position (0.1xxx...); * the corresponding IEEE exponent would be -1.
30593. string hash

30594. Relookup in case coerce_func() has side effects, e.g. ends up coercing an object
30595. 0 = no update
30596. regexp opcodes
30597. 'function'
30598. recursion limit
30599. utf-8 validation ensures these
30600. count, not including sep
30601. string data is external (duk_hstring_external)
30602. currently paused: talk with debug client until step/resume
30603. duk_concat() coerces arguments with ToString() in correct order
30604. 2 to 11
30605. guaranteed to finish
30606. const flag for B
30607. one i_step added at top of loop
30608. 'res' may be NULL if new allocation size is 0.
30609. **comment:** Shared entry code for many Array built-ins. Note that length is left * on stack (it could be popped, but that's not necessary).
 label: code-design
30610. thread
30611. assume PC is at most 32 bits and non-negative
30612. 1 <2 bits>
30613. **comment:** * Shared handling for logical AND and logical OR. * * args = (truthval << 8) + rbp * * Truthval determines when to skip right-hand-side. * For logical AND truthval=1, for logical OR truthval=0. * * See doc/compiler.rst for discussion on compiling logical * AND and OR expressions. The approach here is very simplistic, * generating extra jumps and multiple evaluations of truth values, * but generates code on-the-fly with only local back-patching. * * Both logical AND and OR are syntactically left-associated. * However, logical ANDs are compiled as right associative * expressions, i.e. "A && B && C" as "A && (B && C)", to allow * skip jumps to skip over the entire tail. Similarly for logical OR.
 label: code-design
30614. NULL obj->p is OK
30615. * In strict mode E5 protects 'eval' and 'arguments' from being * assigned to (or even declared anywhere). Attempt to do so * should result in a compile time SyntaxError. See the internal * design documentation for details. * * Thus, we should never come here, run-time, for strict code, * and name 'eval' or 'arguments'.
30616. Assume that thr->valstack_bottom has been set-up before getting here.
30617. len: 9
30618. * E5 Section 11.4.9
30619. deal with weak references first
30620. duk_push_sprintf constants
30621. negative: dst info not available
30622. topmost element is the result array already
30623. Fast path, handle units with just actual encoding characters.
30624. out of spec, must be configurable
30625. * Debug logging after adjustment.
30626. * Pushers
30627. **comment:** * Note: each API operation potentially resizes the callstack, * so be careful to re-lookup after every operation. Currently * these is no issue because we don't store a temporary 'act' * pointer at all. (This would be a non-issue if we operated * directly on the array part.)
 label: code-design
30628. implicit_return_value
30629. XXX: direct valstack write
30630. [... source? filename]
30631. No resize has occurred so temp_desc->e_idx is still OK
30632. lookup for 0x000a above assumes shortest encoding now
30633. XXX: E5.1 Section 11.1.4 coerces the final length through * ToUint32() which is odd but happens now as a side effect of * 'arr_idx' type.
30634. Read tvals from the message and push them onto the valstack, * then call the request callback to process the request.
30635. true, despite side effect resizes
30636. the stack is unbalanced here on purpose; we only rely on the * initial two values: [name this].
30637. [start end str]
30638. source is eval code (not global)
30639. switch initial byte
30640. Note: pc is unsigned and cannot be negative
30641. * Casting
30642. **comment:** * User declarations, e.g. prototypes for user functions used by Duktape * macros. Concretely, if DUK_USE_PANIC_HANDLER is used and the macro * value calls a user function, it needs to be declared for Duktape * compilation to avoid warnings.
 label: code-design
30643. Longjmp handling has restored jmpbuf_ptr.
30644. * Round a number upwards to a prime (not usually the nearest one). * * Uses a table of successive 32-bit primes whose ratio is roughly * constant. This keeps the relative upwards 'rounding error' bounded * and the data size small. A simple 'predict-correct' compression is * used to compress primes to one byte per prime. See genhashsizes.py * for details. * * The minimum prime returned here must be coordinated with the possible * probe sequence steps in duk_hobject and duk_heap_stringtable.
30645. -> [... varname val this]
30646. processed one or more messages
30647. [... v1(filename) v2(line+flags)]
30648. must have been, since in array part
30649. slower but more compact variant
30650. TypedArray (or other non-ArrayBuffer duk_hbufferobject). * Conceptually same behavior as for an Array-like argument, * with a few fast paths.
30651. **comment:** parameter passing, not thread safe
 label: requirement
30652. **comment:** Two temporaries are preallocated here for variants 3 and 4 which need * registers which are never clobbered by expressions in the loop * (concretely: for the enumerator object and the next enumerated value). * Variants 1 and 2 "release" these temps.
 label: code-design
30653. Note: decimal number may start with a period, but must be followed by a digit
30654. main reachability roots
30655. DUK_HBUFFER_H_INCLUDED
30656. Inputs: explicit arguments (nargs), +1 for key, +2 for obj_index/nargs passing. * If the value stack does not contain enough args, an error is thrown; this matches * behavior of the other protected call API functions.
30657. [value offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
30658. placed in duk_heapdr_string
30659. * Main mark-and-sweep function. * * 'flags' represents the features requested by the caller. The current * heap->mark_and_sweep_base_flags is ORed automatically into the flags; * the base flags mask typically prevents certain mark-and-sweep operations * to avoid trouble.
30660. 'private'
30661. toJSON() can also be a lightfunc

30662. Special behavior for 'caller' property of (non-bound) function objects * and non-strict Arguments objects: if 'caller' -value- (!) is a strict * mode function, throw a TypeError (E5 Sections 15.3.5.4, 10.6). * Quite interestingly, a non-strict function with no formal arguments * will get an arguments object -without- special 'caller' behavior! * * The E5.1 spec is a bit ambiguous if this special behavior applies when * a bound function is the base value (not the 'caller' value): Section * 15.3.4.5 (describing bind()) states that [[Get]] for bound functions * matches that of Section 15.3.5.4 ([[Get]] for Function instances). * However, Section 13.3.5.4 has "NOTE: Function objects created using * Function.prototype.bind use the default [[Get]] internal method." * The current implementation assumes this means that bound functions * should not have the special [[Get]] behavior. * * The E5.1 spec is also a bit unclear if the TypeError throwing is * applied if the 'caller' value is a strict bound function. The * current implementation will throw even for both strict non-bound * and strict bound functions. * * See test-dev-strict-func-as-caller-prop-value.js for quite extensive * tests. * * This exotic behavior is disabled when the non-standard 'caller' property * is enabled, as it conflicts with the free use of 'caller'.
30663. move pivot out of the way
30664. final configuration
30665. -1 == not set, -2 == pending (next statement list)
30666. **comment:** temporary, must be signed and 32-bit to hold Unicode code points
label: code-design
30667. NB: 'length' property is automatically updated by the array setup loop
30668. index for next new key ([0,e_next[are gc reachable)
30669. idx is unsigned, < 0 check is not necessary
30670. indirect opcode follows direct
30671. * toFixed(), toExponential(), toPrecision()
30672. * Helpers for UTF-8 handling * * For bytecode readers the duk_uint32_t and duk_int32_t types are correct * because they're used for more than just codepoints.
30673. follow prototype chain
30674. [value]
30675. XXX: incref by count (here 2 times)
30676. equals
30677. true even with reattach
30678. Fatal error handling, called e.g. when a longjmp() is needed but * lj.jmpbuf_ptr is NULL. fatal_func must never return; it's not * declared as "noreturn" because doing that for typedefs is a bit * challenging portability-wise.
30679. **comment:** XXX: very messy now, but works; clean up, remove unused variables (nominally * used so compiler doesn't complain).
label: code-design
30680. * Note: type is treated as a field separate from flags, so some masking is * involved in the macros below.
30681. Compute final offset in seconds, positive if local time ahead of * UTC (returned value is UTC-to-local offset). * * difftime() returns a double, so coercion to int generates quite * a lot of code. Direct subtraction is not portable, however. * XXX: allow direct subtraction on known platforms.
30682. **comment:** XXX: this is a very narrow check, and doesn't cover * zeroes, subnormals, infinities, which compare normally.
label: code-design
30683. Don't allow a zero divisor. Fast path check by * "verifying" with multiplication. Also avoid zero * dividend to avoid negative zero issues (0 / -1 = -0 * for instance).
30684. numeric value of a hex digit (also covers octal and decimal digits)
30685. DUK_TOK_MUL_EQ
30686. DUK_USE_DEBUG
30687. **comment:** XXX: optimize to use direct reads, i.e. avoid * value stack operations.
label: code-design
30688. 0x00: finish (non-white) * 0x01: continue
30689. %lu
30690. [... arg1 ... argN]
30691. executor interrupt running (used to avoid nested interrupts)
30692. [... func this]
30693. Constants: variable size encoding.
30694. * Variable declarations. * * Unlike function declarations, variable declaration values don't get * assigned on entry. If a binding of the same name already exists, just * ignore it silently.
30695. **comment:** XXX: duk_small_uint_t would be enough for this loop
label: code-design
30696. no res->strs[]
30697. * Debug dumping of duk_heap.
30698. Detach a debugger if attached (can be called multiple times) * safely.
30699. if 1, doing a string-to-number; else doing a number-to-string
30700. **comment:** Faster alternative: avoid making a temporary copy of tvptr_dst and use * fast incref/decref macros.
label: code-design
30701. * duk_hbuffer allocation and freeing.
30702. stash to bottom of value stack to keep new_env reachable for duration of eval
30703. step 4.a
30704. Handle a RETURN opcode. Avoid using longjmp() for return handling because * it has a measurable performance impact in ordinary environments and an extreme * impact in Emscripten (GH-342). Return value is on value stack top.
30705. exports
30706. Note: must behave like a no-op with NULL and any pointer * returned from malloc/realloc with zero size.
30707. **comment:** XXX: use duk_is_valid_index() instead?
label: code-design
30708. eat closing bracket
30709. * Object.isSealed() and Object.isFrozen() (E5 Sections 15.2.3.11, 15.2.3.13) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: all virtual (non-concrete) properties are currently non-configurable * and non-writable (and there are no accessor virtual properties), so they don't * need to be considered here now.
30710. * The switch statement is pretty messy to compile. * See the helper for details.
30711. re_ctx->captures at start and end of atom parsing. * Since 'captures' indicates highest capture number emitted * so far in a DUK_REOP_SAVE, the captures numbers saved by * the atom are: Jstart_captures,end_captures].
30712. Loop check using a hybrid approach: a fixed-size visited[] array * with overflow in a loop check object.
30713. first value: comma must not precede the value
30714. actually used, non-NULL keys
30715. r <- (2 * b) * f
30716. [... re_obj input bc saved_buf res_obj]
30717. * Create wrapper object and serialize
30718. t1 <- t1 - s
30719. * Label handling * * Labels are initially added with flags prohibiting both break and continue. * When the statement type is finally uncovered (after potentially multiple * labels), all the labels are updated to allow/prohibit break and continue.
30720. 112.5%, i.e. new size less than 12.5% higher -> fast resize
30721. * URI handling
30722. * ASCII character constants * * C character literals like 'x' have a platform specific value and do * not match ASCII (UTF-8) values on e.g. EBCDIC platforms. So, use * these (admittedly awkward) constants instead. These constants must * also have signed values to avoid unexpected coercions in comparisons. * * <http://en.wikipedia.org/wiki/ASCII>
30723. 32-bit value

30724. string intern table (weak refs)

30725. **comment:** * Shared error message strings * * To minimize code footprint, try to share error messages inside Duktape * code. Modern compilers will do this automatically anyway, this is mostly * for older compilers.

label: code-design

30726. Convert indices to byte offsets.

30727. T

30728. Fast path: source is a TypedArray (or any bufferobject).

30729. Note: object cannot be a finalizable unreachable object, as * they have been marked temporarily reachable for this round, * and are handled above.

30730. 'var'

30731. [args(n) [crud] formals arguments map mappednames]

30732. **comment:** Note: 'this' is not necessarily an Array object. The push() * algorithm is supposed to work for other kinds of objects too, * so the algorithm has e.g. an explicit update for the 'length' * property which is normally "magical" in arrays.

label: code-design

30733. **comment:** * XXX: array indices are mostly typed as duk_uint32_t here; duk_uarridx_t * might be more appropriate.

label: code-design

30734. 'pointer'

30735. Return value would be pointless: because throw_flag==1, we always * throw if the identifier doesn't resolve.

30736. PC points to next instruction, find offending PC, * PC == 0 for native code.

30737. DUK_UNICODE_H_INCLUDED

30738. **comment:** dead code, but ensures portability (see Linux man page notes)

label: code-design

30739. DUK__ALLOW_AUTO_SEMI_ALWAYS workaround

30740. end of valstack allocation (exclusive)

30741. previous value is assumed to be garbage, so don't touch it

30742. * Scan number and setup for Dragon4. * * The fast path case is detected during setup: an integer which * can be converted without rounding, no net exponent. The fast * path could be implemented as a separate scan, but may not really * be worth it: the multiplications for building 'f' are not * expensive when 'f' is small. * * The significand ('f') must contain enough bits of (apparent) * accuracy, so that Dragon4 will generate enough binary output digits. * For decimal numbers, this means generating a 20-digit significand, * which should yield enough practical accuracy to parse IEEE doubles. * In fact, the Ecmascript specification explicitly allows an * implementation to treat digits beyond 20 as zeroes (and even * to round the 20th digit upwards). For non-decimal numbers, the * appropriate number of digits has been precomputed for comparable * accuracy. * * Digit counts: * * [dig_lzero] * | * .-...-[dig_prec]---. * | | | * 0000123.456789012345678901e+123456 * | | | | * `--+-`----[dig_frac]-----`-+-`* | * [dig_whole] [dig_expt] * * dig_frac and dig_expt are -1 if not present * dig_lzero is only computed for whole number part * * Parsing state * * Parsing whole part dig_frac < 0 AND dig_expt < 0 * Parsing fraction part dig_frac >= 0 AND dig_expt < 0 * Parsing exponent part dig_expt >= 0 (dig_frac may be < 0 or >= 0) * * Note: in case we hit an implementation limit (like exponent range), * we should throw an error, NOT return NaN or Infinity. Even with * very large exponent (or significand) values the final result may be * finite, so NaN/Infinity would be incorrect.

30743. **comment:** suppress warning, not used

label: code-design

30744. Buffer/string -> compare contents.

30745. * Update cache entry (allocating if necessary), and move the * cache entry to the first place (in an "LRU" policy).

30746. finished jump

30747. string is ASCII, clen == blen

30748. DUK_USE_HOBJECT_HASH_PART || DUK_USE_STRTAB_PROBE

30749. * Since character data is only generated by decoding the source or by * the compiler itself, we rely on the input codepoints being correct * and avoid a check here. * * Character data can also come here through decoding of Unicode * escapes ("udead\ubeef") so all 16-bit unsigned values can be * present, even when the source file itself is strict UTF-8.

30750. **comment:** XXX: more emergency behavior, e.g. find smaller hash sizes etc

label: code-design

30751. * Copy array elements to new array part.

30752. For strings, special form for short lengths.

30753. Setup function properties.

30754. **comment:** * In a fast check we assume old_size equals old_used (i.e., existing * array is fully dense). * * Slow check if: * * (new_size - old_size) / old_size > limit * new_size - old_size > limit * old_size * new_size > (1 + limit) * old_size || limit' is 3 bits fixed point * new_size > (1 + (limit' / 8)) * old_size || * 8 * 8 * new_size > (8 + limit') * old_size || : 8 * new_size > (8 + limit') * (old_size / 8) * new_size > limit' * (old_size / 8) || limit' = 9 -> max 25% increase * arr_idx + 1 > limit' * (old_size / 8) * * This check doesn't work well for small values, so old_size is rounded * up for the check (and the '+ 1' of arr_idx can be ignored in practice): * * arr_idx > limit' * ((old_size + 7) / 8)

label: code-design

30755. **comment:** * A few notes on the algorithm: * * - Terms are not allowed to begin with a period unless the term * is either '.' or '..'. This simplifies implementation (and * is within CommonJS modules specification). * * - There are few output bound checks here. This is on purpose: * the resolution input is length checked and the output is never * longer than the input. The resolved output is written directly * over the input because it's never longer than the input at any * point in the algorithm. * * - Non-ASCII characters are processed as individual bytes and * need no special treatment. However, U+0000 terminates the * algorithm; this is not an issue because U+0000 is not a * desirable term character anyway.

label: code-design

30756. * Utilities

30757. * Helpers * * The fast path checks are done within a macro to ensure "inlining" * while the slow path actions use a helper (which won't typically be * inlined in size optimized builds).

30758. **comment:** If the platform doesn't support the entire Ecmascript range, we need * to return 0 so that the caller can fall back to the default formatter. * * For now, assume that if time_t is 8 bytes or more, the whole Ecmascript * range is supported. For smaller time_t values (4 bytes in practice), * assumes that the signed 32-bit range is supported. * * XXX: detect this more correctly per platform. The size of time_t is * probably not an accurate guarantee of strftime() supporting or not * supporting a large time range (the full Ecmascript range).

label: code-design

30759. **comment:** XXX: now that pc2line is used by the debugger quite heavily in * checked execution, this should be optimized to avoid value stack * and perhaps also implement some form of pc2line caching (see * future work in debugger.rst).

label: code-design

30760. * Case clause. * * Note: cannot use reg_case as a temp register (for SEQ target) * because it may be a constant.

30761. **comment:** XXX: not sure what the correct semantic details are here, * e.g. handling of missing values (gaps), handling of non-array * trap results, etc. * * For keys, we simply skip non-string keys which seems to be * consistent with how e.g. Object.keys() will process proxy trap * results (ES6, Section 19.1.2.14).

label: code-design

30762. **comment:** XXX: 'copy properties' API call?

label: code-design

30763. * Determine whether to use the constructor return value as the created * object instance or not.

30764. Stack top contains plain value

30765. make current token the previous; need to fiddle with valstack "backing store"

30766. '{_func:true}'

30767. prediction: portable variant using doubles if 64-bit values not available

30768. [... arr val]

30769. Casting convenience.

30770. The stack has a variable shape here, so force it to the * desired one explicitly.

30771. **comment:** XXX: does not work if thr->catchstack is NULL

label: code-design
30772. Object.getOwnPropertyNames
30773. 'undefined'
30774. **comment:** XXX: with 'caller' property the callstack would need * to be unwound to update the 'caller' properties of * functions in the callstack.
label: code-design
30775. Overestimate required size; debug code so not critical to be tight.
30776. **comment:** no special formatting
label: code-design
30777. proplist is very rare
30778. * Handling DUK_OP_ADD this way is more compact (experimentally) * than a separate case with separate argument decoding.
30779. currently processing messages or breakpoints: don't enter message processing recursively (e.g. no breakpoints when processing debugger eval)
30780. DUK_USE_JC
30781. function refers to 'arguments' identifier
30782. don't want an intermediate exposed state with invalid pc
30783. Use loop to minimize code size of relookup after bound function case
30784. 'Undefined'
30785. **comment:** * Variable already declared, ignore re-declaration. * The only exception is the updated behavior of E5.1 for * global function declarations, E5.1 Section 10.5, step 5.e. * This behavior does not apply to global variable declarations.
label: code-design
30786. **comment:** Avoid fake finalization for the duk__refcount_fake_finalizer function * itself, otherwise we're in infinite recursion.
label: code-design
30787. **comment:** Borrow is detected based on wrapping which relies on exact 32-bit * types.
label: code-design
30788. if abandon_array, new_a_size must be 0
30789. Parse enumeration target and initialize enumerator. For 'null' and 'undefined', * INITENUM will creates a 'null' enumerator which works like an empty enumerator * (E5 Section 12.6.4, step 3). Note that INITENUM requires the value to be in a * register (constant not allowed).
30790. pointer is aligned, guaranteed for fixed buffer
30791. See MPUTOBJ comments above.
30792. Pause on the next opcode executed. This is always safe to do even * inside the debugger message loop: the interrupt counter will be reset * to its proper value when the message loop exits.
30793. Ensure there is internal valstack spare before we exit; this may * throw an alloc error. The same guaranteed size must be available * as before the call. This is not optimal now: we store the valstack * allocated size during entry; this value may be higher than the * minimal guarantee for an application.
30794. \xffPc2line'
30795. **comment:** XXX: better coercion
label: code-design
30796. guaranteed to finish, as hash is never full
30797. shared flags for a subset of types
30798. nothing to incref
30799. The short string/integer initial bytes starting from 0x60 don't have * defines now.
30800. **comment:** XXX: use a helper for prototype traversal; no loop check here
label: code-design
30801. FunctionDeclaration: not strictly a statement but handled as such. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().
30802. unpack args
30803. hash lookup
30804. * Cleanup: restore original function, restore valstack state.
30805. don't touch property attributes or hash part
30806. Note: duk_uint8_t type yields larger code
30807. * The error object has been augmented with a traceback and other * info from its creation point -- usually another thread. The * error handler is called here right before throwing, but it also * runs in the resumer's thread. It might be nice to get a traceback * from the resumee but this is not the case now.
30808. Push function object, init flags etc. This must match * duk_js_push_closure() quite carefully.
30809. return arg as-is
30810. NULL accepted
30811. Lightfunc, not blamed now.
30812. 0 = no throw
30813. **comment:** XXX: macros
label: code-design
30814. Interrupt counter for triggering a slow path check for execution * timeout, debugger interaction such as breakpoints, etc. The value * is valid for the current running thread, and both the init and * counter values are copied whenever a thread switch occurs. It's * important for the counter to be conveniently accessible for the * bytecode executor inner loop for performance reasons.
30815. start value for current countdown
30816. maximum recursion depth for loop detection stacks
30817. us
30818. [... holder name val]
30819. Allow automatic detection of hex base ("0x" or "0X" prefix), * overrides radix argument and forces integer mode.
30820. * Init arguments properties, map, etc.
30821. **comment:** NUL terminator handling doesn't matter here
label: code-design
30822. **comment:** XXX: error handling is incomplete. It would be cleanest if * there was a setjmp catchpoint, so that all init code could * freely throw errors. If that were the case, the return code * passing here could be removed.
label: code-design
30823. keep in on valstack, use borrowed ref below
30824. [... Logger clog logfunc clog(=this) msg]
30825. -> [... func this arg1 ... argN _Args]
30826. * Found * * Arguments object has exotic post-processing, see E5 Section 10.6, * description of [[GetOwnProperty]] variant for arguments.
30827. compiler's responsibility
30828. [... func this arg1 ... argN] (not tail call) * [this | arg1 ... argN] (tail call) * * idx_args updated to match
30829. func is NULL for lightfuncs
30830. shared exit path now
30831. Finish the wrapped module source. Force module.filename as the * function .filename so it gets set for functions defined within a * module. This also ensures loggers created within the module get * the module ID (or overridden filename) as their default logger name. * (Note capitalization: .filename matches Node.js while .fileName is * used elsewhere in Duktape.)
30832. **comment:** This limitation would be fixable but adds unnecessary complexity.
label: code-design
30833. -> [O toISOString O]
30834. * Finish
30835. -> [timeval this timeval]
30836. cached: valstack_end - valstack (in entries, not bytes)

30837. **comment:** XXX: awkward, especially the bunch of separate output values -> output struct?
label: code-design

30838. According to E5.1 Section 15.4.4.4 nonexistent trailing * elements do not affect 'length' of the result. Test262 * and other engines disagree, so update idx_last here too.

30839. * Bitstream encoder.

30840. * Once the whole refzero cascade has been freed, check for * a voluntary mark-and-sweep.

30841. Resolve start/end offset as element indices first; arguments * at idx_start/idx_end are element offsets. Working with element * indices first also avoids potential for wrapping.

30842. IdentifierStart production with ASCII and non-BMP excluded

30843. temp object for tracking / detecting duplicate keys

30844. negative

30845. * First create all built-in bare objects on the empty valstack. ** Built-ins in the index range [0,DUK_NUM_BUILTINS-1] have value * stack indices matching their eventual thr->builtins[] index. ** Built-ins in the index range [DUK_NUM_BUILTINS,DUK_NUM_ALL_BUILTINS] * will exist on the value stack during init but won't be placed * into thr->builtins[]. These are objects referenced in some way * from thr->builtins[] roots but which don't need to be indexed by * Duktape through thr->builtins[] (e.g. user custom objects).

30846. DUK_BUFOBJ_INT8ARRAY

30847. DUK_USE_ROM_STRINGS

30848. get pointer offsets for tweaking below

30849. Note: not normalized, but duk_push_number() will normalize

30850. NULL if not an object

30851. * Exposed number-to-string API ** Input: [number] * Output: [string]

30852. XXX: could add fast path for u8 compatible views

30853. Note2

30854. 'resume'

30855. should never happen for a strict callee

30856. **comment:** XXX: This is VERY inefficient now, and should be e.g. a * binary search or perfect hash, to be fixed.
label: code-design

30857. [... env]

30858. Lookup an identifier name in the current varmap, indicating whether the * identifier is register-bound and if not, allocating a constant for the * identifier name. Returns 1 if register-bound, 0 otherwise. Caller can * also check (out_reg_varbind >= 0) to check whether or not identifier is * register bound. The caller must NOT use out_rc_varname at all unless * return code is 0 or out_reg_varbind is < 0; this is because out_rc_varname * is unsigned and doesn't have a "unused" / none value.

30859. 'export'

30860. DUK_ERR_URL_ERROR: no macros needed

30861. array case is handled comprehensively above

30862. * Bytecode executor call. ** Execute bytecode, handling any recursive function calls and * thread resumptions. Returns when execution would return from * the entry level activation. When the executor returns, a * single return value is left on the stack top. ** The only possible longjmp() is an error (DUK_LJ_TYPE_THROW), * other types are handled internally by the executor.

30863. **comment:** XXX: refactor into an internal helper, pretty awkward
label: code-design

30864. Allocate a new valstack. ** Note: cannot use a plain DUK_REALLOC() because a mark-and-sweep may * invalidate the original thr->valstack base pointer inside the realloc * process. See doc/memory-management.rst

30865. Note that this is the same operation for strict and loose equality: * - E5 Section 11.9.3, step 1.c (loose) * - E5 Section 11.9.6, step 4 (strict)

30866. force_no_namebind

30867. DUK_TOK_NEW

30868. * Breakpoint and step state checks

30869. **comment:** XXX: this behavior is quite useless now; it would be nice to be able * to create pointer values from e.g. numbers or strings. Numbers are * problematic on 64-bit platforms though. Hex encoded strings?
label: code-design

30870. global object doesn't have array part

30871. **comment:** Then evaluate RHS fully (its value becomes the expression value too). * Technically we'd need the side effect safety check here too, but because * we always throw using INVLSH the result doesn't matter.
label: code-design

30872. **comment:** XXX: specific getter
label: code-design

30873. [... escape_source bytecode]

30874. First we need to insert a jump in the middle of previously * emitted code to get the control flow right. No jumps can * cross the position where the jump is inserted. See doc/compiler.rst * for discussion on the intricacies of control flow and side effects * for variants 3 and 4.

30875. Cannot simulate individual finalizers because finalize_list only * contains objects with actual finalizers. But simulate side effects * from finalization by doing a bogus function call and resizing the * stacks.

30876. lastIndex is initialized to zero by new RegExp()

30877. **comment:** Note: we could return the hash index here too, but it's not * needed right now.
label: code-design

30878. >>>

30879. [... closure template formals]

30880. inherit

30881. filename being compiled (ends up in functions' _filename' property)

30882. filter out flags from exprop rbp_flags here to save space

30883. **comment:** * Debugging macros, DUK_DPRINT() and its variants in particular. ** DUK_DPRINT() allows formatted debug prints, and supports standard * and Duktape specific formatters. See duk_debug_vsnprintf.c for details. ** DUK_D(x), DUK_DD(x), and DUK_DDD(x) are used together with log macros * for technical reasons. They are concretely used to hide 'x' from the * compiler when the corresponding log level is disabled. This allows * clean builds on non-C99 compilers, at the cost of more verbose code. * Examples: ** DUK_D(DUK_DPRINT("foo")); * DUK_DD(DUK_DPRINT("foo")); * DUK_DDD(DUK_DDDPRINT("foo")); * This approach is preferable to the old "double parentheses" hack because * double parentheses make the C99 solution worse: __FILE__ and __LINE__ can * no longer be added transparently without going through globals, which * works poorly with threading.
label: code-design

30884. prop index in 'array part', < 0 if not there

30885. caller handles sign change

30886. **comment:** load factor too low or high, count actually used entries and resize
label: code-design

30887. * Coercion and fast path processing.

30888. Not strictly necessary because if key == NULL, flag MUST be ignored.

30889. make the new thread reachable

30890. XXX: print built-ins array?

30891. temp reg

30892. * Externs and prototypes

30893. no incref

30894. Test signaling NaN and alias assignment in all endianness combinations.

30895. 1 GB

30896. XXX: fast path for array arguments?
30897. E5.1 Section 15.1.3.3: uriReserved + uriUnescaped + '#'
30898. DUK_USE_PACKED_TVAL
30899. **comment:** * Enumeration semantics come from for-in statement, E5 Section 12.6.4. * If called with 'null' or 'undefined', this opcode returns 'null' as * the enumerator, which is special cased in NEXTENUM. This simplifies * the compiler part
 label: code-design
30900. Insert a reserved area somewhere in the buffer; caller fills it. * Evaluates to a (duk_uint_t *) pointing to the start of the reserved * area for convenience.
30901. values for the state field
30902. How large a loop detection stack to use
30903. pop enum
30904. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)
30905. <
30906. [... closure template undefined]
30907. **comment:** * To determine whether to use an optimized Ecmascript-to-Ecmascript * call, we need to know whether the final, non-bound function is an * Ecmascript function. * * This is now implemented so that we start to do an ecma-to-ecma call * setup which will resolve the bound chain as the first thing. If the * final function is not eligible, the return value indicates that the * ecma-to-ecma call is not possible. The setup will overwrite the call * target at DUK_REGP(idx) with the final, non-bound function (which * may be a lightfunc), and fudge arguments if necessary. * * XXX: If an ecma-to-ecma call is not possible, this initial call * setup will do bound function chain resolution but won't do the * "effective this binding" resolution which is quite confusing. * * Perhaps add a helper for doing bound function and effective this * binding resolution - and call that explicitly? Ecma-to-ecma call * setup and normal function handling can then assume this prestep has * been done by the caller.
 label: code-design
30908. dynamic buffer ops
30909. not found in 'curr', next in prototype chain; impose max depth
30910. 13: getUTCDate
30911. **comment:** Truncate to 16 bits here, so that a computed hash can be compared * against a hash stored in a 16-bit field.
 label: code-design
30912. Buffer has virtual properties similar to string, but indexed values * are numbers, not 1-byte buffers/strings which would perform badly.
30913. [... this name]
30914. * Date/time parsing helper. * * Parse a datetime string into a time value. We must first try to parse * the input according to the standard format in E5.1 Section 15.9.1.15. * If that fails, we can try to parse using custom parsing, which can * either be platform neutral (custom code) or platform specific (using * existing platform API calls). * * Note in particular that we must parse whatever toString(), toUTCString(), * and toISOString() can produce; see E5.1 Section 15.9.4.2. * * Returns 1 to allow tail calling. * * There is much room for improvement here with respect to supporting * alternative datetime formats. For instance, V8 parses '2012-01-01' as * UTC and '2012/01/01' as local time.
30915. **comment:** XXX: check flags
 label: requirement
30916. ASCII (and EOF) fast path -- quick accept and reject
30917. 0xc0...0xcf
30918. '-' as a range indicator
30919. -> [... enum_target res handler undefined target]
30920. [arg1 ... argN this loggerLevel loggerName 'fmt' arg]
30921. "123.456"
30922. Helper for creating the arguments object and adding it to the env record * on top of the value stack. This helper has a very strict dependency on * the shape of the input stack.
30923. If an implicit return value is needed by caller, it must be * initialized to 'undefined' because we don't know whether any * non-empty (where "empty" is a continuation type, and different * from an empty statement) statements will be executed. * * However, since 1st pass is a throwaway one, no need to emit * it here.
30924. 'false'
30925. Some internal code now assumes that all duk_uint_t values * can be expressed with a duk_size_t.
30926. data property or accessor without getter
30927. * Args validation
30928. DUK_TAG_NUMBER is intentionally first, as it is the default clause in code * to support the 8-byte representation. Further, it is a non-heap-allocated * type so it should come before DUK_TAG_STRING. Finally, it should not break * the tag value ranges covered by case-clauses in a switch-case.
30929. * String built-ins
30930. * Clear (reachable) flags of refzero work list.
30931. **comment:** * Number should already be in NaN-normalized form, but let's * normalize anyway.
 label: code-design
30932. -> [... obj key trap handler]
30933. catchstack
30934. * C call recursion depth check, which provides a reasonable upper * bound on maximum C stack size (arbitrary C stack growth is only * possible by recursive handle_call / handle_safe_call calls).
30935. Call sites don't need the result length so it's not accumulated.
30936. non-zero: make copy
30937. Return non-zero (true) if we have a good reason to believe * the notify was delivered; if we're still attached at least * a transport error was not indicated by the transport write * callback. This is not a 100% guarantee of course.
30938. fromPresent = false
30939. Check that there's room to push one value.
30940. comma check
30941. backtrack to last output slash (dups already eliminated)
30942. **comment:** Fastint range is signed 48-bit so longest value is $-2^{47} = -140737488355328$ * (16 chars long), longest signed 64-bit value is $-2^{63} = -9223372036854775808$ * (20 chars long). Alloc space for 64-bit range to be safe.
 label: code-design
30943. Flags for call handling.
30944. jump to end
30945. * RegExp built-ins
30946. allocation size
30947. no -> decref members, then free
30948. If any non-Array value had enumerable virtual own * properties, they should be serialized here. Standard * types don't.
30949. XXX: C recursion limit if proxies are allowed as handler/target values
30950. try part
30951. 'caller' must only take on 'null' or function value
30952. sign doesn't matter when writing
30953. **comment:** XXX: use macros for the repetitive tval/refcount handling.
 label: code-design
30954. **comment:** Built-in-objects; may or may not be shared with other threads, * threads existing in different "compartments" will have different * built-ins. Must be stored on a per-thread basis because there * is no intermediate structure for a thread group / compartment. * This takes quite a lot of space, currently 43x4 = 172 bytes on * 32-bit platforms. * * In some cases the builtins array could be ROM based, but it's * sometimes edited (e.g. for sandboxing) so it's better to keep * this array in RAM.
 label: code-design

30955. **comment:** unused
 label: code-design

30956. * Other file level defines

30957. Fast path for numbers (one of which may be a fastint)

30958. maximum length for a SKIP-1 diffstream: 35 bits per entry, rounded up to bytes

30959. [enum_target enum res]

30960. orig value

30961. **comment:** XXX use get_tval ptr, more efficient
 label: code-design

30962. See comments in duk_pcall().

30963. The 'left' value must not be a register bound variable * because it may be mutated during the rest of the expression * and E5.1 Section 11.2.1 specifies the order of evaluation * so that the base value is evaluated first. * See: test-bug-nested-prop-mutate.js.

30964. A bit tricky overflow test, see doc/code-issues.rst.

30965. key must not already exist in entry part

30966. Digit generation

30967. reset function state (prepare for pass 2)

30968. +0 = catch

30969. avoid side effects!

30970. **comment:** NOTE: "get" and "set" are not officially ReservedWords and the lexer * currently treats them always like ordinary identifiers (DUK_TOK_GET * and DUK_TOK_SET are unused). They need to be detected based on the * identifier string content.
 label: code-design

30971. **comment:** XXX: regetting the pointer may be overkill - we're writing * to a side-effect free array here.
 label: code-design

30972. * Because buffer values may be looped over and read/written * from, an array index fast path is important.

30973. duk_small_uint_fast_t c = DUK_DEC_C(ins);

30974. For all duk_hbufferobjects, get the plain buffer inside * without making a copy. This is compatible with Duktape 1.2 * but means that a slice/view information is ignored and the * full underlying buffer is returned. * * If called as a constructor, a new Duktape.Buffer object * pointing to the same plain buffer is created below.

30975. Copy values by index reads and writes. Let virtual * property handling take care of coercion.

30976. **comment:** No coercions or other side effects, so safe
 label: requirement

30977. 0x20-0x2f

30978. may become sparse...

30979. Pointers may be NULL for a while when 'buf' size is zero and before any * ENSURE calls have been made. Once an ENSURE has been made, the pointers * are required to be non-NULL so that it's always valid to use memcpy() and * memmove(), even for zero size.

30980. * Create and throw an Ecmascript error object based on a code and a message. * * Used when we throw errors internally. Ecmascript generated error objects * are created by Ecmascript code, and the throwing is handled by the bytecode * executor.

30981. XXXXXX-

30982. lightfuncs are treated like objects and not coerced

30983. jump to end or else part

30984. **comment:** Registers 'bc' and 'bc + 1' are written in longjmp handling * and if their previous values (which are temporaries) become * unreachable -and- have a finalizer, there'll be a function * call during error handling which is not supported now (GH-287). * Ensure that both 'bc' and 'bc + 1' have primitive values to * guarantee no finalizer calls in error handling. Scrubbing also * ensures finalizers for the previous values run here rather than * later. Error handling related values are also written to 'bc' * and 'bc + 1' but those values never become unreachable during * error handling, so there's no side effect problem even if the * error value has a finalizer.
 label: code-design

30985. no_block

30986. The original value needs to be preserved for filter(), hence * this funny order. We can't re-get the value because of side * effects.

30987. for duk_error_augment.c

30988. just one 'int' for C++ exceptions

30989. '~'

30990. reset voluntary gc trigger count

30991. If the separator is a RegExp, make a "clone" of it. The specification * algorithm calls [[Match]] directly for specific indices; we emulate this * by tweaking lastIndex and using a "force global" variant of duk_regexp_match() * which will use global-style matching even when the RegExp itself is non-global.

30992. * Defines for JSON, especially duk.bi.json.c.

30993. * Fast path tables

30994. -> [... handler trap]

30995. DUK_TOK_ALSHIFT_EQ

30996. Note: assumes 'data' is always a fixed buffer

30997. * Stringable entry for fixed size stringtable

30998. sometimes stack and array indices need to go on the stack

30999. hash size ratio goal, must match genhashsizes.py

31000. * Heap structure. * * Heap contains allocated heap objects, interned strings, and built-in * strings for one or more threads.

31001. Require a lot of stack to force a value stack grow/shrink.

31002. **comment:** XXX: lastIndex handling produces a lot of asm
 label: code-design

31003. DUK_DEBUG_H_INCLUDED

31004. input size is good output starting point

31005. XXX: Current putvar implementation doesn't have a success flag, * add one and send to debug client?

31006. **comment:** must be able to emit code, alloc consts, etc.
 label: code-design

31007. **comment:** Should never happen but avoid infinite loop just in case.
 label: code-design

31008. unchanged from Duktape.Thread.yield()

31009. value1 -> return value, pseudo-type to indicate a return continuation (for ENDFIN)

31010. Negative offsets cause a RangeError.

31011. side effect free

31012. main expression parser function

31013. Always overflow.

31014. may happen if size is very close to 2^32-1

31015. fail: restore saves

31016. DUK_USE_DATE_PRS_GETDATE

31017. needed for stepping

31018. **comment:** XXX: could map/decode be unified with duk_unicode_support.c code? * Case conversion needs also the character surroundings though.
 label: code-design

31019. clen == blen -> pure ascii

31020. * The attempt to allocate may cause a GC. Such a GC must not attempt to resize * the stringtable (though it can be swept); finalizer execution and object * compaction must also be postponed to avoid the pressure to add strings to the * string table. Call site must prevent these.

31021. current setjmp() catchpoint

31022. Result is undefined.
31023. [... proplist]
31024. This may throw an error though not for valid E5 strings.
31025. * Marking functions for heap types: mark children recursively
31026. * Any catch bindings ("catch (e)") also affect identifier binding. ** Currently, the varmap is modified for the duration of the catch * clause to ensure any identifier accesses with the catch variable * name will use slow path.
31027. XXX: set with duk_hobject_set_length() when tracedata is filled directly
31028. **comment:** XXX: optimize by creating array into correct size directly, and * operating on the array part directly; values can be memcpy()'d from * value stack directly as long as refcounts are increased.
 label: code-design
31029. Slow path.
31030. 'compile'
31031. **comment:** XXX: does not work if thr->catchstack is allocated but lowest pointer
 label: code-design
31032. DUK_TOK_RCURLY
31033. **comment:** XXX: check .caller writability?
 label: code-design
31034. replacement handler
31035. $2^{**31-1} \approx 2G$ properties
31036. part of string
31037. Bytecode instructions: endian conversion needed unless * platform is big endian.
31038. DUK_TOK_EXTENDS
31039. XXX: optimize reconfig valstack operations so that resize, clamp, and setting * top are combined into one pass.
31040. fastint downgrade check for return values
31041. **comment:** bytes of indent still needed
 label: code-design
31042. DUK_JS_COMPILER_H_INCLUDED
31043. nop
31044. Restore the previous setjmp catcher so that any error in * error handling will propagate outwards rather than re-enter * the same handler. However, the error handling path must be * designed to be error free so that sandboxing guarantees are * reliable, see e.g. <https://github.com/svaarala/duktape/issues/476>.
31045. Double error
31046. * Valstack spare check
31047. This would be pointless: unexpected type and lightfunc would both return NULL
31048. For internal use: get non-accessor entry value
31049. trailing comma followed by rcurly
31050. Prevent mark-and-sweep for the pending finalizers, also prevents * refzero handling from moving objects away from the heap_allocated * list. (The flag meaning is slightly abused here.)
31051. [... value this_binding]
31052. DUK_TOK_INCREMENT
31053. * Shared handling of binary operations * * args = (is_extraop << 16) + (opcode << 8) + rbp
31054. insert code for matching the remainder - infinite or finite
31055. **comment:** XXX: shared api error strings, and perhaps even throw code for rare cases?
 label: code-design
31056. Although there are writable virtual properties (e.g. plain buffer * and buffer object number indices), they are handled before we come * here.
31057. * Fast paths
31058. **comment:** XXX: static alloc is OK until separate chaining stringtable * resizing is implemented.
 label: code-design
31059. truncate towards zero
31060. 37: setUTCFullYear
31061. **comment:** Function name: missing/undefined is mapped to empty string, * otherwise coerce to string.
 label: code-design
31062. * Assert helpers
31063. resume lookup from target
31064. **comment:** XXX: assume these?
 label: code-design
31065. When creating built-ins, some of the built-ins may not be set * and we want to tolerate that when throwing errors.
31066. standard prototype
31067. **comment:** XXX: consolidated algorithm step 15.f -> redundant?
 label: code-design
31068. flags: "gm"
31069. no need to decref, as previous value is 'undefined'
31070. may need unwind
31071. catch part will catch
31072. -> [... name index]
31073. * Node.js Buffer: toString([encoding], [start], [end])
31074. * EcmaScript compiler.
31075. Allow fraction part
31076. explicit 'this' binding, see GH-164
31077. Initial bytecode size allocation.
31078. XXX: patch initial size afterwards?
31079. already zero
31080. late lookup, avoid side effects
31081. Return value: $<0 \iff x < y * 0 \iff x == y * >0 \iff x > y$
31082. match against rule/part/sep bits
31083. 'd' and 'res' agree here
31084. no value, pseudo-type to indicate a normal continuation (for ENDFIN)
31085. Eval code doesn't need an automatic .prototype object.
31086. The coercion potentially invokes user .valueOf() and .toString() * but can't result in a function value because [[DefaultValue]] would * reject such a result: test-dev-json-stringify-coercion-1.js.
31087. never here
31088. **comment:** Macro for creating flag initializer from a class number. * Unsigned type cast is needed to avoid warnings about coercing * a signed integer to an unsigned one; the largest class values * have the highest bit (bit 31) set which causes this.
 label: code-design
31089. Note that 'l' and 'r' may cross, i.e. $r < l$
31090. For internal use: get non-accessor entry value and attributes
31091. It's important that duk_xdef_prop() is a 'raw define' so that any * properties in an ancestor are never an issue (they should never be * e.g. non-writable, but just in case).
31092. DUK_USE_JSON_EATWHITE_FASTPATH

label: code-design

31158. e.g. DUK_OP_POSTINCR

31159. 'Uint8ClampedArray'

31160. No copy, leave zero bytes in the buffer. There's no * ambiguity with Float32/Float64 because zero bytes also * represent 0.0.

31161. no typedef

31162. [...] func_template]

31163. Note: any entries above the callstack top are garbage and not zeroed. * Also topmost activation idx_retval is garbage (not zeroed), and must * be ignored.

31164. [arg1 ... argN obj length]

31165. Previous value of 'func' caller, restored when unwound. Only in use * when 'func' is non-strict.

31166. When doing string-to-number, lowest_mantissa is always 0 so * the exponent check, while incorrect, won't matter.

31167. If the value has a prototype loop, it's critical not to * throw here. Instead, assume the value is not to be * augmented.

31168. **comment:** unused with some debug level combinations

label: code-design

31169. Line number range for function. Needed during debugging to * determine active breakpoints.

31170. **comment:** slow init, do only for slow path cases

label: code-design

31171. DUK_BUFOBJ_DUKTAPE_BUFFER

31172. positive

31173. **comment:** XXX: Here we have a nasty dependency: the need to manipulate * the callstack means that catchstack must always be unwound by * the caller before unwinding the callstack. This should be fixed * later.

label: code-design

31174. e.g. DUK_OP_PREINCP

31175. DUK_JS_H_INCLUDED

31176. DUK_USE_DATE_NOW_WINDOWS

31177. !DUK_SINGLE_FILE

31178. this is added to checks to allow for Duktape * API calls in addition to function's own use

31179. [...] key_obj key]

31180. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

31181. is_setget

31182. flags always encodes to 1 byte

31183. Ensure space for 1:1 output plus one escape.

31184. [...] obj key value]

31185. * Byte-based matching would be possible for case-sensitive * matching but not for case-insensitive matching. So, we * match by decoding the input and bytecode character normally. * * Bytecode characters are assumed to be already canonicalized. * Input characters are canonicalized automatically by * duk_inp_get_cp() if necessary. * * There is no opcode for matching multiple characters. The * regexp compiler has trouble joining strings efficiently * during compilation. See doc/regexp.rst for more discussion.

31186. C array of duk_labelinfo

31187. **comment:** XXX: double evaluation for 'tv' argument.

label: code-design

31188. with stack depth (affects identifier lookups)

31189. [...] Duktape.modSearch resolved_id last_comp fresh_require exports module]

31190. * Detect recursive invocation

31191. **comment:** known to be 32-bit

label: code-design

31192. * Directive prologue tracking.

31193. Handle lightfuncs through object coercion for now.

31194. exponent 0

31195. Identifier found in registers (always non-deletable) * or declarative environment record and non-configurable.

31196. duk_handle_ecma_call_setup: setup for a tail call

31197. match: keep wiped/resaved values

31198. E5 Section 11.7.2, steps 7 and 8

31199. **comment:** * Since there are no references in the catcher structure, * unwinding is quite simple. The only thing we need to * look out for is popping a possible lexical environment * established for an active catch clause.

label: code-design

31200. * String cache. * * Provides a cache to optimize indexed string lookups. The cache keeps * track of (byte offset, char offset) states for a fixed number of strings. * Otherwise we'd need to scan from either end of the string, as we store * strings in (extended) UTF-8.

31201. heap thread, used internally and for finalization

31202. Don't allow a constant for the object (even for a number etc), as * it goes into the 'A' field of the opcode.

31203. For now all calls except Ecma-to-Ecma calls prevent a yield.

31204. If tv1==tv2 this is a NOP, no check is needed

31205. Dispatch loop.

31206. DUK_TOK_SUB

31207. Shared Windows helpers.

31208. This is a pretty awkward control flow, but we need to recheck the * key coercion here.

31209. catches EOF and invalid digits

31210. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur. If we end up not making the * call we must restore the value.

31211. **comment:** XXX: better macro for DUK_TVAL_IS_UNDEFINED_OR_NULL(tv)

label: code-design

31212. * Call the constructor function (called in "constructor mode").

31213. minimum prime

31214. set by atom case clauses

31215. [...] error func fileName]

31216. reset callstack limit

31217. right paren eaten

31218. T

31219. idx_desc

31220. E5 Section 11.7.1, steps 7 and 8

31221. coerced value is updated to value stack even when RangeError thrown

31222. Note: special DUK_EMIT_FLAG_B_IS_TARGETSOURCE * used to indicate that B is both a source and a * target register. When shuffled, it needs to be * both input and output shuffled.

31223. * Number is special because it doesn't have a specific * tag in the 8-byte representation.

31224. Align 'p' to 4; the input data may have arbitrary alignment. * End of string check not needed because blen >= 16.

31225. Expression_opt

31226. Exact halfway, round to even.

31227. * Arguments check

31228. **comment:** Very minimal inlining to handle common idioms '!0' and '!1', * and also boolean arguments like '!false' and '!true'.
label: code-design

31229. name is empty -> return message

31230. * Rescue or free.

31231. grow by at most one

31232. [A B C D E F G H] rel_index = 2, del_count 3, item count 3 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)

31233. **comment:** XXX: helper; rely on Boolean.prototype as being non-writable, non-configurable
label: code-design

31234. Note: proxy handling must happen before key is string coerced.

31235. DUK_HBUFFEROBJECT_H_INCLUDED

31236. [... res]

31237. Note: unconditional throw

31238. DUK_ERRMSG_H_INCLUDED

31239. Neutered buffer, zero length seems * like good behavior here.

31240. no mark-and-sweep gc

31241. * DUK_BSWAP macros

31242. literals (E5 Section 7.8), except null, true, false, which are treated * like reserved words (above).

31243. loggerName

31244. will result in undefined

31245. DUK_TOK_QUESTION

31246. * Delayed initialization only occurs for 'NEWENV' functions.

31247. Allow automatic detection of octal base, overrides radix * argument and forces integer mode.

31248. Load constants onto value stack but don't yet copy to buffer.

31249. [hobject props enum(props) key desc]

31250. 'Float32Array'

31251. eat comma

31252. range

31253. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT32

31254. Since stack indices are not reliable, we can't do anything useful * here. Invoke the existing setjmp catcher, or if it doesn't exist, * call the fatal error handler.

31255. [A B C D E F G H] rel_index = 2, del_count 3, item count 4 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)

31256. Commands and notifies initiated by Duktape.

31257. DUK_TOK_GT

31258. * Stringtable

31259. offset is now target of the pending split (right after jump)

31260. U+03A3 = GREEK CAPITAL LETTER SIGMA

31261. always 0 args

31262. exit bytecode executor by rethrowing an error to caller

31263. Note: e_next indicates the number of gc-reachable entries * in the entry part, and also indicates the index where the * next new property would be inserted. It does *not* indicate * the number of non-NULL keys present in the object. That * value could be counted separately but requires a pass through * the key list.

31264. initialize built-ins - either by copying or creating new ones

31265. Note: thr->catchstack_top may be 0, so that cat < thr->catchstack * initially. This is OK and intended.

31266. SCANBUILD: scan-build complains here about assigned value * being garbage or undefined. This is correct but operating * on undefined values has no ill effect and is ignored by the * caller in the case where this happens.

31267. IsAccessorDescriptor(desc) == true

31268. avoid issues with relocation

31269. avoid recursive string table call

31270. res.x1 -> res.x2

31271. -> [... x y fn]

31272. 'aint' result as complex -- this is conservative, * as lookaheads do not backtrack.

31273. LHS is already side effect free

31274. **comment:** XXX: hack, remove when const lookup is not O(n)
label: code-design

31275. Here the specification requires correct array index enumeration * which is a bit tricky for sparse arrays (it is handled by the * enum setup code). We now enumerate ancestors too, although the * specification is not very clear on whether that is required.

31276. object has an array part (a_size may still be 0)

31277. 0x20...0x2f

31278. eat dup slashes

31279. Get the current data pointer (caller must ensure buf != NULL) as a * duk_uint8_t ptr.

31280. initial estimate based on format string

31281. enable exotic behaviors last

31282. **comment:** if 'src < src_end_safe', safe to read 4 bytes
label: requirement

31283. **comment:** Straightforward algorithm, makes fewer compiler assumptions.
label: code-design

31284. [obj key desc]

31285. **comment:** Convenience for some situations; the above macros don't allow NULLs * for performance reasons.
label: code-design

31286. [... env callee varmap key val]

31287. currently required

31288. * Yield the current thread. * * The thread must be in yieldable state: it must have a resumer, and there * must not be any yield-preventing calls (native calls and constructor calls, * currently) in the thread's call stack (otherwise a resume would not be * possible later). This method must be called from an EcmaScript function. * * Args: * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.

31289. **comment:** No need to length check string: it will never exceed even * the 16-bit length maximum.
label: code-design

31290. (flags<<32) + (line), flags = 0

31291. insert range count

31292. XXX: Set 32-bit result (but must then handle signed and * unsigned results separately).

31293. [... new_glob new_env]

31294. Count actually used (non-NULL, non-DELETED) entries.

31295. no refcount check

31296. Parse statements until a closing token (EOF or '}') is found.

31297. duk_hthread_callstack_unwind() will decrease this on unwind

31298. Note: array dump will include elements beyond * 'length'.

31299. Note: currently the catch binding is handled without a register * binding because we don't support dynamic register bindings (they * must be fixed for an entire function). So, there is no need to * record regbases etc.

31300. Shared helper to provide `toString()` and `valueOf()`. Checks 'this', gets * the primitive value to stack top, and optionally coerces with `ToString()`.

31301. size stored in duk_heapheader unused flag bits
31302. genbuiltins.py ensures
31303. -> [... val']
31304. CR LF again a special case
31305. Note: computes with 'idx' in assertions, so caller beware. * 'idx' is preincremented, i.e. '1' on first call, because it * is more convenient for the caller.
31306. Full, aligned 4-byte reads.
31307. stack[0] = start * stack[1] = end * stack[2] = ToObject(this) * stack[3] = ToUint32(length) * stack[4] = result array
31308. * Zero the struct, and start initializing roughly in order
31309. stack[0...nargs-1] = unshift args (vararg) * stack[nargs] = ToObject(this) * stack[nargs+1] = ToUint32(length)
31310. duk_unicode_idp_m_ids_noa[]
31311. mark-and-sweep is currently running
31312. **comment:** * Error codes: defined in duktape.h * * Error codes are used as a shorthand to throw exceptions from inside * the implementation. The appropriate EcmaScript object is constructed * based on the code. EcmaScript code throws objects directly. The error * codes are defined in the public API header because they are also used * by calling code.
label: code-design
31313. DUK_API_INTERNAL_H_INCLUDED
31314. '\xffTarget'
31315. FUNCTION EXPRESSIONS
31316. 0x50-0x5f
31317. Note: there is no standard formatter for function pointers
31318. default: no change
31319. internal type tag
31320. * EcmaScript bytecode executor.
31321. **comment:** 0 = DUK_HOBJECT_CLASS_UNUSED
label: code-design
31322. p points to digit part ('%xy', p points to 'x')
31323. the compressed pointer is zeroed which maps to NULL, so nothing to do.
31324. 'count' is more or less comparable to normal trigger counter update * which happens in memory block (re)allocation.
31325. bottom of current frame
31326. Shared offset/length coercion helper.
31327. E5 Section 15.10.2.11
31328. 'writable'
31329. These functions have trouble working as lightfuncs. * Some of them have specific asserts and some may have * additional properties (e.g. 'require.id' may be written).
31330. Two lowest bits of opcode are used to distinguish * variants. Bit 0 = inc(0)/dec(1), bit 1 = pre(0)/post(1).
31331. internal spare
31332. value (error) is at stack top
31333. input byte length
31334. gap (if empty string, NULL)
31335. [args [crud] arguments]
31336. lightfunc
31337. higher value conserves memory; also note that linear scan is cache friendly
31338. **comment:** XXX: to be implemented?
label: code-design
31339. **comment:** XXX: convert to fixed buffer?
label: code-design
31340. **comment:** Debug macro to print all parts and dparts (used manually because of debug level).
label: code-design
31341. Assume arrays are dense in the fast path.
31342. cannot happen: strings are not put into refzero list (they don't even have the next/prev pointers)
31343. * Value stack top handling
31344. Set property slot to an empty state. Careful not to invoke * any side effects while using desc.e_idx so that it doesn't * get invalidated by a finalizer mutating our object.
31345. x <= y --> not (x > y) --> not (y < x)
31346. [key setter this val]
31347. little endian
31348. Typing: use duk_small_(u)int_fast_t when decoding small * opcode fields (op, A, B, C) and duk_(u)int_fast_t when * decoding larger fields (e.g. BC which is 18 bits). Use * unsigned variant by default, signed when the value is used * in signed arithmetic. Using variable names such as 'a', 'b', * 'c', 'bc', etc makes it easier to spot typing mismatches.
31349. * Create required objects: * - 'arguments' object: array-like, but not an array * - 'map' object: internal object, tied to 'arguments' * - 'mappedNames' object: temporary value used during construction
31350. * Misc util stuff
31351. Contract, either: * - Push string to value stack and return 1 * - Don't push anything and return 0
31352. Node.js return value for failed writes is offset + #bytes * that would have been written.
31353. NaN sign bit is platform specific with unpacked, un-normalized NaNs
31354. **comment:** XXX: union would be more correct
label: code-design
31355. * Allocate an duk_hobject. * * The allocated object has no allocation for properties; the caller may * want to force a resize if a desired size is known. * * The allocated object has zero reference count and is not reachable. * The caller MUST make the object reachable and increase its reference * count before invoking any operation that might require memory allocation.
31356. **comment:** XXX: Stack discipline is annoying, could be changed in numconv.
label: code-design
31357. 'stack'
31358. Also check for the refzero_list; must not be there unless it is * being finalized when duk_push_heapptr() is called. * * Corner case: similar to finalize_list.
31359. * Heap compiled function (EcmaScript function) representation. * * There is a single data buffer containing the EcmaScript function's * bytecode, constants, and inner functions.
31360. Behavior for nargs < 2 is implementation dependent: currently we'll * set a NaN time value (matching V8 behavior) in this case.
31361. ".123"
31362. currently used -> new size
31363. caller is responsible for ensuring this
31364. must be signed
31365. uppercase
31366. **comment:** This variant is needed by String.prototype.split(); it needs to perform * global-style matching on a cloned RegExp which is potentially non-global.
label: code-design
31367. **comment:** XXX: optimize this filling behavior later
label: code-design
31368. replaced

31369. **comment:** * Binary bitwise operations use different coercions (ToInt32, ToUint32) * depending on the operation. We coerce the arguments first using * ToInt32(), and then cast to an 32-bit value if necessary. Note that * such casts must be correct even if there is no native 32-bit type * (e.g., duk_int32_t and duk_uint32_t are 64-bit). * * E5 Sections 11.10, 11.7.1, 11.7.2, 11.7.3
label: code-design

31370. [... regexp input] -> [res_obj]
31371. Mostly API and built-in method related
31372. E5 Sections 15.9.3.1, B.2.4, B.2.5
31373. alloc and init
31374. prediction corrections for prime list (see genhashsizes.py)
31375. Make _Target and _Handler non-configurable and non-writable. * They can still be forcibly changed by C code (both user and * Duktape internal), but not by EcmaScript code.
31376. DUK_USE_REGEXP_SUPPORT
31377. the DUK_TOK_RCURLY is eaten by duk_parse_stmts()
31378. * Interrupt counter fixup (for development only).
31379. Note: recursive call
31380. maximum bytecode length in instructions
31381. get const for value at valstack top
31382. value1 -> label number, pseudo-type to indicate a break continuation (for ENDFIN)
31383. line tracking
31384. first call
31385. * All done
31386. undefined is accepted
31387. * Wrap up
31388. this case can no longer occur because refcount is unsigned
31389. automatic semi will be inserted
31390. Special shuffling for INITGET/INITSET, where slot C * identifies a register pair and cannot be shuffled * normally. Use an indirect variant instead.
31391. reachable through activation
31392. [... filename &comp_stk]
31393. **comment:** XXX: faster initialization (direct access or better primitives)
label: code-design

31394. Loop structure ensures that we don't compute t1^2 unnecessarily * on the final round, as that might create a bignum exceeding the * current DUK_BL_MAX_PARTS limit.
31395. **comment:** * Check for invalid backreferences; note that it is NOT an error * to back-reference a capture group which has not yet been introduced * in the pattern (as in \1(foo)/); in fact, the backreference will * always match! It IS an error to back-reference a capture group * which will never be introduced in the pattern. Thus, we can check * for such references only after parsing is complete.
label: code-design

31396. + 1 for rounding
31397. note: any entries above the callstack top are garbage and not zeroed
31398. must be, since env was created when catcher was created
31399. side effects (currently none though)
31400. DUK_BUILTIN_PROTOS_H_INCLUDED
31401. * Find a matching label catcher or 'finally' catcher in * the same function. * * A label catcher must always exist and will match unless * a 'finally' captures the break/continue first. It is the * compiler's responsibility to ensure that labels are used * correctly.
31402. shadowed, ignore
31403. 'Uint16Array'
31404. -> [... enum_target res]
31405. * Round-up limit. * * For even values, divides evenly, e.g. 10 -> roundup_limit=5. * * For odd values, rounds up, e.g. 3 -> roundup_limit=2. * If radix is 3, 0/3 -> down, 1/3 -> down, 2/3 -> up.
31406. +1 = one retval
31407. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. * * XXX: this is an overkill for some paths, so optimize this later * (or maybe switch to a stack arguments model entirely).
label: code-design

31408. %p' and NULL is portable, but special case it * anyway to get a standard NULL marker in logs.
31409. [... obj key val]
31410. **comment:** XXX: for Object.keys() we should check enumerability of key
label: code-design

31411. idx_value may be < 0 (no value), set and get may be NULL
31412. numeric value (character, count)
31413. Note: first character is checked against this. But because * IdentifierPart includes all IdentifierStart characters, and * the first character (if unescaped) has already been checked * in the if condition, this is OK.
31414. assume 'var' has been eaten
31415. empty separator can always match
31416. * Close environment record(s) if they exist. * * Only variable environments are closed. If lex_env != var_env, it * cannot currently contain any register bound declarations. * * Only environments created for a NEWENV function are closed. If an * environment is created for e.g. an eval call, it must not be closed.
31417. coerce to string
31418. * Not found in registers, proceed to the parent record. * Here we need to determine what the parent would be, * if 'env' was not NULL (i.e. same logic as when initializing * the record). * * Note that environment initialization is only deferred when * DUK_HOBJECT_HAS_NEWENV is set, and this only happens for: * - Function code * - Strict eval code * * We only need to check _Lexenv here; _Varenv exists only if it * differs from _Lexenv (and thus _Lexenv will also be present).
31419. no check
31420. **comment:** XXX: optimize to 16 bytes
label: code-design

31421. * Endian conversion
31422. nrets
31423. * Second (and possibly third) pass. * * Generate actual code. In most cases the need for shuffle * registers is detected during pass 1, but in some corner cases * we'll only detect it during pass 2 and a third pass is then * needed (see GH-115).
31424. embed: integer value
31425. 0=base, 1=esc, 2=class, 3=class+esc
31426. XXX: proper flags?
31427. mark-and-sweep flags automatically active (used for critical sections)
31428. heap string indices are autogenerated in duk_strings.h
31429. some assertions
31430. array writes autoincrement length
31431. **comment:** * Special behavior in E5.1. * * Note that even though parents == 0, the conflicting property * may be an inherited property (currently our global object's * prototype is Object.prototype). Step 5.e first operates on * the existing property (which is potentially in an ancestor) * and then defines a new property in the global object (and * never modifies the ancestor). * * Also note that this logic would become even more complicated * if the conflicting property might be a virtual one. Object * prototype has no virtual properties, though. * * XXX: this is now very awkward, rework.
label: code-design

31432. Avoid zero copy with an invalid pointer. If obj->p is NULL, * the 'new_a' pointer will be invalid which is not allowed even * when copy size is zero.
31433. reserved words: keywords
31434. The #ifdef clutter here handles the JX/JC enable/disable * combinations properly.
31435. [... key val]
31436. **comment:** * Check whether or not we have an error handler. * * We must be careful of not triggering an error when looking up the * property. For instance, if the property is a getter, we don't want * to call it, only plain values are allowed. The value, if it exists, * is not checked. If the value is not a function, a TypeError happens * when it is called and that error replaces the original one.
label: code-design
31437. **comment:** XXX: 'res' setup can be moved to function body level; in fact, two 'res' * intermediate values suffice for parsing of each function. Nesting is needed * for nested functions (which may occur inside expressions).
label: code-design
31438. **comment:** This seems to waste a lot of stack frame entries, but good compilers * will compute these as needed below. Some of these initial flags are * also modified in the code below, so they can't all be removed.
label: code-design
31439. [... env callee varmap]
31440. duk_push_(u)int() is guaranteed to support at least (un)signed 32-bit range
31441. [body formals source template]
31442. User callback did not return source code, so module loading * is finished: just update modLoaded with final module.exports * and we're done.
31443. **comment:** XXX: make active breakpoints actual copies instead of pointers?
label: code-design
31444. XXX: this bound function resolution also happens elsewhere, * move into a shared helper.
31445. **comment:** XXX: non-callable . toJSON() doesn't need to cause an abort * but does at the moment, probably not worth fixing.
label: code-design
31446. num args starting from idx_argbase
31447. caller must change active thread, and set thr->resumer to NULL
31448. check that the valstack has space for the final amount and any * intermediate space needed; this is unoptimal but should be safe
31449. unary plus
31450. * Shortcut for accessing global object properties
31451. * slice()
31452. may be changed
31453. step 3.e: replace 'Desc.[[Value]]'
31454. Captures which are undefined have NULL pointers and are returned * as 'undefined'. The same is done when saved[] pointers are insane * (this should, of course, never happen in practice).
31455. Only direct eval inherits strictness from calling code * (E5.1 Section 10.1.1).
31456. Get a (possibly canonicalized) input character from current sp. The input * itself is never modified, and captures always record non-canonicalized * characters even in case-insensitive matching.
31457. **comment:** * Update idx_retval of current activation. * * Although it might seem this is not necessary (bytecode executor * does this for Ecmascript-to-Ecmascript calls; other calls are * handled here), this turns out to be necessary for handling yield * and resume. For them, an Ecmascript-to-native call happens, and * the Ecmascript call's idx_retval must be set for things to work.
label: code-design
31458. [... varmap]
31459. * Type checking
31460. * ===== * Duktape authors * ===== * * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6dman <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang\u00f3 <lango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6ttsche (<https://github.com/jaseg>) * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6dman * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * https://github.com/yushli * * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * https://github.com/Kelledin * * https://github.com/sstruktrup * * Michael Drake (<https://github.com/tlsa>) * * https://github.com/chris-y * * Laurent Zubaiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn\u00e5s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.
31461. Grow array part for a new highest array index.
31462. no need to mask idx portion
31463. remove hash entry (no decref)
31464. index
31465. exclusive endpoint
31466. Append bytes from a slice already in the buffer.
31467. this also increases refcount by one
31468. whole, won't clip
31469. [source template]
31470. Cannot overlap.
31471. DUK_L0() cannot be a digit, because the loop doesn't terminate if it is
31472. apparently no hint?
31473. **comment:** DUK_USE_REGEXP_CANON_WORKAROUND
label: code-design
31474. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array
31475. XXX: clumsy sign extend and masking of 16 topmost bits
31476. Note: no key on stack
31477. [... retval]
31478. DUK_USE_JX
31479. num args
31480. end of _Formals
31481. This would be pointless: we'd return NULL for both lightfuncs and * unexpected types.
31482. 2 when called by debugger
31483. * Top level wrappers
31484. DUK_INTERNAL_H_INCLUDED
31485. Report next pc/line to be executed.
31486. Non-object argument is simply int coerced, matches * V8 behavior (except for "null", which we coerce to * 0 but V8 TypeErrors).
31487. * Build function fixed size 'data' buffer, which contains bytecode, * constants, and inner function references. * * During the building phase 'data' is reachable but incomplete. * Only incref's occur during building (no refzero or GC happens), * so the building process is atomic.

31488. fixed offsets in valstack
31489. * Math built-ins
31490. **comment:** Temporary structure used to pass a stack allocated region through * duk_safe_call().
label: code-design
31491. check that there is space for at least one new entry
31492. 0xa0...0xaf
31493. eval left argument first
31494. -> [... closure template]
31495. 1 << 52
31496. must match genhashsizes.py
31497. setter and/or getter may be NULL
31498. * Encoding helpers * * Some of the typing is bytecode based, e.g. slice sizes are unsigned 32-bit * even though the buffer operations will use duk_size_t.
31499. current thread
31500. **comment:** Called on a read/write error: NULL all callbacks except the detached * callback so that we never accidentally call them after a read/write * error has been indicated. This is especially important for the transport * I/O callbacks to fulfill guaranteed callback semantics.
label: code-design
31501. Emit 3 bytes and backtrack if there was padding. There's * always space for the whole 3 bytes so no check needed.
31502. Duktape.Thread.yield() should prevent
31503. encode \xffBar as _Bar if no quotes are applied, this is for * readable internal keys.
31504. LICENSE.txt
31505. [... varname val]
31506. string is 'eval' or 'arguments'
31507. * The default property attributes are correct for all * function valued properties of built-in objects now.
31508. Heap udata, used for allocator functions but also for other heap * level callbacks like pointer compression, etc.
31509. IdentifierStart production with ASCII excluded
31510. nothing to mark
31511. **comment:** XXX: function properties
label: code-design
31512. prepend regexp to valstack 0 index
31513. act_caller->func may be NULL in some finalization cases, * just treat like we don't know the caller.
31514. convenience
31515. Previous value doesn't need refcount changes because its ownership * is transferred to prev_caller.
31516. * Pop built-ins from stack: they are now INCREF'd and * reachable from the builtins[] array or indirectly * through builtins[].
31517. [obj key value desc]
31518. stmt has non-empty value
31519. -> [... holder name val val ToString(i)]
31520. stage 2: delete configurable entries above target length
31521. When LHS is not register bound, always go through a * temporary. No optimization for top level assignment.
31522. keep key reachable for GC etc; guaranteed not to fail
31523. generate mantissa with a single leading whole number digit
31524. * Complex case conversion helper which decodes a bit-packed conversion * control stream generated by unicode/extract_caseconv.py. The conversion * is very slow because it runs through the conversion data in a linear * fashion to save space (which is why ASCII characters have a special * fast path before arriving here). * * The particular bit counts etc have been determined experimentally to * be small but still sufficient, and must match the Python script * (src/extract_caseconv.py). * * The return value is the case converted codepoint or -1 if the conversion * results in multiple characters (this is useful for regexp Canonicalization * operation). If 'buf' is not NULL, the result codepoint(s) are also * appended to the hbuffer. * * Context and locale specific rules must be checked before consulting * this function.
31525. Stored in duk_heaphdr unused flags.
31526. **comment:** * Augmenting errors at their creation site and their throw site. * * When errors are created, traceback data is added by built-in code * and a user error handler (if defined) can process or replace the * error. Similarly, when errors are thrown, a user error handler * (if defined) can process or replace the error. * * Augmentation and other processing at error creation time is nice * because an error is only created once, but it may be thrown and * rethrown multiple times. User error handler registered for processing * an error at its throw site must be careful to handle rethrowing in * a useful manner. * * Error augmentation may throw an internal error (e.g. alloc error). * * Ecmascript allows throwing any values, so all values cannot be * augmented. Currently, the built-in augmentation at error creation * only augments error values which are Error instances (= have the * built-in Error.prototype in their prototype chain) and are also * extensible. User error handlers have no limitations in this respect.
label: code-design
31527. **comment:** * Emit initializers in sets of maximum max_init_values. * Corner cases such as single value initializers do not have * special handling now. * * Elided elements must not be emitted as 'undefined' values, * because such values would be enumerable (which is incorrect). * Also note that trailing elisions must be reflected in the * length of the final array but cause no elements to be actually * inserted.
label: code-design
31528. * Initialize built-in objects. Current thread must have a valstack * and initialization errors may longjmp, so a setjmp() catch point * must exist.
31529. terminator
31530. DUK_TOK_EQUALSIGN
31531. * Comparison
31532. DUK_JSON_H_INCLUDED
31533. may have side effects
31534. increases key refcount
31535. on first round, skip
31536. **comment:** * Ecmascript specification algorithm and conversion helpers. * * These helpers encapsulate the primitive Ecmascript operation * semantics, and are used by the bytecode executor and the API * (among other places). Note that some primitives are only * implemented as part of the API and have no "internal" helper. * (This is the case when an internal helper would not really be * useful; e.g. the operation is rare, uses value stack heavily, * etc.) * * The operation arguments depend on what is required to implement * the operation: * * - If an operation is simple and stateless, and has no side * effects, it won't take an duk_hthread argument and its * arguments may be duk_tval pointers (which are safe as long * as no side effects take place). * * - If complex coercions are required (e.g. a "ToNumber" coercion) * or errors may be thrown, the operation takes an duk_hthread * argument. This also implies that the operation may have * arbitrary side effects, invalidating any duk_tval pointers. * * - For operations with potential side effects, arguments can be * taken in several ways: * * a) as duk_tval pointers, which makes sense if the "common case" * can be resolved without side effects (e.g. coercion); the * arguments are pushed to the valstack for coercion if * necessary * * b) as duk_tval values * * c) implicitly on value stack top * * d) as indices to the value stack * * Future work: * * - Argument styles may not be the most sensible in every case now. * * - In-place coercions might be useful for several operations, if * in-place coercion is OK for the bytecode executor and the API.
label: code-design
31537. Note: may be triggered even if minimal new_size would not reach the limit, * plan limit accordingly (taking DUK_VALSTACK_GROW_STEP into account).
31538. **comment:** A little tricky string approach to provide the flags string. * This depends on the specific flag values in duk_regexp.h, * which needs to be asserted for. In practice this doesn't * produce more compact code than the easier approach in use.
label: code-design
31539. Note: intentionally use approximations to shave a few instructions: * a_used = old_used (accurate: old_used + 1) * a_size = arr_idx (accurate: arr_idx + 1)
31540. Slot C is used in a non-standard fashion (range of regs), * emitter code has special handling for it (must not set the * "no shuffle" flag).
31541. Note: if atom were to contain e.g. captures, we would need to * re-match the atom to get correct captures. Simply quantifiers * do not allow captures in their atom now, so this is not an issue.
31542. * Criteria for augmenting: * * - augmentation enabled in build (naturally) * - error value internal prototype chain contains the built-in * Error prototype object (i.e. 'val instanceof Error') * * Additional criteria for built-in augmenting: * * - error value is an extensible object

31543. **comment:** XXX: solve into closed form (smaller code)

label: code-design

31544. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * If the final target function cannot be handled by an ecma-to-ecma * call, return to the caller with a return value indicating this case. * The bound chain is resolved and the caller can resume with a plain * function call.

31545. must be signed for loop structure

31546. is_extra

31547. **comment:** * Memory calls: relative to heap, GC interaction, but no error throwing. * * XXX: Currently a mark-and-sweep triggered by memory allocation will run * using the heap->heap_thread. This thread is also used for running * mark-and-sweep finalization; this is not ideal because it breaks the * isolation between multiple global environments. * * Notes: * * - DUK_FREE() is required to ignore NULL and any other possible return * value of a zero-sized alloc/realloc (same as ANSI C free()). * * - There is no DUK_REALLOC_ZEROED because we don't assume to know the * old size. Caller must zero the reallocated memory. * * - DUK_REALLOC_INDIRECT() must be used when a mark-and-sweep triggered * by an allocation failure might invalidate the original 'ptr', thus * causing a realloc retry to use an invalid pointer. Example: we're * reallocating the value stack and a finalizer resizes the same value * stack during mark-and-sweep. The indirect variant requests for the * current location of the pointer being reallocated using a callback * right before every realloc attempt; this circuitous approach is used * to avoid strict aliasing issues in a more straightforward indirect * pointer (void **) approach. Note: the pointer in the storage * location is read but is NOT updated; the caller must do that.

label: code-design

31548. Here 'x' is a Unicode codepoint

31549. [ToUint32(len) array ToUint32(len)] -> [ToUint32(len) array]

31550. Force a resize so that DELETED entries are eliminated. * Another option would be duk_recheck_strtab_size_probe(); * but since that happens on every intern anyway, this whole * check can now be disabled.

31551. * comp_ctx->curr_func is now ready to be converted into an actual * function template.

31552. Potentially truncated, NUL not guaranteed in any case. * The (int_rc < 0) case should not occur in practice.

31553. If not found, resume existence check from Function.prototype. * We can just substitute the value in this case; nothing will * need the original base value (as would be the case with e.g. * setters/getters.

31554. "-123456\0"

31555. Get the original arguments. Note that obj_index may be a relative * index so the stack must have the same top when we use it.

31556. curr value

31557. These are for rate limiting Status notifications and transport peeking.

31558. * Some E5/E5.1 algorithms require that array indices are iterated * in a strictly ascending order. This is the case for e.g. * Array.prototype.forEach() and JSON.stringify() PropertyList * handling. * * To ensure this property for arrays with an array part (and * arbitrary objects too, since e.g. forEach() can be applied * to an array), the caller can request that we sort the keys * here.

31559. DUK_USE_LEXER_SLIDING_WINDOW

31560. DUK_TOK_NULL

31561. * Must be careful to catch errors related to value stack manipulation * and property lookup, not just the call itself.

31562. The file/line arguments are NULL and 0, they're ignored by DUK_ERROR_RAW() * when non-verbose errors are used.

31563. **comment:** 8-byte format could be: * 1111 1111 10xx xxxx [41 bits] * * However, this format would not have a zero bit following the * leading one bits and would not allow 0xFF to be used as an * "invalid utf-8" marker for internal keys. Further, 8-byte * encodings (up to 41 bit code points) are not currently needed.

label: code-design

31564. -> [... obj value]

31565. [key trap_result] -> []

31566. replace in stack

31567. (maybe) truncated

31568. nothing to process

31569. Encode string in small chunks, estimating the maximum expansion so that * there's no need to ensure space while processing the chunk.

31570. * Object.getOwnPropertyDescriptor() (E5 Sections 15.2.3.3, 8.10.4) * * This is an actual function call.

31571. knock back "next temp" to this whenever possible

31572. * Detected a directive

31573. 'for'

31574. complete the millisecond field

31575. 'It is true'

31576. Pre/post inc/dec for register variables, important for loops.

31577. **comment:** Lightfunc name, includes Duktape/C native function pointer, which * can often be used to locate the function from a symbol table. * The name also includes the 16-bit duk_tval flags field because it * includes the magic value. Because a single native function often * provides different functionality depending on the magic value, it * seems reasonably to include it in the name. * * On the other hand, a complicated name increases string table * pressure in low memory environments (but only when function name * is accessed).

label: code-design

31578. **comment:** XXX: if duk_hobject_define_property_internal() was updated * to handle a pre-existing accessor property, this would be * a simple call (like for the ancestor case).

label: code-design

31579. Iteration solution

31580. **comment:** 16 bits would be enough for shared heaphdr flags and duk_hstring * flags. The initial parts of duk_heaphdr_string and duk_heaphdr * must match so changing the flags field size here would be quite * awkward. However, to minimize struct size, we can pack at least * 16 bits of duk_hstring data into the flags field.

label: code-design

31581. * Helper tables

31582. 'this'

31583. **comment:** XXX: could check for e16 == 0 because NULL is guaranteed to * encode to zero.

label: code-design

31584. approximate

31585. Avoid NaN-to-integer coercion as it is compiler specific.

31586. Reset to zero size, keep current limit.

31587. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

31588. callback for indirect reallocs, request for current pointer

31589. [... this]

31590. * Portable 12-byte representation

31591. avoid multiple computations of flags address; bypasses macros

31592. **comment:** Reg/const access macros: these are very footprint and performance sensitive * so modify with care.

label: code-design

31593. must not truncate

31594. prefix matches, lengths matter now

31595. Careful with reachability here: don't pop 'obj' before pushing * proxy target.

31596. * Replace valstack top with case converted version.

31597. Shuffle decision not changed.

31598. * Stack index validation/normalization and getting a stack duk_tval ptr. ** These are called by many API entrypoints so the implementations must be * fast and "inlined". ** There's some repetition because of this; keep the functions in sync.

31599. numeric value of token

31600. [...] constructor arg1 ... argN] -> [retval]

31601. * Rebuild the hash part always from scratch (guaranteed to finish). ** Any resize of hash part requires rehashing. In addition, by rehashing * get rid of any elements marked deleted (DUK__HASH_DELETED) which is critical * to ensuring the hash part never fills up.

31602. temps

31603. 'Object'

31604. not strictly necessary

31605. IsDataDescriptor(desc) == true

31606. POSTFIX EXPRESSION

31607. If argument count is 1 and first argument is a buffer, write the buffer * as raw data into the file without a newline; this allows exact control * over stdout/stderr without an additional entrypoint (useful for now). ** Otherwise current print/alert semantics are to ToString() coerce * arguments, join them with a single space, and append a newline.

31608. * Exposed regexp compilation primitive. ** Sets up a regexp compilation context, and calls duk_parse_disjunction() to do the * actual parsing. Handles generation of the compiled regexp header and the * "boilerplate" capture of the matching substring (save 0 and 1). Also does some * global level regexp checks after recursive compilation has finished. ** An escaped version of the regexp source, suitable for use as a RegExp instance * 'source' property (see E5 Section 15.10.3), is also left on the stack. ** Input stack: [pattern flags] * Output stack: [bytecode escaped_source] (both as strings)

31609. 'Error'

31610. * Flags for __FILE__ / __LINE__ registered into tracedata

31611. Count actually used entry part entries (non-NULL keys).

31612. DUK_TOK_ARSHIFT

31613. **comment:** XXX: possibly incorrect handling of empty expression
label: code-design

31614. used entries + approx 100% -> reset load to 50%

31615. '\x0ffRegbase'

31616. is_decl

31617. but don't allow leading plus

31618. **comment:** * String cache should ideally be at duk_hthread level, but that would * cause string finalization to slow down relative to the number of * threads; string finalization must check the string cache for "weak" * references to the string being finalized to avoid dead pointers. ** Thus, string caches are now at the heap level now.
label: code-design

31619. For manual debugging: instruction count based on executor and * interrupt counter book-keeping. Inspect debug logs to see how * they match up.

31620. [...] holder name val enum obj_key val obj_key]

31621. lj.value1 is already set

31622. number of arguments allocated to regs

31623. Optimized for size.

31624. * Default fatal error handler

31625. named "arithmetic" because result is signed

31626. Default implicit return value.

31627. lookup name from current activation record's functions' registers

31628. custom; implies DUK_HOBJECT_IS_BUFFEROBJECT

31629. max # of key-value pairs initialized in one MPUTOBJ set

31630. unicode code points, window[0] is always next

31631. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.

31632. thread not currently running

31633. Refzero head is still the same. This is the case even if finalizer * inserted more refzero objects; they are inserted to the tail.

31634. [...] key arg1 ... argN]

31635. * GETIDREF: a GetIdentifierReference-like helper. ** Provides a parent traversing lookup and a single level lookup * (for HasBinding). ** Instead of returning the value, returns a bunch of values allowing * the caller to read, write, or delete the binding. Value pointers * are duk_tval pointers which can be mutated directly as long as * refcounts are properly updated. Note that any operation which may * reallocate valstacks or compact objects may invalidate the returned * duk_tval (but not object) pointers, so caller must be very careful. ** If starting environment record 'env' is given, 'act' is ignored. * However, if 'env' is NULL, the caller may identify, in 'act', an * activation which hasn't had its declarative environment initialized * yet. The activation registers are then looked up, and its parent * traversed normally. ** The 'out' structure values are only valid if the function returns * success (non-zero).

31636. Ignore the normalize/validate helper outputs on the value stack, * they're popped automatically.

31637. **comment:** A union is used here as a portable struct size / alignment trick: * by adding a 32-bit or a 64-bit (unused) union member, the size of * the struct is effectively forced to be a multiple of 4 or 8 bytes * (respectively) without increasing the size of the struct unless * necessary.
label: code-design

31638. Chop extra retrvals away / extend with undefined.

31639. MPUTARR emitted by outer loop

31640. **comment:** XXX: fastint?
label: code-design

31641. compact the exports table

31642. Code is not accepted before the first case/default clause

31643. Process one debug message. Automatically restore value stack top to its * entry value, so that individual message handlers don't need exact value * stack handling which is convenient.

31644. -> [...] key val replacer holder key val]

31645. Because new_size != 0, if condition doesn't need to be * (new_valstack != NULL || new_size == 0).

31646. No debugger support.

31647. '<'

31648. treat like property not found

31649. Bound functions don't have all properties so we'd either need to * lookup the non-bound target function or reject bound functions. * For now, bound functions are rejected.

31650. Carry is detected based on wrapping which relies on exact 32-bit * types.

31651. eat (first) input slash

31652. value resides in a register or constant

31653. **comment:** "get" and "set" are tokens but NOT ReservedWords. They are currently * parsed and identifiers and these defines are actually now unused.
label: code-design

31654. Overflow, relevant mainly when listlen is 16 bits.

31655. [...] val] --> [...] enum]

31656. E5.1 standard behavior when deleteCount is not given would be * to treat it just like if 'undefined' was given, which coerces * ultimately to 0. Real world behavior is to splice to the end * of array, see test-bi-array-proto-splice-no-delcount.js.

31657. case 0: nop

31658. **comment:** XXX: the duk_hobject_enum.c stack APIs should be reworked
label: code-design

31659. Error: try coercing error to string once.

31660. for resizing of array part, use slow path

31661. y == -Infinity

31662. Initialization and finalization (compaction), converting to other types.
31663. must match exactly the number of internal properties inserted to enumerator
31664. required
31665. + spare
31666. All element accessors are host endian now (driven by TypedArray spec).
31667. Push an arbitrary duk_tval to the stack, coerce it to string, and return * both a duk_hstring pointer and an array index (or DUK__NO_ARRAY_INDEX).
31668. must "goto restart_execution", e.g. breakpoints changed
31669. Avoid side effects that might disturb curr.e_idx until * we're done editing the slot.
31670. 23: getUTCSeconds
31671. no need to normalize
31672. * Windows Date providers ** Platform specific links: ** - [http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473(v=vs.85).aspx)
31673. **comment:** XXX: Change 'anon' handling here too, to use empty string for anonymous functions?
 label: code-design
31674. 'this binding' is just under bottom
31675. Default exports table
31676. **comment:** XXX: macro smaller than call?
 label: code-design
31677. XXX: assert? compiler is responsible for this never happening
31678. pc of label statement: * pc+1: break jump site * pc+2: continue jump site
31679. 'require'
31680. * Assertion helpers
31681. multiple codepoint conversion or non-ASCII mapped to ASCII * --> leave as is.
31682. Computation must not wrap, only srclen + 3 is at risk of * wrapping because after that the number gets smaller. * This limit works for 32-bit size_t: * 0x100000000
 - 3 - 1 = 4294967292
31683. 19: getUTCMilliseconds
31684. Inspect a property using a virtual index into a conceptual property list * consisting of (1) all array part items from [0,a_size[(even when above * .length) and (2)
 all entry part items from [0,e_next[. Unused slots are * indicated using dvalue 'unused'.
31685. duk_get_min_grow_e() is always >= 1
31686. * Reachable object, keep
31687. 'eval'
31688. Array part may be larger than 'length'; if so, iterate * only up to array 'length'.
31689. regCatch+0 and regCatch+1 are reserved for TRYCATCH
31690. resolved id: require(id) must return this same module
31691. We rely on a few object flag / class number relationships here, * assert for them.
31692. default case control flow patchup; note that if pc_prevcase < 0 * (i.e. no case clauses), control enters default case automatically.
31693. DUK_HEAP_H_INCLUDED
31694. note: original, uncoerced base
31695. If len <= 1, middle will be 0 and for-loop bails out * immediately (0 < 0 -> false).
31696. DUK_OP_NONE marks a 'plain' assignment
31697. resume execution from pc_base or pc_base+1 (points to 'func' bytecode, stable pointer)
31698. Stack size decreases.
31699. heap->strs: not worth dumping
31700. -> [... closure template env funcname closure]
31701. step 12
31702. Section 7.8.3 (note): NumericLiteral must be followed by something other than * IdentifierStart or DecimalDigit.
31703. **comment:** XXX: duk_ssize_t
 label: code-design
31704. DUK_ERR_UNCAUGHT_ERROR: no macros needed
31705. -> [ToObject(this) item1 ... itemN arr]
31706. C call site gets blamed next, unless flagged not to do so. * XXX: file/line is disabled in minimal builds, so disable this * too when appropriate.
31707. * Debug dump type sizes
31708. needed for line number tracking (becomes prev_token.start_line)
31709. don't coerce yet to a plain value (variant 3 needs special handling)
31710. **comment:** XXX: opcode specific assertions on when consts are allowed
 label: code-design
31711. Read value pushed on stack.
31712. * When resizing the valstack, a mark-and-sweep may be triggered for * the allocation of the new valstack. If the mark-and-sweep needs * to use our thread for
 something, it may cause *the same valstack* * to be resized recursively. This happens e.g. when mark-and-sweep * finalizers are called. This is taken into account
 carefully in * duk_resize_valstack(). * * 'new_size' is known to be <= valstack_max, which ensures that * size_t and pointer arithmetic won't wrap in
 duk_resize_valstack().
31713. 'null' enumerator case -> behave as with an empty enumerator
31714. Object literal set/get functions have a name (property * name) but must not have a lexical name binding, see * test-bug-getset-func-name.js.
31715. Current compiler state (if any), used for augmenting SyntaxErrors.
31716. push to stack
31717. Note: if DecimalLiteral starts with a '0', it can only be * followed by a period or an exponent indicator which starts * with 'e' or 'E'. Hence the if-check above
 ensures that * OctalIntegerLiteral is the only valid NumericLiteral * alternative at this point (even if y is, say, '9').
31718. 'catch'
31719. * Special cases like NaN and +/- Infinity are handled explicitly * because a plain C coercion from double to int handles these cases * in undesirable ways. For
 instance, NaN may coerce to INT_MIN * (not zero), and INT_MAX + 1 may coerce to INT_MIN (not INT_MAX). * * This double-to-int coercion differs from
 ToInt32() because it * has a finite range (ToInt32() allows e.g. +/- Infinity). It * also differs from ToInt32() because the INT_MIN/INT_MAX clamping *
 depends on the size of the int type on the platform. In particular, * on platforms with a 64-bit int type, the full range is allowed.
31720. element size shift: * 0 = u8/i8 * 1 = u16/i16 * 2 = u32/i32/float * 3 = double
31721. curr_pc synced back above
31722. hash size approx. 1.25 times entries size
31723. Write in little endian
31724. source starts after dest ends
31725. **comment:** * Debugging macro calls.
 label: code-design
31726. DUK_USE_VERBOSE_ERRORS
31727. 0xxx xxxx -> fast path
31728. * Ecmascript modulus (%) does not match IEEE 754 "remainder" * operation (implemented by remainder() in C99) but does seem * to match ANSI C fmod(). * *
 Compare E5 Section 11.5.3 and "man fmod".
31729. unsigned shift
31730. Not a very good provider: only full seconds are available.
31731. 110x xxxx 10xx xxxx [11 bits]
31732. Note: errors are augmented when they are created, not * when they are thrown. So, don't augment here, it would * break re-throwing for instance.
31733. **comment:** Not necessary to unwind catchstack: return handling will * do it. The finally flag of 'cat' is no longer set. The * catch flag may be set, but it's not
 checked by return handling.

label: code-design

31734. ensure value is 1 or 0 (not other non-zero)
31735. Accept plain buffer values like array initializers * (new in Duktape 1.4.0).
31736. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).
31737. Indicate function type in the function body using a dummy * directive.
31738. 'ObjEnv'
31739. lj.value1 already set
31740. millisec -> 100ns units since jan 1, 1970
31741. **comment:** * Note: must use indirect variant of DUK_REALLOC() because underlying * pointer may be changed by mark-and-sweep.
label: code-design
31742. DUK_TOK_SWITCH
31743. class Object, extensible
31744. idx_end
31745. _Source
31746. 1 1 0 <8 bits>
31747. **comment:** two casts to avoid gcc warning: "warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]"
label: code-design
31748. minimum new length is highest_arr_idx + 1
31749. DUK_TOK_RBRACKET
31750. property is configurable and
31751. * Matching complete, create result array or return a 'null'. Update lastIndex * if necessary. See E5 Section 15.10.6.2. ** Because lastIndex is a character (not byte) offset, we need the character * length of the match which we conveniently get as a side effect of interning * the matching substring (0th index of result array). ** saved[0] start pointer (~ byte offset) of current match * saved[1] end pointer (~ byte offset) of current match (exclusive) * char_offset start character offset of current match (-> .index of result) * char_end_offset end character offset (computed below)
31752. **comment:** XXX: Refactor key coercion so that it's only called once. It can't * be trivially lifted here because the object must be type checked * first.
label: code-design
31753. Caller must ensure 'tv' is indeed a double and not a fastint!
31754. Caller has already eaten the first character ('|') which we don't need.
31755. * String scanning helpers ** All bytes other than UTF-8 continuation bytes ([0x80,0xbf]) are * considered to contribute a character. This must match how string * character length is computed.
31756. [... func this | arg1 ... argN] ('this' must precede new bottom)
31757. structs from duk_forwdecl.h
31758. assumed to also be PC of "LABEL"
31759. Check that prev/next links are consistent: if e.g. h->prev is != NULL, * h->prev->next should point back to h.
31760. {'_undef':true}'
31761. DUK_TOK_ADD_EQ
31762. roughly 1.0 kB -> but rounds up to DUK_VALSTACK_GROW_STEP in practice
31763. remove the original 'template' atom
31764. The ANSI C pow() semantics differ from Ecmascript. ** E.g. when x==1 and y is +/- infinite, the Ecmascript required * result is NaN, while at least Linux pow() returns 1.
31765. * Heap native function representation.
31766. Note: DUK_HEAP_HAS_REFZERO_FREE_RUNNING(heap) may be true; a refcount * finalizer may trigger a mark-and-sweep.
31767. 'Invalid Date'
31768. dynamic
31769. **comment:** * Node.js Buffer.concat()
label: code-design
31770. **comment:** XXX: substring in-place at idx_place?
label: code-design
31771. * Object's finalizer was executed on last round, and * object has been happily rescued.
31772. embed: 0 or 1 (false or true)
31773. Labels can be used for iteration statements but also for other statements, * in particular a label can be used for a block statement. All cases of a * named label accept a 'break' so that flag is set here. Iteration statements * also allow 'continue', so that flag is updated when we figure out the * statement type.
31774. string 2 of token (borrowed, stored to ctx->slot2_idx)
31775. UNARY EXPRESSIONS
31776. * Not found (even in global object). ** In non-strict mode this is a silent SUCCESS (!), see E5 Section 11.4.1, * step 3.b. In strict mode this case is a compile time SyntaxError so * we should not come here.
31777. should actually never happen, but check anyway
31778. input offset for window leading edge (not window[0])
31779. * Init object properties * Properties should be added in decreasing order of access frequency. * (Not very critical for function templates.)
31780. [... fallback constructor fallback(this) arg1 ... argN]; * Note: idx_cons points to first 'fallback', not 'constructor'.
31781. **comment:** * Canonicalize a range, generating result ranges as necessary. * Needs to exhaustively scan the entire range (at most 65536 * code points). If 'direct' is set, caller (lexer) has ensured * that the range is already canonicalization compatible (this * is used to avoid unnecessary canonicalization of built-in * ranges like \W, which are not affected by canonicalization). ** NOTE: here is one place where we don't want to support chars * outside the BMP, because the exhaustive search would be * massively larger.
label: code-design
31782. continuation byte
31783. Note: the realloc may have triggered a mark-and-sweep which may * have resized our valstack internally. However, the mark-and-sweep * MUST NOT leave the stack bottom/top in a different state. Particular * assumptions and facts: * - The thr->valstack pointer may be different after realloc, * and the offset between thr->valstack_end <-> thr->valstack * may have changed. * - The offset between thr->valstack_bottom <-> thr->valstack * and thr->valstack_top <-> thr->valstack MUST NOT have changed, * because mark-and-sweep must adhere to a strict stack policy. * In other words, logical bottom and top MUST NOT have changed. * - All values above the top are unreachable but are initialized * to UNDEFINED, up to the post-realloc valstack_end. * 'old_end_offset' must be computed after realloc to be correct.
31784. Note: hstring is in heap but has refcount zero and is not strongly reachable. * Caller should increase refcount and make the hstring reachable before any * operations which require allocation (and possible gc).
31785. **comment:** * "name" is a non-standard property found in at least V8, Rhino, smjs. * For Rhino and smjs it is non-writable, non-enumerable, and non-configurable; * for V8 it is writable, non-enumerable, non-configurable. It is also defined * for an anonymous function expression in which case the value is an empty string. * We could also leave name 'undefined' for anonymous functions but that would * differ from behavior of other engines, so use an empty string. ** XXX: make optional? costs something per function.
label: code-design
31786. **comment:** * duk_hbuffer operations such as resizing and inserting/appending data to * a dynamic buffer. ** Append operations append to the end of the buffer and they are relatively * efficient: the buffer is grown with a "spare" part relative to the buffer * size to minimize reallocations. Insert operations need to move existing * data forward in the buffer with memmove() and are not very efficient. * They are used e.g. by the regexp compiler to "backpatch" regexp bytecode.
label: code-design
31787. steps 11.c.ii.1 - 11.c.ii.4, but our internal book-keeping * differs from the reference model
31788. ASCII fast path
31789. Most input bytes go through here.
31790. -> [... holder name val]

31791. Assume that year, month, day are all coerced to whole numbers. * They may also be NaN or infinity, in which case this function * must return NaN or infinity to ensure time value becomes NaN. * If 'day' is NaN, the final return will end up returning a NaN, * so it doesn't need to be checked here.

31792. useful for debugging

31793. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()! * * A conservative approach would be to use duk__ivalue_totempconst() * for 'left'. However, allowing a reg-bound variable seems safe here * and is nice because "foo.bar" is a common expression. If the ivalue * is used in an expression a GETPROP will occur before any changes to * the base value can occur. If the ivalue is used as an assignment * LHS, the assignment code will ensure the base value is safe from * RHS mutation.

31794. We should never exit the loop above.

31795. may have FINALIZED

31796. LAYOUT 3

31797. **comment:** unused

label: code-design

31798. these computations are guaranteed to be exact for the valid * E5 time value range, assuming milliseconds without fractions.

31799. 'Int32Array'

31800. an error handler (user callback to augment/replace error) is running

31801. without refcounts

31802. arbitrary marker, outside valid exp range

31803. extra -1 for buffer

31804. For invalid characters the value -1 gets extended to * at least 16 bits. If either nybble is invalid, the * resulting 't' will be < 0.

31805. required to guarantee success of rehashing, * intentionally use unadjusted new_e_size

31806. XXX: A fast path for usual integers would be useful when * fastint support is not enabled.

31807. -> [... ToObject(this) ToUInt32(length) final_len final_len]

31808. [...val] -> [...]

31809. allow empty expression

31810. [... closure template name]

31811. # total registers target function wants on entry (< 0 => "as is")

31812. **comment:** unnecessary shift for last byte

label: code-design

31813. varenv remains reachable through 'obj'

31814. Special coercion for Uint8ClampedArray.

31815. valstack slot for temp buffer

31816. * Node.js Buffer: constructor

31817. this optimization is important to handle negative literals * (which are not directly provided by the lexical grammar)

31818. Nothing worked -> not equal.

31819. E5 Section 11.13.1 (and others for other assignments), step 4.

31820. Note: unshift() may operate on indices above unsigned 32-bit range * and the final length may be >= 2**32. However, we restrict the * final result to 32-bit range for practicality.

31821. String and buffer enumeration behavior is identical now, * so use shared handler.

31822. 37x32 = 1184 bits

31823. heap->dbg_detached_cb: keep

31824. This check used to be for (t < 0) but on some platforms * time_t is unsigned and apparently the proper way to detect * an mktime() error return is the cast above. See e.g.: * <http://pubs.opengroup.org/onlinepubs/009695299/functions/mktime.html>

31825. **comment:** Fast variants, inline refcount operations except for refzero handling. * Can be used explicitly when speed is always more important than size. * For a good compiler and a single file build, these are basically the * same as a forced inline.

label: code-design

31826. [... key] -> [...]

31827. * Use current token to decide whether a RegExp can follow. * * We can use either 't' or 't_nores'; the latter would not * recognize keywords. Some keywords can be followed by a * RegExp (e.g. "return"), so using 't' is better. This is * not trivial, see doc/compiler.rst.

31828. Khronos/ES6 requires zeroing even when DUK_USE_ZERO_BUFFER_DATA * is not set.

31829. * Refcount memory freeing loop. * * Frees objects in the refzero_pending list until the list becomes * empty. When an object is freed, its references get decref'd and * may cause further objects to be queued for freeing. * * This could be expanded to allow incremental freeing: just bail out * early and resume at a future alloc/decref/refzero.

31830. Note: 'fast' breaks will jump to pc_label_site + 1, which will * then jump here. The double jump will be eliminated by a * peephole pass, resulting in an optimal jump here. The label * site jumps will remain in bytecode and will waste code size.

31831. **comment:** * Prototype walking starting from 'env'. * * ('act' is not needed anywhere here.)

label: code-design

31832. mark range 'direct', bypass canonicalization (see Wiki)

31833. Do debugger forwarding before raw() because the raw() function * doesn't get the log level right now.

31834. estimate year upwards (towards positive infinity), then back down; * two iterations should be enough

31835. return module.exports

31836. * Parsing an expression starting with 'new' is tricky because * there are multiple possible productions deriving from * LeftHandSideExpression which begin with 'new'. * * We currently resort to one-token lookahead to distinguish the * cases. Hopefully this is correct. The binding power must be * such that parsing ends at an LPAREN (CallExpression) but not at * a PERIOD or LBRACKET (MemberExpression). * * See doc/compiler.rst for discussion on the parsing approach, * and testcases/test-dev-new.js for a bunch of documented tests.

31837. exrptop is the top level variant which resets nud/led counts

31838. control props

31839. push value to stack

31840. * shift()

31841. k+argCount-1; note that may be above 32-bit range

31842. * Assert macro: failure causes panic.

31843. **comment:** Similar workaround for INFINITY.

label: code-design

31844. * Fallback marking handler if recursion limit is reached. * * Iterates 'temproots' until recursion limit is no longer hit. Note * that temproots may reside either in heap allocated list or the * refzero work list. This is a slow scan, but guarantees that we * finish with a bounded C stack. * * Note that nodes may have been marked as temproots before this * scan begun, OR they may have been marked during the scan (as * we process nodes recursively also during the scan). This is * intended behavior.

31845. **comment:** If number would need zero padding (for whole number part), use * exponential format instead. E.g. if input number is 12300, 3 * digits are generated ("123"), output "1.23e+4" instead of "12300". * Used for toPrecision().

label: code-design

31846. number

31847. Must be element size multiple from * start offset to end of buffer.

31848. empty ("") is allowed in some formats (e.g. Number()), as zero

31849. slow path is shared

31850. * If matching with a regexp: * - non-global RegExp: lastIndex not touched on a match, zeroed * on a non-match * - global RegExp: on match, lastIndex will be updated by regexp * executor to point to next char after the matching part (so that * characters in the matching part are not matched again) * * If matching with a string: * - always non-global match, find first occurrence * * We need: * - The character offset of start-of-match for the replacer function * - The byte offsets for start-of-match and end-of-match to implement * the replacement values \$\$, \$`, and \$', and to copy non-matching * input string portions (including header and

trailer) verbatim. ** NOTE: the E5.1 specification is a bit vague how the RegExp should * behave in the replacement process; e.g. is matching done first for * all matches (in the global RegExp case) before any replacer calls * are made? See: test-bi-string-proto-replace.js for discussion.

31851. -> [... func]

31852. * Various defines and file specific helper macros

31853. no net exponent

31854. **comment:** integer, but may be +/- Infinite, +/- zero (not NaN, though)

label: code-design

31855. 11 bits

31856. **comment:** Lookup active label information. Break/continue distinction is necessary to handle switch * statement related labels correctly: a switch will only catch a 'break', not a 'continue'. ** An explicit label cannot appear multiple times in the active set, but empty labels (unlabelled * iteration and switch statements) can. A break will match the closest unlabelled or labelled * statement. A continue will match the closest unlabelled or labelled iteration statement. It is * a syntax error if a continue matches a labelled switch statement; because an explicit label cannot * be duplicated, the continue cannot match any valid label outside the switch. ** A side effect of these rules is that a LABEL statement related to a switch should never actually * catch a continue abrupt completion at run-time. Hence an INVALID opcode can be placed in the * continue slot of the switch's LABEL statement.

label: code-design

31857. get before side effects

31858. DUK_HSTRING_H_INCLUDED

31859. avoid empty label at the end of a compound statement

31860. no specific action, resume normal execution

31861. for manual torture testing: tight allocation, useful with valgrind

31862. XXX: skip count_free w/o debug?

31863. everything except object stay as is

31864. identifiers and environment handling

31865. back to fast loop

31866. DUK_USE_JX || DUK_USE_JC

31867. Run fake finalizer. Avoid creating new refzero queue entries * so that we are not forced into a forever loop.

31868. In some cases it may be that lo > hi, or hi < 0; these * degenerate cases happen e.g. for empty arrays, and in * recursion leaves.

31869. is resume, not a tail call

31870. * Basic stack manipulation: swap, dup, insert, replace, etc

31871. **comment:** Note: nargs (and nregs) may be negative for a native, * function, which indicates that the function wants the * input stack "as is" (i.e. handles "vararg" arguments).

label: code-design

31872. * Indirect magic value lookup for Date methods. ** Date methods don't put their control flags into the function magic value * because they wouldn't fit into a LIGHTFUNC's magic field. Instead, the * magic value is set to an index pointing to the array of control flags * below. ** This must be kept in strict sync with genbuiltins.py!

31873. strings may have inner refs (extdata) in some cases

31874. class masks

31875. keep label catcher

31876. state updated, restart bytecode execution

31877. Flags for duk_js_equals_helper().

31878. [... func this <some bound args> arg1 ... argN _Args]

31879. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module result(ignored)]

31880. DUK_USE_TAILCALL

31881. **comment:** should never be called

label: code-design

31882. denormal

31883. -> loop body

31884. note: this updates refcounts

31885. avoid calling at end of input, will DUK_ERROR (above check suffices to avoid this)

31886. t1 <- (+ r m+)

31887. denormal or zero

31888. mark-and-sweep control

31889. A 'return' statement is only allowed inside an actual function body, * not as part of eval or global code.

31890. tailcall cannot be flagged to resume calls, and a * previous frame must exist

31891. DUK_TOK_LET

31892. Remove a slice from inside buffer.

31893. valstack slot for 2nd token value

31894. one-based -> zero-based

31895. Set .buffer

31896. terminate

31897. max size: radix=2 + sign

31898. remaining actual steps are carried out if standard DefineOwnProperty succeeds

31899. **comment:** The prev/next pointers of the removed duk_heaphdr are left as garbage. * It's up to the caller to ensure they're written before inserting the * object back.

label: code-design

31900. {"_func":true}'

31901. **comment:** XXX: support unary arithmetic ivalue (useful?)

label: code-design

31902. E5 Section 15.5.5.2

31903. True if slice is full, i.e. offset is zero and length covers the entire * buffer. This status may change independently of the duk_hbufferobject if * the underlying buffer is dynamic and changes without the hbufferobject * being changed.

31904. avoid calling write callback in detach1()

31905. http://en.wikipedia.org/wiki/Replacement_character#Replacement_character

31906. -> [Object res]

31907. * Property attribute defaults are defined in E5 Section 15 (first * few pages); there is a default for all properties and a special * default for 'length' properties. Variation from the defaults is * signaled using a single flag bit in the bitstream.

31908. IdentifierStart production with Letter and ASCII excluded

31909. **comment:** * DecimalLiteral, HexIntegerLiteral, OctalIntegerLiteral * "pre-parsing", followed by an actual, accurate parser step. ** Note: the leading sign character ('+' or '-') is -not- part of * the production in E5 grammar, and that the a DecimalLiteral * starting with a '0' must be followed by a non-digit. Leading * zeroes are syntax errors and must be checked for. ** XXX: the two step parsing process is quite awkward, it would * be more straightforward to allow numconv to parse the longest * valid prefix (it already does that, it only needs to indicate * where the input ended). However, the lexer decodes characters * using a lookup window, so this is not a trivial change.

label: code-design

31910. * The escapes are same as outside a character class, except that \b has a * different meaning, and \B and backreferences are prohibited (see E5 * Section 15.10.2.19). However, it's difficult to share code because we * handle e.g. "\n" very differently: here we generate a single character * range for it.

31911. **comment:** Ensure there are no stale active breakpoint pointers. * Breakpoint list is currently kept - we could empty it * here but we'd need to handle refcounts correctly, and * we'd need a 'thr' reference for that. ** XXX: clear breakpoint on either attach or detach?

label: code-design

31912. end switch

31913. DUK_TOK_LE
31914. XXX: Currently no inspection of threads, e.g. value stack, call * stack, catch stack, etc.
31915. [... v1(func) v2(pc+flags)]
31916. Matches both +0 and -0 on purpose.
31917. Note: ToPrimitive(object,hint) == [[DefaultValue]](object,hint), * so use [[DefaultValue]] directly.
31918. The code for writing reg_temps + 0 to the left hand side has already * been emitted.
31919. start line of token (first char)
31920. 10: getMonth
31921. If caller is global/eval code, 'caller' should be set to * 'null'. * * XXX: there is no exotic flag to infer this correctly now. * The NEWENV flag is used now which works as intended for * everything (global code, non-strict eval code, and functions) * except strict eval code. Bound functions are never an issue * because 'func' has been resolved to a non-bound function.
31922. x <- y
31923. eat 'var'
31924. cannot have >4G captures
31925. **comment:** not caught by current thread, thread terminates (yield error to resumer); * note that this may cause a cascade if the resumer terminates with an uncaught * exception etc (this is OK, but needs careful testing)
label: code-design
31926. normalize NaN which may not match our canonical internal NaN
31927. Recompute argument count: bound function handling may have shifted.
31928. [...] key val [...] -> [...]
31929. minval
31930. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.
31931. DUK_TOK_ARSHIFT_EQ
31932. Slightly smaller code without explicit flag, but explicit flag * is very useful when 'cлен' is dropped.
31933. Here again we parse bytes, and non-ASCII UTF-8 will cause end of * parsing (which is correct except if there are non-shortest encodings). * There is also no need to check explicitly for end of input buffer as * the input is NUL padded and NUL will exit the parsing loop. * * Because no unescaping takes place, we can just scan to the end of the * plain string and intern from the input buffer.
31934. **comment:** XXX: merge this with duk_js_call.c, as this function implements * core semantics (or perhaps merge the two files altogether).
label: code-design
31935. * New length lower than old length => delete elements, then * update length. * * Note: even though a bunch of elements have been deleted, the 'desc' is * still valid as properties haven't been resized (and entries compacted).
31936. * Program code (global and eval code) has an implicit return value * from the last statement value (e.g. eval("1; 2+3;") returns 3). * This is not the case with functions. If implicit statement return * value is requested, all statements are coerced to a register * allocated here, and used in the implicit return statement below.
31937. * Pass 1
31938. Use unsigned arithmetic to optimize comparison.
31939. type and control flags, label number
31940. DUK_TOK_RSHIFT_EQ
31941. probe steps (see genhashsizes.py), currently assumed to be 32 entries long * (DUK_UTIL_GET_HASH_PROBE_STEP macro).
31942. [...] enum_target res trap_result]
31943. 1111 1110 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [36 bits]
31944. * Conversions and coercions * * The conversion/coercions are in-place operations on the value stack. * Some operations are implemented here directly, while others call a * helper in duk_js_ops.c after validating arguments.
31945. **comment:** Select a thread for mark-and-sweep use. * * XXX: This needs to change later.
label: code-design
31946. Automatic filename: 'eval' or 'input'.
31947. cumulative opcode execution count (overflows are OK)
31948. Fast path for fastints
31949. **comment:** XXX: any chance of merging these three similar but still slightly * different algorithms so that footprint would be reduced?
label: code-design
31950. comp_ctx->lex has been pre-initialized by caller: it has been * zeroed and input/input_length has been set.
31951. 'input'
31952. countdown state
31953. switch
31954. * Return to bytecode executor, which will resume execution from * the topmost activation.
31955. Shared exit handling for object/array serialization.
31956. input string scan
31957. as elements
31958. * Rewind lexer. * * duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp * literal" mode with current strictness. * * curr_token line number info should be initialized for pass 2 before * generating prologue, to ensure prologue bytecode gets nice line numbers.
31959. Shared helper for Object.getOwnPropertyNames() and Object.keys(). * Magic: 0=getOwnPropertyNames, 1=Object.keys.
31960. no need to unwind
31961. When looking for .fileName/.lineNumber, blame first * function which has a .fileName.
31962. **comment:** XXX: if attempt to push beyond allocated valstack, this double fault * handling fails miserably. We should really write the double error * directly to thr->heap->lj.value1 and avoid valstack use entirely.
label: code-design
31963. Strict functions don't get their 'caller' updated.
31964. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined fresh_require exports module mod_func]
31965. currently nop
31966. [...] closure template]
31967. * A tiny random number generator. * * Currently used for Math.random(). * * <http://www.woodmann.com/forum/archive/index.php/t-3100.html>
31968. 'func' on stack (borrowed reference)
31969. **comment:** * "prototype" is, by default, a fresh object with the "constructor" * property. * * Note that this creates a circular reference for every function * instance (closure) which prevents refcount-based collection of * function instances. * * XXX: Try to avoid creating the default prototype object, because * many functions are not used as constructors and the default * prototype is unnecessary. Perhaps it could be created on-demand * when it is first accessed?
label: code-design
31970. -> [...] val this]
31971. **comment:** code emission temporary
label: code-design
31972. * TypedArray.prototype.set() * * TypedArray set() is pretty interesting to implement because: * * - The source argument may be a plain array or a typedarray. If the * source is a TypedArray, values are decoded and re-encoded into the * target (not as a plain byte copy). This may happen even when the * element byte size is the same, e.g. integer values may be re-encoded * into floats. * * - Source and target may refer to the same underlying buffer, so that * the set() operation may overlap. The specification requires that this * must work as if a copy was made before the operation. Note that this * is NOT a simple memmove() situation because the source and target * byte sizes may be different -- e.g. a 4-byte source (Int8Array) may * expand to a 16-byte target (Uint32Array) so that the target overlaps * the source both from beginning and the end (unlike in typical memmove). * * - Even if 'buf' pointers of the source and target differ, there's no * guarantee that their memory areas don't overlap. This may be the * case with external buffers. * * Even so, it is nice to optimize for the common case: * * - Source and target separate buffers or non-overlapping. * * - Source and target have a compatible type so that a plain byte copy * is possible. Note that while e.g. uint8 and int8 are compatible * (coercion one way or another doesn't change the byte representation), * e.g. int8 and uint8clamped are NOT compatible when writing int8 * values into uint8clamped typedarray (-1 would clamp to 0 for instance). * * See test-bi-typedarray-proto-set.js.

31973. (quotient-remainder (* r B) s) using a dummy subtraction loop
31974. Note: either pointer may be NULL (at entry), so don't assert
31975. String-number-buffer/object -> coerce object to primitive (apparently without hint), then try again.
31976. * Property handling ** The API exposes only the most common property handling functions. * The caller can invoke Ecmascript built-ins for full control (e.g. * defineProperty, getOwnPropertyDescriptor).
31977. Format of magic, bits: * 0...1: elem size shift (0-3) * 2...5: elem type (DUK_HBUFFEROBJECT_ELEM_xxx)
31978. **comment:** Slow gathering of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. Handling * of negative numbers is a bit non-obvious in both cases.
label: code-design
31979. Built-in 'name' is not writable by default. Function '.name' * is writable to allow user code to set a '.name' on a native * function.
31980. reuse last emitted atom for remaining 'infinite' quantifier
31981. TypeError if rval is not an object (or lightfunc which should behave * like a Function instance).
31982. * Dragon4 slow path digit generation.
31983. Rule control flags.
31984. USE_PROP_HASH_PART
31985. don't compact objects; needed during object property allocation resize
31986. r <- (remainder (* r B) s)
31987. [... error arr]
31988. String is an own (virtual) property of a lightfunc.
31989. 16: getHours
31990. 'debugger'
31991. * String manipulation
31992. fmod() return value has same sign as input (negative) so * the result here will be in the range [-2,0], 1 indicates * odd. If x is -Infinity, NaN is returned and the odd check * always concludes "not odd" which results in desired outcome.
31993. DUK_USE_DEBUGGER_INSPECT
31994. Lookup 'key' from arguments internal 'map', perform a variable write if mapped. * Used in E5 Section 10.6 algorithm for [[DefineOwnProperty]] (used by [[Put]]). * Assumes stack top contains 'put' value (which is NOT popped).
31995. * Macro for object validity check ** Assert for currently guaranteed relations between flags, for instance.
31996. **comment:** Process debug messages until we are no longer paused.
label: code-design
31997. * Entries part
31998. [in_val in_val]
31999. DUK_TOK_LBRACKET
32000. 36 bits
32001. for zero size allocations NULL is allowed
32002. -> [... obj key value]
32003. * Some change(s) need to be made. Steps 7-11.
32004. -1 = error, -2 = allowed whitespace, -3 = padding ('='), 0...63 decoded bytes
32005. duk_new() will call the constructor using duk_handle_call(). * A constructor call prevents a yield from inside the constructor, * even if the constructor is an Ecmascript function.
32006. Select a safe loop count where no output checks are * needed assuming we won't encounter escapes. Input * bound checks are not necessary as a NUL (guaranteed) * will cause a SyntaxError before we read out of bounds.
32007. end of diff run
32008. * Buffer
32009. Note: ctx.steps is intentionally not reset, it applies to the entire unanchored match
32010. **comment:** XXX: use the pointer as a seed for now: mix in time at least
label: code-design
32011. XXX: stringtable emergency compaction?
32012. thr->ptr_curr_pc is set by bytecode executor early on entry
32013. Allow leading plus sign
32014. if slice not fully valid, treat as error
32015. Now two entries in the same slot, alloc list
32016. * The loop below must avoid issues with potential callstack * reallocations. A resize (and other side effects) may happen * e.g. due to finalizer/errhandler calls caused by a refzero or * mark-and-sweep. Arbitrary finalizers may run, because when * an environment record is refzero'd, it may refer to arbitrary * values which also become refzero'd. ** So, the pointer 'p' is re-looked-up below whenever a side effect * might have changed it.
32017. thr->heap->j.value1 is already the value to throw
32018. Trailing whitespace has been eaten by duk_dec_value(), so if * we're not at end of input here, it's a SyntaxError.
32019. If user callback did not return source code, module loading * is finished (user callback initialized exports table directly).
32020. **comment:** * Emit a final RETURN. ** It would be nice to avoid emitting an unnecessary "return" opcode * if the current PC is not reachable. However, this cannot be reliably * detected; even if the previous instruction is an unconditional jump, * there may be a previous jump which jumps to current PC (which is the * case for iteration and conditional statements, for instance).
label: code-design
32021. * Because buffer values are often looped over, a number fast path * is important.
32022. starting line number
32023. x <- x - y, use t as temp
32024. EQUALITY EXPRESSION
32025. * Unicode support tables automatically generated during build.
32026. return thread
32027. **comment:** No catch variable, e.g. a try-finally; replace LDCONST with * NOP to avoid a bogus LDCONST.
label: code-design
32028. 0x08583b00
32029. Signed shift, named "arithmetic" (asl) because the result * is signed, e.g. 4294967295 << 1 -> -2. Note that result * must be masked.
32030. fmax() with args -0 and +0 is not guaranteed to return * +0 as Ecmascript requires.
32031. not a wordchar
32032. XXX: a "first" flag would suffice
32033. reg count
32034. Buffer used for generated digits, values are in the range [0,B-1].
32035. assume memset with zero size is OK
32036. If ctx->offset >= ctx->length, we "shift zeroes in" * instead of croaking.
32037. Define own property without inheritance looks and such. This differs from * [[DefineOwnProperty]] because special behaviors (like Array 'length') are * not invoked by this method. The caller must be careful to invoke any such * behaviors if necessary.
32038. Parse a number, other than NaN or +/- Infinity
32039. Must be a "string object", i.e. class "String"
32040. no decref needed for a number
32041. [... | (crud)]
32042. x <- y - z, require x >= y => z >= 0, i.e. y >= z
32043. 26 bits
32044. Used to support single-byte stream lookahead.

32045. Filename/line from C macros (`__FILE__`, `__LINE__`) are added as an * entry with a special format: (string, number). The number contains * the line and flags.
32046. -> [ToObject(this)]
32047. Buffer sizes for some UNIX calls. Larger than strictly necessary * to avoid Valgrind errors.
32048. '\Udeadbeef'
32049. * every(), some(), forEach(), map(), filter()
32050. Test without volatiles
32051. Lexer codepoint with additional info like offset/line number
32052. Low memory options
32053. Constants should be signed so that signed arithmetic involving them * won't cause values to be coerced accidentally to unsigned.
32054. Note: either pointer may be NULL (at entry), so don't assert.
32055. **comment:** Portability workaround is required for platforms without * unaligned access. The replacement code emulates little * endian access even on big endian architectures, which is * OK as long as it is consistent for a build.
label: code-design
32056. tail call -> reuse current "frame"
32057. refcount is unsigned, so always true
32058. work list for objects to be finalized (by mark-and-sweep)
32059. **comment:** XXX: remove the preventcount and make yield walk the callstack? * Or perhaps just use a single flag, not a counter, faster to just * set and restore?
label: code-design
32060. **comment:** still in use with verbose messages
label: code-design
32061. stack is unbalanced, but: [<something> buf]
32062. parse ranges until character class ends
32063. 'delete'
32064. **comment:** unused now, not needed until Turkish/Azeri
label: requirement
32065. No platform-specific parsing, this is not an error.
32066. mark finalizable as reachability roots
32067. **comment:** XXX: Add a proper condition. If formals list is omitted, recheck * handling for 'length' in `duk_js_push_closure()`; it currently relies * on `_Formals` being set. Removal may need to be conditional to debugging * being enabled/disabled too.
label: code-design
32068. **comment:** XXX: naming is inconsistent: ATOM_END_GROUP ends an ASSERT_START_LOOKAHEAD
label: code-design
32069. `ToUint32()` coercion required
32070. 'global'
32071. not popped by side effect
32072. Add a number containing: pc, activation flags. * * PC points to next instruction, find offending PC. Note that * PC == 0 for native code.
32073. Check array density and indicate whether or not the array part should be abandoned.
32074. * Duktape.Buffer: `toString()`, `valueOf()`
32075. descriptor type specific checks
32076. `DUK_USE_ERRTHROW`
32077. Can happen if `duk_throw()` is called on an empty * callstack.
32078. * Longjmp and other control flow transfer for the bytecode executor. * * The longjmp handler can handle all longjmp types: error, yield, and * resume (pseudotypes are never actually thrown). * * Error policy for longjmp: should not ordinarily throw errors; if errors * occur (e.g. due to out-of-memory) they bubble outwards rather than being * handled recursively.
32079. unary minus
32080. ignore non-strings
32081. Clamp to target's end if too long. * * NOTE: there's no overflow possibility in the comparison; * both target_ustart and copy_size are ≥ 0 and based on * values in `duk_int_t` range. Adding them as `duk_uint_t` * values is then guaranteed not to overflow.
32082. move pivot to its final place
32083. Setting to NULL causes 'caller' to be set to * 'null' as desired.
32084. **comment:** * Global state for working around missing variadic macros
label: code-design
32085. * Allocate and initialize a new entry, resizing the properties allocation * if necessary. Returns entry index (e_idx) or throws an error if alloc fails. * * Sets the key of the entry (increasing the key's refcount), and updates * the hash part if it exists. Caller must set value and flags, and update * the entry value refcount. A decref for the previous value is not necessary.
32086. tolerates NULL h_buf
32087. forced_reg
32088. DUK_TOK_PACKAGE
32089. B -> target register * C -> constant index of identifier name
32090. no match, next case
32091. * Interned identifier is compared against reserved words, which are * currently interned into the heap context. See `genbuiltins.py`. * * Note that an escape in the identifier disables recognition of * keywords; e.g. "\u0069f = 1;" is a valid statement (assigns to * identifier named "if"). This is not necessarily compliant, * see test-dec-escaped-char-in-keyword.js. * * Note: "get" and "set" are awkward. They are not officially * ReservedWords (and indeed e.g. "var set = 1;" is valid), and * must come out as DUK_TOK_IDENTIFIER. The compiler needs to * work around this a bit.
32092. NULL if tv_obj is primitive
32093. lj value2: thread
32094. Note: use 'curr' as a temp propdesc
32095. Setter APIs detect special year numbers (0...99) and apply a +1900 * only in certain cases. The legacy `getYear()` getter applies -1900 * unconditionally.
32096. * `substring()`, `substr()`, `slice()`
32097. 30: setHours
32098. Expression, ']' terminates
32099. * Number checkers
32100. E5 Section 11.2.3, step 6.a.i
32101. -> [hobject props]
32102. XXX: shared handling for 'duk_expr_lhs'?
32103. Fix for <https://github.com/svaarala/duktape/issues/155>: * If 'default' is first clause (detected by `pc_prevcase < 0`) * we need to ensure we stay in the matching chain.
32104. eat 'in'
32105. This may trigger mark-and-sweep with arbitrary side effects, * including an attempted resize of the object we're resizing, * executing a finalizer which may add or remove properties of * the object we're resizing etc.
32106. Call with count == `DUK_LEXER_WINDOW_SIZE` to fill buffer initially.
32107. zero-based month
32108. * Must be very careful here, every DECREF may cause reallocation * of our valstack.
32109. expect_token
32110. periods
32111. idx of first argument on stack
32112. Add trailer if: * a) non-empty input * b) empty input and no (zero size) match found (step 11)
32113. check stack first
32114. mod_id may be NULL

32115. http://en.wikipedia.org/wiki/ANSI_escape_code
32116. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
32117. DUK_TOK_SET
32118. * Lookup variable and update in-place if found.
32119. 'Int8Array'
32120. Last component name in resolved path
32121. Must match src/extract_chars.py, generate_match_table3().
32122. Resize target buffer for requested size. Called by the macro only when the * fast path test (= there is space) fails.
32123. Ready to make the copy. We must proceed element by element * and must avoid any side effects that might cause the buffer * validity check above to become invalid. * * Although we work through the value stack here, only plain * numbers are handled which should be side effect safe.
32124. **comment:** * Compared to a direct macro expansion this wrapper saves a few * instructions because no heap dereferencing is required.
label: code-design
32125. Force pause if we were doing "step into" in another activation.
32126. allow fmt==NULL
32127. **comment:** * Relocate property allocation, moving properties to the new allocation. * * Includes key compaction, rehashing, and can also optionally abandoning * the array part, 'migrating' array entries into the beginning of the * new entry part. Arguments are not validated here, so e.g. new_h_size * MUST be a valid prime. * * There is no support for in-place reallocation or just compacting keys * without resizing the property allocation. This is intentional to keep * code size minimal. * * The implementation is relatively straightforward, except for the array * abandonment process. Array abandonment requires that new string keys * are interned, which may trigger GC. All keys interned so far must be * reachable for GC at all times; valstack is used for that now. * * Also, a GC triggered during this reallocation process must not interfere * with the object being resized. This is currently controlled by using * heap->mark_and_sweep_base_flags to indicate that no finalizers will be * executed (as they can affect ANY object) and no objects are compacted * (it would suffice to protect this particular object only, though). * * Note: a non-checked variant would be nice but is a bit tricky to * implement for the array abandonment process. It's easy for * everything else. * * Note: because we need to potentially resize the valstack (as part * of abandoning the array part), any tval pointers to the valstack * will become invalid after this call.
label: code-design
32128. Note: target registers a and a+1 may overlap with DUK__REGP(b) * and DUK__REGCONSTP(c). Careful here.
32129. duk_hcompiledfunction flags; quite version specific
32130. these are not necessarily 0 or 1 (may be other non-zero), that's ok
32131. +1 = finally
32132. Clamping needed if duk_int_t is 64 bits.
32133. 'id'
32134. output bufwriter
32135. -> x in [-2**31,2**31[
32136. DUK_USE_SELF_TESTS
32137. built-in strings
32138. * Process messages and send status if necessary. * * If we're paused, we'll block for new messages. If we're not * paused, we'll process anything we can peek but won't block * for more. Detach (and re-attach) handling is all localized * to duk_debug_process_messages() too. * * Debugger writes outside the message loop may cause debugger * detach1 phase to run, after which dbg_read_cb == NULL and * dbg_detaching != 0. The message loop will finish the detach * by running detach2 phase, so enter the message loop also when * detaching.
32139. * The error object has been augmented with a traceback and other * info from its creation point -- usually the current thread. * The error handler, however, is called right before throwing * and runs in the yielder's thread.
32140. * Error throwing helpers
32141. [... new_glob new_env new_glob new_glob]
32142. For lightfuncs, simply read the virtual property.
32143. 0x70-0x7f
32144. stable
32145. XXX: Could add fast path (for each transform callback) with direct byte * lookups (no shifting) and no explicit check for x < 0x80 before table * lookup.
32146. Duktape.Thread.resume()
32147. **comment:** * Note: of native Ecmascript objects, only Function instances * have a [[HasInstance]] internal property. Custom objects might * also have it, but not in current implementation. * * XXX: add a separate flag, DUK_HOBJECT_FLAG_ALLOW_INSTANCEOF?
label: code-design
32148. nothing to NULL
32149. use SameValue instead of non-strict equality
32150. a value is left on stack regardless of rc
32151. Detach is pending; can be triggered from outside the * debugger loop (e.g. Status notify write error) or by * previous message handling. Call detached callback * here, in a controlled state, to ensure a possible * reattach inside the detached_cb is handled correctly. * * Recheck for detach in a while loop: an immediate * reattach involves a call to duk_debugger_attach() * which writes a debugger handshake line immediately * inside the API call. If the transport write fails * for that handshake, we can immediately end up in a * "transport broken, detaching" case several times here. * Loop back until we're either cleanly attached or * fully detached. * * NOTE: Reset dbg_processing = 1 forcibly, in case we * re-attached; duk_debugger_attach() sets dbg_processing * to 0 at the moment.
32152. Run the finalizer, duk_hobject_run_finalizer() sets FINALIZED. * Next mark-and-sweep will collect the object unless it has * become reachable (i.e. rescued). FINALIZED prevents the * finalizer from being executed again before that.
32153. Note: this must match ToObject() behavior
32154. 'new' MemberExpression
32155. The code below is incorrect if .toString() or .valueOf() have * have been overridden. The correct approach would be to look up * the method(s) and if they resolve to the built-in function we * can safely bypass it and look up the internal value directly. * Unimplemented for now, abort fast path for boxed values.
32156. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx [21 bits]
32157. **comment:** XXX: avoid this at least when enum_target is an Array, it has an * array part, and no ancestor properties were included? Not worth * it for JSON, but maybe worth it for forEach().
label: code-design
32158. Report it to the debug client
32159. no compact
32160. restricted to regs
32161. dummy so sp won't get updated
32162. * Replace global object.
32163. require a (mutable) temporary as a result (or a const if allowed)
32164. contains value
32165. %s' and NULL is not portable, so special case * it for debug printing.
32166. Compiler SyntaxErrors (and other errors) come first, and are * blamed by default (not flagged "noblame").
32167. Relevant PC is just before current one because PC is * post-incremented. This should match what error augment * code does.
32168. Parse formals.
32169. [... key arg1 ... argN func]
32170. exit bytecode execution with return value
32171. * Status message and helpers
32172. Note: 'func' is popped from valstack here, but it is * already reachable from the activation.
32173. points to LABEL; pc+1 = jump site for break; pc+2 = jump site for continue
32174. DUK_SELFTEST_H_INCLUDED
32175. Duktape object
32176. 0xb0-0xbf

32177. * 'key_obj' tracks keys encountered so far by associating an * integer with flags with already encountered keys. The checks * below implement E5 Section 11.1.5, step 4 for production: * * PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment

32178. [... res]

32179. * UTF-8 decoder which accepts longer than standard byte sequences. * This allows full 32-bit code points to be used.

32180. accessor with defined getter

32181. **comment:** XXX: optimize for packed duk_tval directly?
label: code-design

32182. * Augment created errors upon creation (not when they are thrown or * rethrown). __FILE__ and __LINE__ are not desirable here; the call * stack reflects the caller which is correct.

32183. allow source elements

32184. one token before

32185. 'filename'

32186. Neutered. We could go into the switch-case safely with * buf == NULL because check_length == 0. To avoid scanbuild * warnings, fail directly instead.

32187. Allowed.

32188. -> x in J-2**32, 2**32[

32189. no need to incref

32190. Ecmascript function

32191. embed: void ptr

32192. Lightfuncs are always native functions and have "newenv".

32193. This upper value has been experimentally determined; debug build will check * bigint size with assertions.

32194. **comment:** XXX: cover this with the generic >1 case?
label: code-design

32195. * Variant 1 or 3

32196. 'enumerable'

32197. **comment:** * There are two [[Construct]] operations in the specification: * * - E5 Section 13.2.2: for Function objects * - E5 Section 15.3.4.5.2: for "bound" Function objects * * The chain of bound functions is resolved in Section 15.3.4.5.2, * with arguments "piling up" until the [[Construct]] internal * method is called on the final, actual Function object. Note * that the "prototype" property is looked up *only* from the * final object, *before* calling the constructor. * * Currently we follow the bound function chain here to get the * "prototype" property value from the final, non-bound function. * However, we let duk_handle_call() handle the argument "piling" * when the constructor is called. The bound function chain is * thus now processed twice. * * When constructing new Array instances, an unnecessary object is * created and discarded now: the standard [[Construct]] creates an * object, and calls the Array constructor. The Array constructor * returns an Array instance, which is used as the result value for * the "new" operation; the object created before the Array constructor * call is discarded. * * This would be easy to fix, e.g. by knowing that the Array constructor * will always create a replacement object and skip creating the fallback * object in that case. * * Note: functions called via "new" need to know they are called as a * constructor. For instance, built-in constructors behave differently * depending on how they are called.
label: code-design

32198. zero size not an issue: pointers are valid

32199. fromPresent = true

32200. The counter value is one less than the init value: init value should * indicate how many instructions are executed before interrupt. To * execute 1 instruction (after interrupt handler return), counter must * be 0.

32201. assumes that allocated pointers and alloc funcs are valid * if res exists

32202. DUK_TOK_EQ

32203. * >>> struct.pack('>d', 102030405060).encode('hex') * '4237c17c6dc40000'

32204. XXX: add 'language' argument when locale/language sensitive rule * support added.

32205. Currently there are no deletable virtual properties, but * with force_flag we might attempt to delete one.

32206. **comment:** XXX: This helper is a bit awkward because the handling for the different iteration * callers is quite different. This now compiles to a bit less than 500 bytes, so with * 5 callers the net result is about 100 bytes / caller.
label: code-design

32207. Infinity

32208. **comment:** XXX: unoptimal use of temps, resetting
label: code-design

32209. -> [... regexp string] -> [... res_obj]

32210. token type (with reserved words as DUK_TOK_IDENTIFIER)

32211. Note: assumes that these string indexes are 8-bit, genstrings.py must ensure that

32212. start variant 3/4 left-hand-side code (L1 in doc/compiler.rst example)

32213. # argument registers target function wants (< 0 => "as is")

32214. 'magic' constants for Murmurhash2

32215. If something is thrown with the debugger attached and nobody will * catch it, execution is paused before the longjmp, turning over * control to the debug client. This allows local state to be examined * before the stack is unwound. Errors are not intercepted when debug * message loop is active (e.g. for Eval).

32216. Object built-in methods

32217. Skip fully.

32218. switch (ch2)

32219. type

32220. * Allocate memory with garbage collection

32221. * Main switch for statement / source element type.

32222. Shared entry handling for object/array serialization.

32223. overwrite sep

32224. [message error message]

32225. [... parent stash stash] -> [... parent stash]

32226. Non-verbose errors for low memory targets: no file, line, or message.

32227. DUK_BUFOBJ_ARRAYBUFFER

32228. **comment:** * Recursion limit check. * * Note: there is no need for an "ignore recursion limit" flag * for duk_handle_safe_call now.
label: code-design

32229. Exponent handling: if exponent format is used, record exponent value and * fake k such that one leading digit is generated (e.g. digits=123 -> "1.23"). * * toFixed() prevents exponent use; otherwise apply a set of criteria to * match the other API calls (toString(), toPrecision, etc).

32230. **comment:** We want the argument expression value to go to "next temp" * without additional moves. That should almost always be the * case, but we double check after expression parsing. * * This is not the cleanest possible approach.
label: code-design

32231. seconds, doesn't fit into 16 bits

32232. unreferenced w/o tracebacks

32233. We want to avoid making a copy to process set() but that's * not always possible: the source and the target may overlap * and because element sizes are different, the overlap cannot * always be handled with a memmove() or choosing the copy * direction in a certain way. For example, if source type is * uint8 and target type is uint32, the target area may exceed * the source area from both ends! * * Note that because external buffers may point to the same * memory areas, we must ultimately make this check using * pointers. * * NOTE: careful with side effects: any side effect may cause * a buffer resize (or external buffer pointer/length update)!

32234. **comment:** NEXTENUM needs a jump slot right after the main instruction. * When the JUMP is taken, output spilling is not needed so this * workaround is possible. The jump slot PC is exceptionally * plumbed through comp_ctx to minimize call sites.
label: code-design

32235. must be updated to work properly (e.g. creation of 'arguments')

32236. Unescaped '/' ANYWHERE in the regexp (in disjunction, * inside a character class, ...) => same escape works.
32237. 1:1 or special conversions, but not locale/context specific: script generated rules
32238. roughly 0.5 kB
32239. * Value stack resize and stack top adjustment helper. ** XXX: This should all be merged to duk_valstack_resize_raw().
32240. **comment:** * Macro support functions for reading/writing raw data. ** These are done using mempcy to ensure they're valid even for unaligned * reads/writes on platforms where alignment counts. On x86 at least gcc * is able to compile these into a bswap+mov. "Always inline" is used to * ensure these macros compile to minimal code. ** Not really bufwriter related, but currently used together.
label: code-design
32241. If tracebacks are enabled, the '_Tracedata' property is the only * thing we need: 'fileName' and 'lineNumber' are virtual properties * which use '_Tracedata'.
32242. **comment:** XXX: could duk_is_undefined() provide defaulting undefined to 'len' * (the upper limit)?
label: code-design
32243. DUK_TOK_WITH
32244. 0xa0-0xaf
32245. * Table for base-64 decoding
32246. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor).
label: code-design
32247. [... buf]
32248. * Regexp compilation. ** See doc/regexp.rst for a discussion of the compilation approach and * current limitations. ** Regexp bytecode assumes jumps can be expressed with signed 32-bit * integers. Consequently the bytecode size must not exceed 0x7fffffffL. * The implementation casts duk_size_t (buffer size) to duk_(u)int32_t * in many places. Although this could be changed, the bytecode format * limit would still prevent regexps exceeding the signed 32-bit limit * from working. ** XXX: The implementation does not prevent bytecode from exceeding the * maximum supported size. This could be done by limiting the maximum * input string size (assuming an upper bound can be computed for number * of bytecode bytes emitted per input byte) or checking buffer maximum * size when emitting bytecode (slower).
32249. Production allows 'DecimalDigits', including leading zeroes
32250. Note: must coerce to a (writable) temp register, so that e.g. "!x" where x * is a reg-mapped variable works correctly (does not mutate the variable register).
32251. [] -> []
32252. key
32253. * Exception for Duktape internal throws when C++ exceptions are used * for long control transfers. ** Doesn't inherit from any exception base class to minimize the chance * that user code would accidentally catch this exception.
32254. -> [array totalLength buf]
32255. Maximum prototype traversal depth. Sanity limit which handles e.g. * prototype loops (even complex ones like 1->2->3->4->2->3->4->2->3->4).
32256. is valid size
32257. [O Properties obj]
32258. extra coercion of strings is OK
32259. **comment:** XXX: increase ctx->expr_tokens here for every consumed token * (this would be a nice statistic)?
label: code-design
32260. FAIL
32261. Two digit octal escape, digits validated. ** The if-condition is a bit tricky. We could catch e.g. * '\039' in the three-digit escape and fail it there (by * validating the digits), but we want to avoid extra * additional validation code.
32262. For n == 0, Node.js ignores totalLength argument and * returns a zero length buffer.
32263. * Push result object and init its flags
32264. number of continuation (non-initial) bytes in [0x80,0xbf]
32265. External buffer with 'curr_alloc' managed by user code and pointing to an * arbitrary address. When heap pointer compression is not used, this struct * has the same layout as duk_hbuffer_dynamic.
32266. **comment:** Note: we can reuse 'desc' here
label: code-design
32267. empty statement with an explicit semicolon
32268. * Constructor
32269. * Array exotic behaviors can be implemented at this point. The local variables * are essentially a 'value copy' of the input descriptor (Desc), which is modified * by the Array [[DefineOwnProperty]] (E5 Section 15.4.5.1).
32270. * Expression parsing. ** Upon entry to 'expr' and its variants, 'curr_tok' is assumed to be the * first token of the expression. Upon exit, 'curr_tok' will be the first * token not part of the expression (e.g. semicolon terminating an expression * statement).
32271. The index range space is conceptually the array part followed by the * entry part. Unlike normal enumeration all slots are exposed here as * is and return 'unused' if the slots are not in active use. In particular * the array part is included for the full a_size regardless of what the * array .length is.
32272. * toString()
32273. shuffle registers if large number of regs/consts
32274. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8
32275. **comment:** XXX: add support for variables to not be register bound always, to * handle cases with a very large number of variables?
label: code-design
32276. entry_top + 2
32277. [result]
32278. * Determine call type, then finalize activation, shift to * new value stack bottom, and call the target.
32279. [source]
32280. * Arguments object creation. ** Creating arguments objects involves many small details, see E5 Section * 10.6 for the specific requirements. Much of the arguments object exotic * behavior is implemented in duk_hobject_props.c, and is enabled by the * object flag DUK_HOBJECT_FLAG_EXOTIC_ARGUMENTS.
32281. **comment:** XXX: space in valstack? see discussion in duk_handle_call_xxx().
label: code-design
32282. combines steps 2 and 5; -len ensures max() not needed for step 5
32283. for register-bound and declarative env identifiers
32284. bound functions are never in act 'func'
32285. Lightfuncs cannot be bound.
32286. **comment:** The resulting buffer object gets the same class and prototype as * the buffer in 'this', e.g. if the input is a Node.js Buffer the * result is a Node.js Buffer; if the input is a Float32Array, the * result is a Float32Array. ** For the class number this seems correct. The internal prototype * is not so clear: if 'this' is a bufferobject with a non-standard * prototype object, that value gets copied over into the result * (instead of using the standard prototype for that object type).
label: code-design
32287. Don't allow mixed padding and actual chars.
32288. reject RegExp literal on next advance() call; needed for handling IdentifierName productions
32289. DUK_TOK_CONTINUE
32290. unbalanced stack on purpose
32291. arguments MUST have an initialized lexical environment reference
32292. **comment:** XXX: sanitize to printable (and maybe ASCII)
label: code-design
32293. tags
32294. regexp compiler should catch these
32295. **comment:** Current 'this' binding is the value just below idx_bottom. * Previously, 'this' binding was handled with an index to the * (calling) valstack. This works for everything except tail * calls, which must not "cumulate" valstack temps.

label: code-design

32296. * ctx->prev_token token to process with duk__expr_nud() * ctx->curr_token updated by caller * * Note: the token in the switch below has already been eaten.
32297. -> [... voidp true]
32298. The E5.1 specification does not seem to allow IdentifierPart characters * to be used as identity escapes. Unfortunately this includes '\$', which * cannot be escaped as '\$'; it needs to be escaped e.g. as '\u0024'. * Many other implementations (including V8 and Rhino, for instance) do * accept '\$' as a valid identity escape, which is quite pragmatic. * See: test-regexp-identity-escape-dollar.js.
32299. mp <- b^(e+1)
32300. * Debugger handling for executor restart * * Check for breakpoints, stepping, etc, and figure out if we should execute * in checked or normal mode. Note that we can't do this when an activation * is created, because breakpoint status (and stepping status) may change * later, so we must recheck every time we're executing an activation. * This primitive should be side effect free to avoid changes during check.
32301. at most sizeof(buf) - 1
32302. val
32303. Old solution: don't iterate, incorrect
32304. * API calls related to general value stack manipulation: resizing the value * stack, pushing and popping values, type checking and reading values, * coercing values, etc. * * Also contains internal functions (such as duk_get_tval()), defined * in duk_api_internal.h, with semantics similar to the public API.
32305. No BC shuffling now.
32306. syntactically left-associative but parsed as right-associative
32307. * Local result type for duk_get_identifier_reference() lookup.
32308. **comment:** XXX: would be nice to enumerate an object at specified index
label: code-design
32309. ''
32310. * Heap stringtable handling, string interning.
32311. By default the global object is read-only which is often much * more of an issue than having read-only built-in objects (like * RegExp, Date, etc). Use a RAM-based copy of the global object * and the global environment object for convenience.
32312. Creation time error augmentation
32313. The casts through duk_intr_pt is to avoid the following GCC warning: * * warning: cast from pointer to integer of different size [-Wpointer-to-int-cast] * * This still generates a /Wp64 warning on VS2010 when compiling for x86.
32314. **comment:** * Note: currently register bindings must be fixed for the entire * function. So, even though the catch variable is in a register * we know, we must use an explicit environment record and slow path * accesses to read/write the catch binding to make closures created * within the catch clause work correctly. This restriction should * be fixable (at least in common cases) later. * * See: test-bug-catch-binding-2.js. * * XXX: improve to get fast path access to most catch clauses.
label: code-design
32315. check that byte has the form 10xx xxxx
32316. * Memory constants
32317. return 'res' (of right part) as our result
32318. -> [func funcname env]
32319. this variant is used when an assert would generate a compile warning by * being always true (e.g. >= 0 comparison for an unsigned value
32320. Stringcache is used for speeding up char-offset-to-byte-offset * translations for non-ASCII strings.
32321. **comment:** Note: strictness is *not* inherited from the current Duktape/C. * This would be confusing because the current strictness state * depends on whether we're running inside a Duktape/C activation * (= strict mode) or outside of any activation (= non-strict mode). * See tests/api/test-eval-strictness.c for more discussion.
label: code-design
32322. * Finalizer check. * * Note: running a finalizer may have arbitrary side effects, e.g. * queue more objects on refzero_list (tail), or even trigger a * mark-and-sweep. * * Note: quick reject check should match vast majority of * objects and must be safe (not throw any errors, ever).
32323. Get Proxy target object. If the argument is not a Proxy, return it as is. * If a Proxy is revoked, an error is thrown.
32324. * Assertion helpers.
32325. is_setget
32326. * Compute new alloc size and alloc new area. * * The new area is allocated as a dynamic buffer and placed into the * valstack for reachability. The actual buffer is then detached at * the end. * * Note: heap_mark_and_sweep_base_flags are altered here to ensure * no-one touches this object while we're resizing and rehashing it. * The flags must be reset on every exit path after it. Finalizers * and compaction is prevented currently for all objects while it * would be enough to restrict it only for the current object.
32327. replacer function
32328. [...]
32329. **comment:** remove the string (mark DELETED), could also call * duk_heap_string_remove() but that would be slow and * pointless because we already know the slot.
label: code-design
32330. +(function(){})-> NaN
32331. **comment:** * Date built-ins * * Unlike most built-ins, Date has some platform dependencies for getting * UTC time, converting between UTC and local time, and parsing and * formatting time values. These are all abstracted behind DUK_USE_xxx * config options. There are built-in platform specific providers for * POSIX and Windows, but external providers can also be used. * * See doc/datetime.rst. *
label: code-design
32332. Reject attempt to change a read-only object.
32333. final result is already in 'res'
32334. 21: getUTCSeconds
32335. Eval is just a wrapper now.
32336. With ROM objects "needs refcount update" is true when the value is * heap allocated and is not a ROM object.
32337. 22: getMilliseconds
32338. binary arithmetic using extraops; DUK_EXTRAOP_INSTOF etc
32339. combined algorithm matching E5 Sections 9.5 and 9.6
32340. Evaluates to (duk_uint8_t *) pointing to start of area.
32341. module.id = resolved_id
32342. Property access expressions ('a[b]') are critical to correct * LHS evaluation ordering, see test-dev-assign-eval-order*.js. * We must make sure that the LHS target slot (base object and * key) don't change during RHS evaluation. The only concrete * problem is a register reference to a variable-bound register * (i.e., non-temp). Require temp regs for both key and base. * * Don't allow a constant for the object (even for a number * etc), as it goes into the 'A' field of the opcode.
32343. Maximum number of characters in formatted value.
32344. Read fully.
32345. * Prototypes
32346. For indirect allocs.
32347. always allow 'in', coerce to 'tr' just in case
32348. DUK_USE_DEBUGGER_SUPPORT && (DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT)
32349. Pointer object internal value is immutable
32350. Leave 'value' on stack
32351. prev char
32352. * [[DefaultValue]] (E5 Section 8.12.8) * * ==> implemented in the API.
32353. 'extends'
32354. result: hash_prime(floor(1.2 * e_size))
32355. bp is assumed to be even
32356. stridx_logfunc[] must be static to allow initializer with old compilers like BCC

32357. Positive index can be higher than valstack top but must * not go above allocated stack (equality is OK).
32358. out_desc is left untouched (possibly garbage), caller must use return * value to determine whether out_desc can be looked up
32359. If buffer has been exhausted, truncate bitstream
32360. Can be called multiple times with no harm.
32361. Helper for component setter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), modify one or more components as specified, recompute * the time value, set it as the internal value. Finally, push the * new time value as a return value to the value stack and return 1 * (caller can then tail call us).
32362. error
32363. 1:1 conversion
32364. Encode into a double constant (53 bits can encode $6^8 = 48$ bits + 3-bit length)
32365. not LS/PS
32366. invalid codepoint encoding or codepoint
32367. **comment:** packed advance/token number macro used by multiple functions
 label: code-design
32368. The JO(value) operation: encode object. * * Stack policy: [object] -> [object].
32369. DUK_TOK_BAND_EQ
32370. remaining
32371. **comment:** * Duktape built-ins * * Size optimization note: it might seem that vararg multipurpose functions * like fin(), enc(), and dec() are not very size optimal, but using a single * user-visible Ecmascript function saves a lot of run-time footprint; each * Function instance takes >100 bytes. Using a shared native helper and a * 'magic' value won't save much if there are multiple Function instances * anyway.
 label: code-design
32372. Value is required to be a number in the fast path so there * are no side effects in write coercion.
32373. Faster but value stack overruns are memory unsafe.
32374. register for writing value of 'non-empty' statements (global or eval code), -1 is marker
32375. * Possible pending array length update, which must only be done * if the actual entry write succeeded.
32376. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
32377. If XUTF-8 decoding fails, treat the offending byte as a codepoint directly * and go forward one byte. This is of course very lossy, but allows some kind * of output to be produced even for internal strings which don't conform to * XUTF-8. All standard Ecmascript strings are always CESU-8, so this behavior * does not violate the Ecmascript specification. The behavior is applied to * all modes, including Ecmascript standard JSON. Because the current XUTF-8 * decoding is not very strict, this behavior only really affects initial bytes * and truncated codepoints. * * Another alternative would be to scan forwards to start of next codepoint * (or end of input) and emit just one replacement codepoint.
32378. For initial entry use default value; zero forces an * interrupt before executing the first instruction.
32379. already in correct reg
32380. the next 'if' is guaranteed to match after swap
32381. Encoding state. Heap object references are all borrowed.
32382. -> [... repl_value]
32383. no prototype
32384. Note: 'q' is top-1
32385. **comment:** * Abandon array part because all properties must become non-configurable. * Note that this is now done regardless of whether this is always the case * (skips check, but performance problem if caller would do this many times * for the same object; not likely).
 label: code-design
32386. **comment:** XXX: incref by count (2) directly
 label: code-design
32387. ')'
32388. source is a function expression (used for Function constructor)
32389. * Variant 2
32390. Stored in duk_heapdr h_extra16.
32391. Note: intentionally signed.
32392. 'Date'
32393. input linenumber at input_offset (not window[0]), init to 1
32394. **comment:** * Missing bytes snip base64 example * 0 4 XXXX * 1 3 XXX= * 2 2 XX==
 label: code-design
32395. * The remaining matches are emitted as sequence of SPLITs and atom * copies; the SPLITs skip the remaining copies and match the sequel. * This sequence needs to be emitted starting from the last copy * because the SPLITs are variable length due to the variable length * skip offset. This causes a lot of memory copying now. * * Example structure (greedy, match maximum # atoms): * * SPLIT1 LSEQ * (atom) * SPLIT1 LSEQ ; <- the byte length of this instruction is needed * (atom) ; to encode the above SPLIT1 correctly * ... * LSEQ:
32396. idx_value
32397. [... this_new | arg1 ... argN]
32398. -----XX XXXX----
32399. * Get old and new length
32400. **comment:** Hot variables for interpretation. Critical for performance, * but must add sparingly to minimize register shuffling.
 label: code-design
32401. 38: getYear
32402. initjs data is NUL terminated
32403. this.raw(buffer)
32404. return h_bufres
32405. Here we rely on duk_hstring instances always being zero * terminated even if the actual string is not.
32406. Must coerce key: if key is an object, it may coerce to e.g. 'length'.
32407. -> [... regexp_instance]
32408. Final result is at stack top.
32409. Strict equality is NOT enough, because we cannot use the same * constant for e.g. +0 and -0.
32410. right exists, [[Put]] regardless whether or not left exists
32411. * Return target object.
32412. reuse 'res' as 'left'
32413. st_used remains the same, DELETED is counted as used
32414. is_join
32415. binding power must be high enough to NOT allow comma expressions directly
32416. Note: these variables must reside in 'curr_func' instead of the global * context: when parsing function expressions, expression parsing is nested.
32417. Abandon array part, moving array entries into entries part. * This requires a props resize, which is a heavy operation. * We also compact the entries part while we're at it, although * this is not strictly required.
32418. _Formals
32419. base of known return values
32420. TimeClip(), which also handles Infinity -> NaN conversion
32421. [... obj]
32422. Encoding endianness must match target memory layout, * build scripts and genbuiltins.py must ensure this.
32423. flags used for property attributes in duk_propdesc and packed flags
32424. directly uses slow accesses

32425. * Finalizer torture. Do one fake finalizer call which causes side effects * similar to one or more finalizers on actual objects.
32426. add AC*2^32
32427. result: updated refcount
32428. Entry level info.
32429. catch depth from current func
32430. indexOf: clamp fromIndex to [-len, len] * (if fromIndex == len, for-loop terminates directly) * * lastIndexOf: clamp fromIndex to [-len - 1, len - 1] * (if clamped to -len-1 -> fromIndex becomes -1, terminates for-loop directly)
32431. decoding
32432. inline initializer for coercers[] is not allowed by old compilers like BCC
32433. * Create normalized 'source' property (E5 Section 15.10.3).
32434. variable access
32435. Double casting for pointer to avoid gcc warning (cast from pointer to integer of different size)
32436. **comment:** XXX: refactor out?
label: code-design
32437. same for both error and each subclass like TypeError
32438. [... res_obj re_obj input bc saved_buf]
32439. B -> register for writing enumerator object * C -> value to be enumerated (register)
32440. entire string is whitespace
32441. 'multiline'
32442. '+'
32443. * Parse an identifier and then check whether it is: * - reserved word (keyword or other reserved word) * - "null" (NullLiteral) * - "true" (BooleanLiteral) * - "false" (BooleanLiteral) * - anything else => identifier * * This does not follow the E5 productions cleanly, but is * useful and compact. * * Note that identifiers may contain Unicode escapes, * see E5 Sections 6 and 7.6. They must be decoded first, * and the result checked against allowed characters. * The above if-clause accepts an identifier start and an * '\ character -- no other token can begin with a '\'. * * Note that "get" and "set" are not reserved words in E5 * specification so they are recognized as plain identifiers * (the tokens DUK_TOK_GET and DUK_TOK_SET are actually not * used now). The compiler needs to work around this. * * Strictly speaking, following EcmaScript longest match * specification, an invalid escape for the first character * should cause a syntax error. However, an invalid escape * for IdentifierParts should just terminate the identifier * early (longest match), and let the next tokenization * fail. For instance Rhino croaks with 'foo\z' when * parsing the identifier. This has little practical impact.
32444. 'lineNumber'
32445. refer to callstack entries below current
32446. statement parsing
32447. **comment:** Pick a destination register. If either base value or key * happens to be a temp value, reuse it as the destination. * * XXX: The temp must be a "mutable" one, i.e. such that no * other expression is using it anymore. Here this should be * the case because the value of a property access expression * is neither the base nor the key, but the lookup result.
label: code-design
32448. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()!
32449. ignore retval -> [key]
32450. %ld
32451. * Octal escape or zero escape: * \0 (lookahead not DecimalDigit) * \1 ... \7 (lookahead not DecimalDigit) * \ZeroToThree OctalDigit (lookahead not DecimalDigit)
* \FourToSeven OctalDigit (no lookahead restrictions) * \ZeroToThree OctalDigit (no lookahead restrictions) * * Zero escape is part of the standard syntax. Octal escapes are * defined in E5 Section B.1.2, and are only allowed in non-strict mode. * Any other productions starting with a decimal digit are invalid.
32452. **comment:** XXX: for Object.keys() the [[OwnPropertyKeys]] result (trap result) * should be filtered so that only enumerable keys remain. Enumerability * should be checked with [[GetOwnProperty]] on the original object * (i.e., the proxy in this case). If the proxy has a getOwnPropertyDescriptor * trap, it should be triggered for every property. If the proxy doesn't have * the trap, enumerability should be checked against the target object instead. * We don't do any of this now, so Object.keys() and Object.getOwnPropertyNames() * return the same result now for proxy traps. We still do clean up the trap * result, so that Object.keys() and Object.getOwnPropertyNames() will return a * clean array of strings without gaps.
label: code-design
32453. syntax error
32454. don't push value
32455. 'buffer'
32456. inherit initial strictness from parent
32457. module
32458. Without tracebacks the concrete .fileName and .lineNumber need * to be added directly.
32459. not emergency
32460. * Raw memory calls: relative to heap, but no GC interaction
32461. Lookup 'key' from arguments internal 'map', perform a variable lookup * if mapped, and leave the result on top of stack (and return non-zero). * Used in E5 Section 10.6 algorithms [[Get]] and [[GetOwnProperty]].
32462. Evaluate RHS only when LHS is safe.
32463. debugger detaching; used to avoid calling detach handler recursively
32464. no catchers
32465. * Variant 4
32466. 'package'
32467. nothing now
32468. * Reset trigger counter
32469. * Preliminaries
32470. count is already incremented, take into account
32471. **comment:** * Setters. * * Setters are a bit more complicated than getters. Component setters * break down the current time value into its (normalized) component * parts, replace one or more components with -unnormalized- new values, * and the components are then converted back into a time value. As an * example of using unnormalized values: * * var d = new Date(1234567890); * * is equivalent to: * * var d = new Date(0); * d.setUTCMilliseconds(1234567890); * * A shared native helper to provide almost all setters. Magic value * contains a set of flags and also packs the "maxnargs" argument. The * helper provides: * * setMilliseconds() * setUTCMilliseconds() * setSeconds() * setUTCSeconds() * setMinutes() * setUTCMinutes() * setHours() * setUTCHours() * setDate() * setUTCDate() * setMonth() * setUTCMonth() * setFullYear() * setUTCFullYear() * setYear() * * Notes: * * - Date.prototype.setYear() (Section B addition): special year check * is omitted. NaN / Infinity will just flow through and ultimately * result in a NaN internal time value. * * - Date.prototype.setYear() does not have optional arguments for * setting month and day-in-month (like setFullYear()), but we indicate * 'maxnargs' to be 3 to get the year written to the correct component * index in duk__set_part_helper(). The function has nargs == 1, so only * the year will be set regardless of actual argument count.
label: code-design
32472. inline string concatenation
32473. DUK_USE_JX && DUK_USE_JC
32474. **comment:** XXX: ideally this would be just one flag (maybe a derived one) so * that a single bit test is sufficient to check the condition.
label: test
32475. [key] (coerced)
32476. **comment:** XXX: no optimized variants yet
label: requirement
32477. Encode a double (checked by caller) from stack top. Stack top may be * replaced by serialized string but is not popped (caller does that).
32478. This function is only called for objects with array exotic behavior. * The [[DefineOwnProperty]] algorithm for arrays requires that * 'length' can never have a value outside the unsigned 32-bit range, * attempt to write such a value is a RangeError. Here we can thus * assert for this. When Duktape internals go around the official * property write interface (doesn't happen often) this assumption is * easy to accidentally break, so such code must be written carefully. * See test-bi-array-push-maxlen.js.

32479. **comment:** DUK__ADVANCECHARS(lex_ctx, 2) would be correct here, but it unnecessary
label: code-design

32480. 'base64'
32481. [sep ToObject(this) len sep]
32482. **comment:** * Various sanity checks for typing
label: code-design

32483. Check whether statement list ends.
32484. invalidates tv1, tv2
32485. Note: MUST NOT wipe_and_return here, as heap->lj must remain intact
32486. -> sets 'f' and 'e'
32487. Check for maximum string length
32488. * Parse an individual source element (top level statement) or a statement. ** Handles labeled statements automatically (peeling away labels before * parsing an expression that follows the label(s)). ** Upon entry, 'curr_tok' contains the first token of the statement (parsed * in "allow regexp literal" mode). Upon exit, 'curr_tok' contains the first * token following the statement (if the statement has a terminator, this is * the token after the terminator).
32489. **comment:** loop_stack_index could be perhaps be replaced by 'depth', but it's nice * to not couple these two mechanisms unnecessarily.
label: code-design

32490. * Fast path for defining array indexed values without interning the key. * This is used by e.g. code for Array prototype and traceback creation so * must avoid interning.
32491. accessor
32492. set to 1 if any match exists (needed for empty input special case)
32493. this should never be possible, because the switch-case is * comprehensive
32494. avoid double coercion
32495. 'DataView'
32496. * Sweep stringtable
32497. [A B C D E F G H] rel_index = 2, del_count 3, item count 1 * -> [A B F G H] (conceptual intermediate step) * -> [A B . F G H] (placeholder marked) * [A B C F G H] (actual result at this point, C will be replaced)
32498. DUK_TOK_IN
32499. jump to false
32500. **comment:** XXX: resolve macro definition issue or call through a helper function?
label: code-design

32501. **comment:** Sanity workaround for handling functions with a large number of * constants at least somewhat reasonably. Otherwise checking whether * we already have the constant would grow very slow (as it is O(N^2)).
label: code-design

32502. heapobj recursion depth when deep printing is selected
32503. basic types
32504. temp reset is not necessary after duk_parse_stmt(), which already does it
32505. mark-and-sweep: reachable
32506. Error object is augmented at its creation here.
32507. DUK_USE_COMMONJS_MODULES
32508. at least effective 'this'
32509. * Default allocation functions. ** Assumes behavior such as malloc allowing zero size, yielding * a NULL or a unique pointer which is a no-op for free.
32510. DUK_USE_LIGHTFUNC_BUILTINS
32511. some derived types
32512. functions always have a NEWENV flag, i.e. they get a * new variable declaration environment, so only lex_env * matters here.
32513. x == +Infinity
32514. temp_start+0 = key, temp_start+1 = closure
32515. E5.1 Section 15.1.3.2: empty
32516. DUK_TOK_FINALLY
32517. **comment:** Static call style.
label: code-design

32518. 'deleteProperty'
32519. Compiling state of one function, eventually converted to duk_hcompiledfunction
32520. to body
32521. Coerce top into Object.prototype.toString() output.
32522. XXX: this is quite clunky. Add Unicode helpers to scan backwards and * forwards with a callback to process codepoints?
32523. actual update happens once write has been completed without * error below.
32524. * 'func' is now a non-bound object which supports [[HasInstance]] * (which here just means DUK_HOBJECT_FLAG_CALLABLE). Move on * to execute E5 Section 15.3.5.3.
32525. Target object must be checked for a conflicting * non-configurable property.
32526. property exists, either 'desc' is empty, or all values * match (SameValue)
32527. [func thisArg argArray]
32528. p overshoots
32529. prefer direct
32530. **comment:** XXX: what about statements which leave a half-cooked value in 'res' * but have no stmt value? Any such statements?
label: code-design

32531. Must restart in all cases because we NULLed thr->ptr_curr_pc.
32532. MAXVAL is inclusive
32533. "release" preallocated temps since we won't need them
32534. temp reg value for start of loop
32535. checked by Duktape.Thread.resume()
32536. * Union aliasing, see misc/clang_aliasing.c.
32537. [offset value littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)
32538. * Replacements for missing platform functions. ** Unlike the originals, fpclassify() and signbit() replacements don't * work on any floating point types, only doubles. The C typing here * mimics the standard prototypes.
32539. eat closing quote
32540. setup many variables in nc_ctx
32541. **comment:** XXX: fast int-to-double
label: code-design

32542. Native function pointer may be different from a void pointer, * and we serialize it from memory directly now (no byte swapping etc).
32543. **comment:** XXX: shorter version for 12-byte representation?
label: code-design

32544. a fresh require() with require.id = resolved target module id
32545. Constructor
32546. elision - flush
32547. temp reg for key literal
32548. Unlike break/continue, throw statement does not allow an empty value.
32549. DUK_TOK_VAR

32550. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object).
Return value is NULL if value is neither an object nor a * lightfunc.

32551. char offset in [0, h_input->clen] (both ends inclusive), checked before entry

32552. NOTE: fmod(x) result sign is same as sign of x, which * differs from what Javascript wants (see Section 9.6).

32553. [... bytecode escaped_source]

32554. [obj key desc value get set curr_value]

32555. >>

32556. msec

32557. unary plus of a number is identity

32558. equals and strict equals

32559. * Loose equality, strict equality, and SameValue (E5 Sections 11.9.1, 11.9.4, * 9.12). These have much in common so they can share some helpers. * * Future work notes: * - Current implementation (and spec definition) has recursion; this should * be fixed if possible. * * - String-to-number coercion should be possible without going through the * value stack (and be more compact) if a shared helper is invoked.

32560. The debugger protocol doesn't support a plain integer encoding for * the full 32-bit unsigned range (only 32-bit signed). For now, * unsigned 32-bit values simply written as signed ones. This is not * a concrete issue except for 32-bit heapdfr fields. Proper solutions * would be to (a) write such integers as IEEE doubles or (b) add an * unsigned 32-bit dvalue.

32561. **comment:** * XXX: attempt to get the call result to "next temp" whenever * possible to avoid unnecessary register shuffles. * * XXX: CSPROP (and CSREG) can overwrite the call target register, and save one temp, * if the call target is a temporary register and at the top of the temp reg "stack".
label: code-design

32562. tv points to element just below prev top

32563. * Duktape/C function magic

32564. * Identifier handling

32565. error gets its 'name' from the prototype

32566. Use a new environment but there's no 'arguments' object; * delayed environment initialization. This is the most * common case.

32567. E5 Sections 11.9.1, 11.9.3

32568. Capital sigma occurred at "end of word", lowercase to * U+03C2 = GREEK SMALL LETTER FINAL SIGMA. Otherwise * fall through and let the normal rules lowercase it to * U+03C3 = GREEK SMALL LETTER SIGMA.

32569. current digit

32570. * Extern

32571. * Mark objects on finalize_list. *

32572. **comment:** XXX: repetition of stack pre-checks -> helper or macro or inline
label: code-design

32573. **comment:** * Data follows the struct header. The struct size is padded by the * compiler based on the struct members. This guarantees that the * buffer data will be aligned-by-4 but not necessarily aligned-by-8. * * On platforms where alignment does not matter, the struct padding * could be removed (if there is any). On platforms where alignment * by 8 is required, the struct size must be forced to be a multiple * of 8 by some means. Without it, some user code may break, and also * Duktape itself breaks (e.g. the compiler stores duk_tvals in a * dynamic buffer).
label: code-design

32574. 'valueOf'

32575. XXX: potential issue with signed pointers, p_end < p.

32576. * Helpers to resize properties allocation on specific needs.

32577. **comment:** The line number tracking is a bit inconsistent right now, which * affects debugger accuracy. Mostly call sites emit opcodes when * they have parsed a token (say a terminating semicolon) and called * duk_advance(). In this case the line number of the previous * token is the most accurate one (except in prologue where * prev_token.start_line is 0). This is probably not 100% correct * right now.
label: code-design

32578. shared object part

32579. "Have" flags must not be conflicting so that they would * apply to both a plain property and an accessor at the same * time.

32580. [... re_obj input]

32581. Backpointers.

32582. string access cache (codepoint offset -> byte offset) for fast string * character looping; 'weak' reference which needs special handling in GC.

32583. bottom of new func

32584. re_ctx->captures at the start of the atom parsed in this loop

32585. * Scan from shortest distance: * - start of string * - end of string * - cache entry (if exists)

32586. refzero_list treated as reachability roots

32587. * Memory calls.

32588. +0.5 is handled by floor, this is on purpose

32589. Duplicate (shadowing) labels are not allowed, except for the empty * labels (which are used as default labels for switch and iteration * statements). * * We could also allow shadowing of non-empty pending labels without any * other issues than breaking the required label shadowing requirements * of the E5 specification, see Section 12.12.

32590. internal temporary value, used for char classes

32591. **comment:** format is too large, abort
label: code-design

32592. -> [... key val']

32593. 0x40-0x4f

32594. MPUTOBJ emitted by outer loop

32595. Errors are augmented when they are created, not when they are * thrown or re-thrown. The current error handler, however, runs * just before an error is thrown.

32596. -0 is accepted here as index 0 because ToString(-0) == "0" which is * in canonical form and thus an array index.

32597. be_ctx->offset == length of encoded bitstream

32598. 'in'

32599. Duktape.modLoaded[resolved_id] = module

32600. Caller must ensure 'tv' is indeed a fastint!

32601. stage 1 guarantees

32602. typed for duk_unicode_decode_xutf8()

32603. Macros for creating and checking bitmasks for character encoding. * Bit number is a bit counterintuitive, but minimizes code size.

32604. * Compile input string into an executable function template without * arguments. * * The string is parsed as the "Program" production of EcmaScript E5. * Compilation context can be either global code or eval code (see E5 * Sections 14 and 15.1.2.1). * * Input stack: [... filename] * Output stack: [... func_template]

32605. **comment:** XXX: macro? sets both heapdfr and object flags
label: code-design

32606. [... varname]

32607. preinitialize lexer state partially

32608. **comment:** XXX: slightly awkward
label: code-design

32609. Once recursion depth is increased, exit path must decrease * it (though it's OK to abort the fast path).

32610. require

32611. rnd in [lo,hi]

32612. must have catch and/or finally

32613. V

32614. Technically Array.prototype.push() can create an Array with length * longer than 2^32-1, i.e. outside the 32-bit range. The final length * is *not* wrapped to 32 bits in the specification. * * This implementation tracks length with a uint32 because it's much * more practical. * * See: test-bi-array-push-maxlen.js.

32615. DUK_TOK_BXOR
32616. [... re_obj input bc saved_buf res_obj val]
32617. No arr_idx update or limit check
32618. [... varname]
32619. fresh require (argument)
32620. * Resolution loop. At the top of the loop we're expecting a valid * term: '.', '..', or a non-empty identifier not starting with a period.
32621. String object internal value is immutable
32622. At this point 'res' holds the potential expression value. * It can be basically any ivalue here, including a reg-bound * identifier (if code above deems it safe) or a unary/binary * operation. Operations must be resolved to a side effect free * plain value, and the side effects must happen exactly once.
32623. highest capture number emitted so far (used as: ++captures)
32624. arbitrary remove only works with double linked heap, and is only required by * reference counting so far.
32625. **comment:** never reached, but avoids warnings of * potentially unused variables.
 label: code-design
32626. new entry: previous value is garbage; set to undefined to share write_value
32627. [... name name]
32628. nargs
32629. each call this helper serves has nargs==2
32630. assertion disabled
32631. * Types are different; various cases for non-strict comparison * * Since comparison is symmetric, we use a "swap trick" to reduce * code size.
32632. Ensure dirty state causes a Status even if never process any * messages. This is expected by the bytecode executor when in * the running state.
32633. * A C * B A * C <- sce ==> B * D D
32634. Sign extend: 0x0000##00 -> 0x##000000 -> sign extend to 0xssssss#
32635. * Virtual properties. * * String and buffer indices are virtual and always enumerable, * 'length' is virtual and non-enumerable. Array and arguments * object props have special behavior but are concrete.
32636. * Case conversion
32637. * Helpers for dealing with the input string
32638. const limits
32639. 0xc0-0xcf
32640. The issues below can be solved with better flags
32641. curr and desc are data
32642. If forced reg, use it as destination. Otherwise try to * use either coerced ispec if it is a temporary. * * When using extraops, avoid reusing arg2 as dest because that * would lead to an LDREG shuffle below. We still can't guarantee * dest != arg2 because we may have a forced_reg.
32643. **comment:** XXX: putvar takes a duk_tval pointer, which is awkward and * should be reworked.
 label: code-design
32644. bottom of current func
32645. statement is guaranteed to be terminal (control doesn't flow to next statement)
32646. Use a fast break/continue when possible. A fast break/continue is * just a jump to the LABEL break/continue jump slot, which then jumps * to an appropriate place (for break, going through ENDLABEL correctly). * The peephole optimizer will optimize the jump to a direct one.
32647. * Heap buffer representation. * * Heap allocated user data buffer which is either: * * 1. A fixed size buffer (data follows header statically) * 2. A dynamic size buffer (data pointer follows header) * * The data pointer for a variable size buffer of zero size may be NULL.
32648. Bernstein hash init value is normally 5381; XOR it in case pointer low bits are 0
32649. **comment:** Allow headroom for calls during error handling (see GH-191). * We allow space for 10 additional recursions, with one extra * for, e.g. a print() call at the deepest level.
 label: code-design
32650. http://en.wikipedia.org/wiki/Exponentiation_by_squaring
32651. chain
32652. [... closure]
32653. **comment:** Calling as a non-constructor is not meaningful.
 label: code-design
32654. * Voluntary periodic GC (if enabled)
32655. Step types
32656. 'default'
32657. **comment:** XXX: awkward; we assume there is space for this, overwrite * directly instead?
 label: code-design
32658. E5 Section steps 7-8
32659. result object is created and discarded; wasteful but saves code space
32660. Gather in little endian
32661. 'fmt'
32662. not caught by anything before entry level; rethrow and let the * final catcher unwind everything
32663. NUL terminate for convenient C access
32664. 'BYTES_PER_ELEMENT'
32665. We don't log or warn about freeing zero refcount objects * because they may happen with finalizer processing.
32666. Don't allow a zero divisor. Restrict both v1 and * v2 to positive values to avoid compiler specific * behavior.
32667. **comment:** XXX: logic for handling character ranges is now incorrect, it will accept * e.g. [\d-z] whereas it should croak from it? SMJS accepts this too, though. * * Needs a read through and a lot of additional tests.
 label: code-design
32668. * Node.js Buffer.prototype.fill()
32669. * Exposed debug macros: debugging disabled
32670. **comment:** No NUL term checks in this debug code.
 label: code-design
32671. DUK_BUFOBJ_FLOAT64ARRAY
32672. embed: duk_hobject ptr
32673. getters
32674. DUK_TOK_DELETE
32675. DUK_TOK_BOR
32676. **comment:** pseudotypes, not used in actual longjmps
 label: code-design
32677. A -> target reg * BC -> inner function index
32678. refcount code is processing refzero list
32679. [... retval]; popped below
32680. 0x80...0x8f
32681. Get current Ecmascript time (= UNIX/Posix time, but in milliseconds).
32682. Note: all references inside 'data' need to get their refcounts * upped too. This is the case because refcounts are decreased * through every function referencing 'data' independently.
32683. **comment:** NOTE: level is used only by the debugger and should never be present * for an Ecmascript eval().
 label: code-design
32684. entry_top + 1
32685. Set datetime parts from stack arguments, defaulting any missing values. * Day-of-week is not set; it is not required when setting the time value.

32686. * ToBoolean() (E5 Section 9.2)
32687. * Shift to new valstack_bottom.
32688. 0 = don't push current value
32689. "." is not accepted in any format
32690. tmp -> res
32691. [body formals source]
32692. **comment:** XXX: for simple cases like x['y'] an unnecessary LDREG is * emitted for the base value; could avoid it if we knew that * the key expression is safe (e.g. just a single literal).
label: code-design
32693. Other initial bytes.
32694. Similar to String comparison.
32695. Finally, blame the innermost callstack entry which has a *.fileName property.
32696. skip finalizers; queue finalizable objects to heap_allocated
32697. Note that duk_exprtop() here can clobber any reg above current temp_next, * so any loop variables (e.g. enumerator) must be "preallocated".
32698. * Parse a statement list. ** Handles automatic semicolon insertion and implicit return value. ** Upon entry, 'curr_tok' should contain the first token of the first * statement (parsed in the "allow regexp literal" mode). Upon exit, * 'curr Tok' contains the token following the statement list terminator * (EOF or closing brace).
32699. fixed top level hashtable size (separate chaining)
32700. not present
32701. end marker
32702. handle negative values
32703. invalidated by pushes, so get out of the way
32704. h is NULL for lightfunc
32705. * Error throwing
32706. Non-buffer value is first ToString() coerced, then converted * to a buffer (fixed buffer is used unless a dynamic buffer is * explicitly requested).
32707. **comment:** unused: not accepted in inbound messages
label: code-design
32708. Reuse buffer as is unless buffer has grown large.
32709. Offset is coerced first to signed integer range and then to unsigned. * This ensures we can add a small byte length (1-8) to the offset in * bound checks and not wrap.
32710. internal align target for props allocation, must be 2*n for some n
32711. Set timeval to 'this' from dparts, push the new time value onto the * value stack and return 1 (caller can then tail call us). Expects * the value stack to contain 'this' on the stack top.
32712. target
32713. DUK_FLD_FLOAT
32714. XXX: shared helper fortoFixed(), toExponential(), toPrecision()
32715. **comment:** * Run an duk_hobject finalizer. Used for both reference counting * and mark-and-sweep algorithms. Must never throw an error. ** There is no return value. Any return value or error thrown by * the finalizer is ignored (although errors are debug logged). ** Notes: ** - The thread used for calling the finalizer is the same as the * 'thr' argument. This may need to change later. ** - The finalizer thread 'top' assertions are there because it is * critical that strict stack policy is observed (i.e. no cruft * left on the finalizer stack).
label: code-design
32716. -> x in [0, 2**32[
32717. rethrow original error
32718. '\xffThis'
32719. Round up digits to a given position. If position is out-of-bounds, * does nothing. If carry propagates over the first digit, a '1' is * prepended to digits and 'k' will be updated. Return value indicates * whether carry propagated over the first digit. ** Note that nc_ctx->count is NOT updated based on the rounding position * (it is updated only if carry overflows over the first digit and an * extra digit is prepended).
32720. Check if any lookup above had a negative result.
32721. DUK_USE_DEBUGGER_SUPPORT
32722. * For/for-in main variants are: ** 1. for (ExpressionNoIn_opt; Expression_opt; Expression_opt) Statement * 2. for (var VariableDeclarationNoIn; Expression_opt; Expression_opt) Statement * 3. for (LeftHandSideExpression in Expression) Statement * 4. for (var VariableDeclarationNoIn in Expression) Statement ** Parsing these without arbitrary lookahead or backtracking is relatively * tricky but we manage to do so for now. ** See doc/compiler.rst for a detailed discussion of control flow * issues, evaluation order issues, etc.
32723. The specification is a bit vague what to do if the return * value is not a number. Other implementations seem to * tolerate non-numbers but e.g. V8 won't apparently do a * ToNumber().
32724. DUK_TOK_SUB_EQ
32725. * Do a longjmp call, calling the fatal error handler if no * catchpoint exists.
32726. Fixed header info.
32727. nothing to finalize
32728. throw flag irrelevant (false in std alg)
32729. **comment:** This approach is a bit shorter than a straight * if-else-ladder and also a bit faster.
label: code-design
32730. All lightfunc own properties are non-writable and the lightfunc * is considered non-extensible. However, the write may be captured * by an inherited setter which means we can't stop the lookup here.
32731. name
32732. accept string if next char is NUL (otherwise reject)
32733. * "WhiteSpace" production check.
32734. value1 -> yield value, iserror -> error / normal
32735. Number of characters that the atom matches (e.g. 3 for 'abc'), * -1 if atom is complex and number of matched characters either * varies or is not known.
32736. **comment:** 'funcs' is quite rarely used, so no local for it
label: code-design
32737. reachable so pop OK
32738. may be NULL, too
32739. No entry in the catchstack which would actually catch a * throw can refer to the callstack entry being reused. * There *can* be catchstack entries referring to the current * callstack entry as long as they don't catch (e.g. label sites).
32740. * Heap creation and destruction
32741. [... regexp_object escaped_source]
32742. now expected to fit into a 32-bit integer
32743. Duktape.modSearch(resolved_id, fresh_require, exports, module).
32744. bits 0...1: shift
32745. Architecture, OS, and compiler strings
32746. overflow
32747. must fit into duk_small_int_t
32748. 'continue'
32749. 'value'
32750. Non-critical.
32751. **comment:** XXX: tail call: return duk_push_false(ctx)
label: code-design

32752. **comment:** * Sweep garbage and remove marking flags, and move objects with * finalizers to the finalizer work list. ** Objects to be swept need to get their refcounts finalized before * they are swept. In other words, their target object refcounts * need to be decreased. This has to be done before freeing any * objects to avoid decreffing dangling pointers (which may happen * even without bugs, e.g. with reference loops) ** Because strings don't point to other heap objects, similar * finalization is not necessary for strings.
label: code-design

32753. Note: target registers a and a+1 may overlap with DUK__REGCONSTP(b) * and DUK__REGCONSTP(c). Careful here.

32754. DUK_USE_DATE_NOW_GETTIMEOFDAY

32755. **comment:** XXX: what's the safest way of creating a negative zero?
label: code-design

32756. **comment:** XXX: several nice-to-have improvements here: * - Use direct reads avoiding value stack operations * - Avoid triggering getters, indicate getter values to debug client * - If side effects are possible, add error catching
label: code-design

32757. ptr may be NULL

32758. E5 Sections 15.10.2.8, 7.3

32759. Target must be stored so that we can recheck whether or not * keys still exist when we enumerate. This is not done if the * enumeration result comes from a proxy trap as there is no * real object to check against.

32760. * Note: currently no fast path for array index writes. * They won't be possible anyway as strings are immutable.

32761. Important to do a fastint check so that constants are * properly read back as fastints.

32762. remove artificial bump

32763. important to use normalized NaN with 8-byte tagged types

32764. Only objects in heap_allocated may have finalizers. Check that * the object itself has a _Finalizer property (own or inherited) * so that we don't execute finalizers for e.g. Proxy objects.

32765. crossed offsets or zero size

32766. [... escaped_source bytecode]

32767. * E5 Section 11.4.8

32768. * Manipulate valstack so that args are on the current bottom and the * previous caller's 'this' binding (which is the value preceding the * current bottom) is replaced with the new 'this' binding: ** [... this_old | (crud) func this_new arg1 ... argN] * --> [... this_new | arg1 ... argN] ** For tail calling to work properly, the valstack bottom must not grow * here; otherwise crud would accumulate on the valstack.

32769. * ToInteger() (E5 Section 9.4)

32770. invalidates tv

32771. * Fatal error * * There are no fatal error macros at the moment. There are so few call * sites that the fatal error handler is called directly.

32772. num_stack_args

32773. property is virtual: used in duk_propdesc, never stored * (used by e.g. buffer virtual properties)

32774. \x0ffVarmap'

32775. Compute (extended) utf-8 length without codepoint encoding validation, * used for string interning. ** NOTE: This algorithm is performance critical, more so than string hashing * in some cases. It is needed when interning a string and needs to scan * every byte of the string with no skipping. Having an ASCII fast path * is useful if possible in the algorithm. The current algorithms were * chosen from several variants, based on x64 gcc -O2 testing. See: * <https://github.com/svaarala/duktape/pull/422> ** NOTE: must match src/dukutil.py:duk_unicode_unvalidated_utf8_length().

32776. Only allow Duktape.Buffer when support disabled.

32777. * Digit generation loop. * * Different termination conditions: * * 1. Free format output. Terminate when shortest accurate * representation found. * * 2. Fixed format output, with specific number of digits. * Ignore termination conditions, terminate when digits * generated. Caller requests an extra digit and rounds. * * 3. Fixed format output, with a specific absolute cut-off * position (e.g. 10 digits after decimal point). Note * that we always generate at least one digit, even if * the digit is below the cut-off point already.

32778. number of digits changed

32779. **comment:** re-lookup to update curr.flags * XXX: would be faster to update directly
label: code-design

32780. * Heap Buffer object representation. Used for all Buffer variants.

32781. **comment:** Note: reuse variables
label: code-design

32782. _Varmap

32783. = 1 + arg count

32784. * Tables generated with src/gennumdigits.py. ** duk__str2num_digits_for_radix indicates, for each radix, how many input * digits should be considered significant for string-to-number conversion. * The input is also padded to this many digits to give the Dragon4 * conversion enough (apparent) precision to work with. ** duk__str2num_exp_limits indicates, for each radix, the radix-specific * minimum/maximum exponent values (for a Dragon4 integer mantissa) * below and above which the number is guaranteed to underflow to zero * or overflow to Infinity. This allows parsing to keep bigint values * bounded.

32785. 'case'

32786. num_pairs and temp_start reset at top of outer loop

32787. Because size > 0, NULL check is correct

32788. 17: getUTCHours

32789. [... enum] -> [... next_key]

32790. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2017 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

32791. Encode to CESU-8; 'out' must have space for at least * DUK_UNICODE_MAX_CESU8_LENGTH bytes; codepoints above U+10FFFF * will encode to garbage but won't overwrite the output buffer.

32792. if density < L, abandon array part, L = 3-bit fixed point, e.g. 2 -> 2/8 = 25%

32793. * Finalize stack

32794. Only a reg fits into 'A' so coerce 'res' into a register * for PUTVAR. ** XXX: here the current A/B/C split is suboptimal: we could * just use 9 bits for reg_res (and support constants) and 17 * instead of 18 bits for the varname const index.

32795. * Helper for setting up var_env and lex_env of an activation, * assuming it does NOT have the DUK_HOBJECT_FLAG_NEWENV flag.

32796. no need for 'caller' post-check, because 'key' must be an array index

32797. [arg undefined]

32798. "

32799. * String-to-number conversion

32800. We can't shortcut zero here if it goes through special formatting * (such as forced exponential notation).

32801. 0x10...0x1f

32802. Main operation

32803. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8CLAMPED * Note: INT8 is -not- copy compatible, e.g. -1 would coerce to 0x00.

32804. invalidated

32805. DUK_USE_CPP_EXCEPTIONS

32806. max(incl)

32807. * Arguments objects have exotic [[DefineOwnProperty]] which updates * the internal 'map' of arguments for writes to currently mapped * arguments. More concretely, writes to mapped arguments generate * a write to a bound variable. * * The [[Put]] algorithm invokes [[DefineOwnProperty]] for existing * data properties and new properties, but not for existing accessors. * Hence, in E5 Section 10.6 ([[DefinedOwnProperty]] algorithm), we * have a Desc with 'Value' (and possibly other properties too), and * we end up in step 5.b.i.

32808. A top-level assignment is e.g. "x = y". For these it's safe * to use the RHS as-is as the expression value, even if the RHS * is a reg-bound identifier. The RHS ('res') is right associative * so it has consumed all other assignment level operations; the * only relevant lower binding power construct is comma operator * which will ignore the expression value provided here. Usually * the top level assignment expression value is ignored, but it * is relevant for e.g. eval code.

32809. **comment:** XXX: use helper for size optimization
label: code-design

32810. 'ignoreCase'

32811. Note: env_thr != thr is quite possible and normal, so careful * with what thread is used for valstack lookup.

32812. frozen and sealed

32813. tv -> value just before prev top value; must relookup

32814. * Found existing inherited plain property. * Do an access control check, and if OK, write * new property to 'orig'.

32815. initial estimate for ASCII only codepoints

32816. Parser separator masks.

32817. just in case callback is broken and won't write 'x'

32818. **comment:** NEXTENUM needs a jump slot right after the main opcode. * We need the code emitter to reserve the slot: if there's * target shuffling, the target shuffle opcodes must happen * after the jump slot (for NEXTENUM the shuffle opcodes are * not needed if the enum is finished).
label: code-design

32819. then to a register

32820. retval to result[i]

32821. Sanity limit on max number of properties (allocated, not necessarily used). * This is somewhat arbitrary, but if we're close to 2**32 properties some * algorithms will fail (e.g. hash size selection, next prime selection). * Also, we use negative array/entry table indices to indicate 'not found', * so anything above 0x80000000 will cause trouble now.

32822. expect EOF instead of }

32823. resizing parameters

32824. value from target

32825. this is especially critical to avoid another write call in detach1()

32826. **comment:** Stripping the filename might be a good idea * ("foo/bar/quux.js" -> logger name "quux"), * but now used verbatim.
label: code-design

32827. return the argument object

32828. new buffer of specified size

32829. * Conversion helpers

32830. %x; only 16 bits are guaranteed

32831. mix-in value for computing string hashes; should be reasonably unpredictable

32832. 'debug'

32833. keep default instance

32834. Make buffer compact, matching current written size.

32835. explicit semi follows

32836. * Packed 8-byte representation

32837. **comment:** XXX: currently no handling for non-allowed identifier characters, * e.g. a '{' in the function name.
label: code-design

32838. No support for lvalues returned from new or function call expressions. * However, these must NOT cause compile-time SyntaxErrors, but run-time * ReferenceErrors. Both left and right sides of the assignment must be * evaluated before throwing a ReferenceError. For instance: * * f() = g(); * * must result in f() being evaluated, then g() being evaluated, and * finally, a ReferenceError being thrown. See E5 Section 11.13.1.

32839. **comment:** XXX: awkward helpers
label: code-design

32840. advance, expecting current token to be a specific token; parse next token in regexp context

32841. stack prepped for func call: [... trap handler]

32842. '{"_nan":true}'

32843. tagged null pointers should never occur

32844. * Heap header definition and assorted macros, including ref counting. * Access all fields through the accessor macros.

32845. * Finally, longjmp

32846. e.g. DUK_OP_POSTINCP

32847. Allow leading minus sign

32848. copy to buffer with spare to avoid Valgrind gripes from strftime

32849. throw

32850. 4: toLocaleDateString

32851. eat 'with'

32852. [... this func]

32853. setters

32854. step 4.c

32855. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)

32856. check func supports [[HasInstance]] (this is checked for every function * in the bound chain, including the final one)

32857. There is no need to NUL delimit the sscanf() call: trailing garbage is * ignored and there is always a NUL terminator which will force an error * if no error is encountered before it. It's possible that the scan * would scan further than between [js_ctx->p,p[though and we'd advance * by less than the scanned value. * * Because pointers are platform specific, a failure to scan a pointer * results in a null pointer which is a better placeholder than a missing * value or an error.

32858. **comment:** XXX: relocate
label: code-design

32859. match fail

32860. * Make a value copy of the input val. This ensures that * side effects cannot invalidate the pointer.

32861. DUK_USE_DATE_TZO_GMTIME

32862. Treat like debugger statement: nop

32863. Get valstack index for the func argument or throw if insane stack.

32864. **comment:** 'd3' is never NaN, so no need to normalize
label: code-design

32865. XXX: push_uint_string / push_u32_string

32866. return value from Duktape.Thread.resume()

32867. For the case n==1 Node.js doesn't seem to type check * the sole member but we do it before returning it. * For this case only the original buffer object is * returned (not a copy).

32868. end of loop (careful with len==0)

32869. Result is always fastint compatible.

32870. Handle a BREAK/CONTINUE opcode. Avoid using longjmp() for BREAK/CONTINUE * handling because it has a measurable performance impact in ordinary * environments and an extreme impact in Emscripten (GH-342).

32871. DUK_TOK_TRY

32872. Optional UTC conversion.

32873. 6
32874. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module]
32875. provideThis=false
32876. * Self test main
32877. prototype is lex_env before catcher created
32878. ToInteger() coercion; NaN -> 0, infinities are clamped to 0 and 10
32879. Keep throwing an error whenever we get here. The unusual values * are set this way because no instruction is ever executed, we just * throw an error until all try/catch/finally and other catchpoints * have been exhausted. Duktape/C code gets control at each protected * call but whenever it enters back into Duktape the RangeError gets * raised. User exec timeout check must consistently indicate a timeout * until we've fully bubbled out of Duktape.
32880. tc1 = true, tc2 = false
32881. no need to handle thread state book-keeping here
32882. Not found. Empty string case is handled specially above.
32883. 4 low bits
32884. **comment:** XXX: option to pretend property doesn't exist if sanity limit is * hit might be useful.
label: code-design
32885. Use explicit steps in computation to try to ensure that * computation happens with intermediate results coerced to * double values (instead of using something more accurate). * E.g. E5.1 Section 15.9.1.11 requires use of IEEE 754 * rules (= Ecmascript 't' and '**' operators). ** Without 'volatile' even this approach fails on some platform * and compiler combinations. For instance, gcc 4.8.1 on Ubuntu * 64-bit, with -m32 and without -std=c99, test-bi-date-canceling.js * would fail because of some optimizations when computing tmp_time * (MakeTime below). Adding 'volatile' to tmp_time solved this * particular problem (annoyingly, also adding debug prints or * running the executable under valgrind hides it).
32886. **comment:** Smaller heapdr than for other objects, because strings are held * in string intern table which requires no link pointers. Much of * the 32-bit flags field is unused by flags, so we can stuff a 16-bit * field in there.
label: code-design
32887. **comment:** Delete semantics are a bit tricky. The description in E5 specification * is kind of confusing, because it distinguishes between resolvability of * a reference (which is only known at runtime) seemingly at compile time * (= SyntaxError throwing).
label: code-design
32888. [...] -> [...]]
32889. * Augment an error being created using Duktape specific properties * like _Tracedata or .fileName/.lineNumber.
32890. misc constants and helper macros
32891. zero-based day
32892. 28: setMinutes
32893. non-object base, no offending virtual property
32894. Fixed seed value used with ROM strings.
32895. duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp literal" mode with current strictness
32896. E5 Sections 11.8.1, 11.8.5; x < y
32897. may be < thr->catchstack initially
32898. complex, multicharacter conversion
32899. Not enough data to provide a full window, so "scroll" window to * start of buffer and fill up the rest.
32900. E5.1 Section 15.1.3.1: uriReserved + '#'
32901. **comment:** XXX: should there be an error or an automatic detach if * already attached?
label: code-design
32902. For user code this could just return 1 (strict) always * because all Duktape/C functions are considered strict, * and strict is also the default when nothing is running. * However, Duktape may call this function internally when * the current activation is an Ecmascript function, so * this cannot be replaced by a 'return 1' without fixing * the internal call sites.
32903. "" was eaten by caller
32904. expression is left-hand-side compatible
32905. string is a reserved word (strict)
32906. **comment:** XXX: Output type could be encoded into native function 'magic' value to * save space. For setters the stridx could be encoded into 'magic'.
label: code-design
32907. **comment:** * Manipulation of thread stacks (valstack, callstack, catchstack). ** Ideally unwinding of stacks should have no side effects, which would * then favor separate unwinding and shrink check primitives for each * stack type. A shrink check may realloc and thus have side effects. ** However, currently callstack unwinding itself has side effects, as it * needs to DECREF multiple objects, close environment records, etc. * Stacks must thus be unwound in the correct order by the caller. ** (XXX: This should be probably reworked so that there is a shared * unwind primitive which handles all stacks as requested, and knows * the proper order for unwinding.) * Valstack entries above 'top' are always kept initialized to * "undefined unused". Callstack and catchstack entries above 'top' * are not zeroed and are left as garbage. ** Value stack handling is mostly a part of the API implementation.
label: code-design
32908. Step 9: copy elements-to-be-deleted into the result array
32909. 0x30-0x3f
32910. 2 bits for heap type
32911. setjmp catchpoint setup
32912. **comment:** not sure whether or not needed; Thursday
label: requirement
32913. emergency mode: try extra hard
32914. Set object 'length'.
32915. byte index limit for element access, exclusive
32916. We could rely on max temp/const checks: if they don't exceed BC * limit, nothing here can either (just asserts would be enough). * Currently we check for the limits, which provides additional * protection against creating invalid bytecode due to compiler * bugs.
32917. require initializer for var/const
32918. DUK_TOK_FALSE
32919. fall-through control flow patchup; note that pc_prevstmt may be * < 0 (i.e. no case clauses), in which case this is a no-op.
32920. [...] eval "eval" eval_input level]
32921. * ToObject() (E5 Section 9.9) * ==> implemented in the API.
32922. DUK_USE_MARKANDSWEEP_FINALIZER_TORTURE
32923. **comment:** * Handle integers in 32-bit range (that is, [-(2**32-1),2**32-1]) * specially, as they're very likely for embedded programs. This * is now done for all radix values. We must be careful not to use * the fast path when special formatting (e.g. forced exponential) * is in force. ** XXX: could save space by supporting radix 10 only and using * sprintf "%lu" for the fast path and for exponent formatting.
label: code-design
32924. !(p < r)
32925. Maximum exponent value when parsing numbers. This is not strictly * compliant as there should be no upper limit, but as we parse the * exponent without a bigint, impose some limit.
32926. num_stack_args
32927. **comment:** Executor interrupt counter check, used to implement breakpoints, * debugging interface, execution timeouts, etc. The counter is heap * specific but is maintained in the current thread to make the check * as fast as possible. The counter is copied back to the heap struct * whenever a thread switch occurs by the DUK_HEAP_SWITCH_THREAD() macro.
label: code-design
32928. complete 'sub atom'
32929. * Intern key via the valstack to ensure reachability behaves * properly. We must avoid longjmp's here so use non-checked * primitives. ** Note: duk_check_stack() potentially reallocs the valstack, * invalidating any duk_tval pointers to valstack. Callers * must be careful.

32930. round out to 8 bytes
32931. for convenience
32932. buffer limits
32933. Handle TypedArray vs. Node.js Buffer arg differences
32934. DUK_TOK_WHILE
32935. offset/value order different from Node.js
32936. * Other cases (function declaration, anonymous function expression, * strict direct eval code). The "outer" environment will be whatever * the caller gave us.
32937. * Setup environment record properties based on the template and * its flags. * * If DUK_HOBJECT_HAS_NEWENV(fun_temp) is true, the environment * records represent identifiers "outside" the function; the * "inner" environment records are created on demand. Otherwise, * the environment records are those that will be directly used * (e.g. for declarations). * * _Lexenv is always set; _Varenv defaults to _Lexenv if missing, * so _Varenv is only set if _Lexenv != _Varenv. * * This is relatively complex, see doc/identifier-handling.rst.
32938. continue matched this label -- we can only continue if this is the empty * label, for which duplication is allowed, and thus there is hope of * finding a match deeper in the label stack.
32939. Plus sign must be accepted for positive exponents * (e.g. '1.5e+2'). This clause catches NULs.
32940. DUK_TOK_VOID
32941. DUK_USE_PC2LINE
32942. DUK_HOBJECT_FLAG_STRICT varies
32943. [... errval errhandler]
32944. **comment:** XXX: Valstack, callstack, and catchstack are currently assumed * to have non-NULL pointers. Relaxing this would not lead to big * benefits (except perhaps for terminated threads).
label: code-design
32945. serialize the wrapper with empty string key
32946. * Free a heap object. * * Free heap object and its internal (non-heap) pointers. Assumes that * caller has removed the object from heap allocated list or the string * intern table, and any weak references (which strings may have) have * been already dealt with.
32947. BITWISE EXPRESSIONS
32948. **comment:** * Copy some internal properties directly * * The properties will be writable and configurable, but not enumerable.
label: code-design
32949. Valstack resize flags
32950. **comment:** XXX: Hex encoded, length limited buffer summary here?
label: code-design
32951. finish up pending jump and split for last alternative
32952. nop
32953. * Local declarations.
32954. stack[0] = object (this) * stack[1] = ToUint32(length) * stack[2] = elem at index 0 (retval)
32955. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers). A prototype loop must not cause * an error to be thrown here; duk_hobject_hasprop_raw() will ignore a * prototype loop silently and indicate that the property doesn't exist.
32956. **comment:** XXX: coerce to regs? it might be better for enumeration use, where the * same PROP ivalue is used multiple times. Or perhaps coerce PROP further * there?
label: code-design
32957. x > y --> y < x
32958. * Found existing accessor property (own or inherited). * Call setter with 'this' set to orig, and value as the only argument. * Setter calls are OK even for ROM objects. * * Note: no exotic arguments object behavior, because [[Put]] never * calls [[DefineOwnProperty]] (E5 Section 8.12.5, step 5.b).
32959. resume: [... initial_func] (currently actually: [initial_func])
32960. Lookup 'key' from arguments internal 'map', and leave replacement value * on stack top if mapped (and return non-zero). * Used in E5 Section 10.6 algorithm for [[GetOwnProperty]] (used by [[Get]]).
32961. proto can also be NULL here (allowed explicitly)
32962. * split()
32963. Validate that the whole slice [0,length[is contained in the underlying * buffer. Caller must ensure 'buf' != NULL.
32964. could also just pop?
32965. Parser part indices.
32966. Compiler SyntaxError (or other error) gets the primary blame. * Currently no flag to prevent blaming.
32967. We don't check the zero padding bytes here right now * (that they're actually zero). This seems to be common * behavior for base-64 decoders.
32968. An object may be in heap_allocated list with a zero * refcount if it has just been finalized and is waiting * to be collected by the next cycle.
32969. * Local prototypes
32970. low-level property functions
32971. helpers for defineProperty() and defineProperties()
32972. w/o refcounts
32973. Skip dvalues to EOM.
32974. all native functions have NEWENV
32975. +1, right after inserted jump
32976. only accept lowercase 'utf8' now.
32977. **comment:** XXX: duplicates should be eliminated here
label: code-design
32978. **comment:** XXX: block removal API primitive
label: code-design
32979. LOGICAL EXPRESSIONS
32980. %lf, %ld etc
32981. * Switch is pretty complicated because of several conflicting concerns: * * - Want to generate code without an intermediate representation, * i.e., in one go * * - Case selectors are expressions, not values, and may thus e.g. throw * exceptions (which causes evaluation order concerns) * * - Evaluation semantics of case selectors and default clause need to be * carefully implemented to provide correct behavior even with case value * side effects * * - Fall through case and default clauses; avoiding dead JUMPs if case * ends with an unconditional jump (a break or a continue) * * - The same case value may occur multiple times, but evaluation rules * only process the first match before switching to a "propagation" mode * where case values are no longer evaluated * * See E5 Section 12.11. Also see doc/compiler.rst for compilation * discussion.
32982. 8: getFullYear
32983. * Debug connection peek and flush primitives
32984. 'do'
32985. duk_xdef_prop() will define an own property without any array * special behaviors. We'll need to set the array length explicitly * in the end. For arrays with elisions, the compiler will emit an * explicit SETALEN which will update the length.
32986. exponential notation forced
32987. NB: 'val' may be invalidated here because put_value may realloc valstack, * caller beware.
32988. Here we'd have the option to normalize -0 to +0.
32989. **comment:** XXX: share helper from lexer; duk_lexer.c / hexval().
label: code-design
32990. **comment:** * Create an internal enumerator object E, which has its keys ordered * to match desired enumeration ordering. Also initialize internal control * properties for enumeration. * * Note: if an array was used to hold enumeration keys instead, an array * scan would be needed to eliminate duplicates found in the prototype chain.
label: code-design

32991. **comment:** This is essentially the 'scale' algorithm, with recursion removed. * Note that 'k' is either correct immediately, or will move in one * direction in the loop. There's no need to do the low/high checks * on every round (like the Scheme algorithm does). ** The scheme algorithm finds 'k' and updates 's' simultaneously, * while the logical algorithm finds 'k' with 's' having its initial * value, after which 's' is updated separately (see the Burger-Dybvig * paper, Section 3.1, steps 2 and 3). ** The case where $m+ == m-$ (almost always) is optimized for, because * it reduces the bigint operations considerably and almost always * applies. The scale loop only needs to work with $m+$, so this works.

label: code-design

32992. A -> flags * BC -> regCatch; base register for two registers used both during * trycatch setup and when catch is triggered ** If DUK_BC_TRYCATCH_FLAG_CATCH_BINDING set: * regCatch + 0: catch binding variable name (string). * Automatic declarative environment is established for * the duration of the 'catch' clause. ** If DUK_BC_TRYCATCH_FLAG_WITH_BINDING set: * regCatch + 0: with 'target value', which is coerced to * an object and then used as a bindind object for an * environment record. The binding is initialized here, for * the 'try' clause. ** Note that a TRYCATCH generated for a 'with' statement has no * catch or finally parts.

32993. default: false

32994. **comment:** fill new_h with u32 0xff = UNUSED

label: code-design

32995. A non-extensible object cannot gain any more properties, * so this is a good time to compact.

32996. if direct eval, calling activation must exist

32997. **comment:** XXX: because of the final check below (that the literal is not * followed by a digit), this could maybe be simplified, if we bail * out early from a leading zero (and if there are no periods etc). * Maybe too complex.

label: code-design

32998. * Create escaped RegExp source (E5 Section 15.10.3). ** The current approach is to special case the empty RegExp * (" -> '(?:')") and otherwise replace unescaped '/' characters * with '\' regardless of where they occur in the regexp. ** Note that normalization does not seem to be necessary for * RegExp literals (e.g. '/foo/') because to be acceptable as * a RegExp literal, the text between forward slashes must * already match the escaping requirements (e.g. must not contain * unescaped forward slashes or be empty). Escaping IS needed * for expressions like 'new RegExp("...","")' however. * Currently, we re-escape in either case. ** Also note that we process the source here in UTF-8 encoded * form. This is correct, because any non-ASCII characters are * passed through without change.

32999. Value stack is used to ensure reachability of constants and * inner functions being loaded. Require enough space to handle * large functions correctly.

33000. **comment:** Automatic defaulting of logger name from caller. This would * work poorly with tail calls, but constructor calls are currently * never tail calls, so tail calls are not an issue now.

label: code-design

33001. Fast check for extending array: check whether or not a slow density check is required.

33002. **comment:** XXX: using duk_put_prop_index() would cause obscure error cases when Array.prototype * has write-protected array index named properties. This was seen as DoubleErrors * in e.g. some test262 test cases. Using duk_xdef_prop_index() is better but heavier. * The best fix is to fill in the tracedata directly into the array part. There are * no side effect concerns if the array part is allocated directly and only INCREFS * happen after that.

label: code-design

33003. filename may be NULL in which case file/line is not recorded

33004. unsigned

33005. **comment:** XXX: function flag to make this automatic?

label: requirement

33006. Must have array part and no conflicting exotic behaviors. * Doesn't need to have array special behavior, e.g. Arguments * object has array part.

33007. Breakpoint entries above the used area are left as garbage.

33008. allow_source_elem

33009. JSON parsing code is allowed to read [p_start,p_end]: p_end is * valid and points to the string NUL terminator (which is always * guaranteed for duk_hstrings).

33010. fall through if overflow etc

33011. inner function numbering

33012. **comment:** XXX: could write output in chunks with fewer ensure calls, * but relative benefit would be small here.

label: code-design

33013. Assumes that saved[0] and saved[1] are always * set by regexp bytecode (if not, char_end_offset * will be zero). Also assumes clen reflects the * correct char length.

33014. **comment:** XXX: V8 throws a TypeError for negative values. Would it * be more useful to interpret negative offsets here from the * end of the buffer too?

label: code-design

33015. function needs shuffle registers

33016. The compiler should never emit DUK_OP_REGEXP if there is no * regexp support.

33017. invalid padding

33018. Note: here we must be wary of the fact that a pointer may be * valid and be a NULL.

33019. **comment:** Here we can choose either to end parsing and ignore * whatever follows, or to continue parsing in case * multiple (possibly padded) base64 strings have been * concatenated. Currently, keep on parsing.

label: code-design

33020. [requested_id require require.id resolved_id last_comp]

33021. advance to get one step of lookup

33022. Write signed 32-bit integer.

33023. * Canonicalize() abstract operation needed for canonicalization of individual * codepoints during regexp compilation and execution, see E5 Section 15.10.2.8. * Note that codepoints are canonicalized one character at a time, so no context * specific rules can apply. Locale specific rules can apply, though.

33024. This should not be necessary because no-one should tamper with the * regexp bytecode, but is prudent to avoid potential segfaults if that * were to happen for some reason.

33025. Needs lookahead

33026. * Churn refzero_list until empty

33027. Note: 'q_instr' is still used below

33028. duk_unicode_caseconv_uc[]

33029. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. ** XXX: this is now an overkill for many fast paths. Rework this * to be faster (although switching to a valstack discipline might * be a better solution overall).

label: code-design

33030. failed, resize and try again

33031. extend with undefined

33032. **comment:** * Traceback handling ** The unified helper decodes the traceback and produces various requested * outputs. It should be optimized for size, and may leave garbage on stack, * only the topmost return value matters. For instance, traceback separator * and decoded strings are pushed even when looking for filename only. ** NOTE: although _Tracedata is an internal property, user code can currently * write to the array (or replace it with something other than an array). * The code below must tolerate arbitrary _Tracedata. It can throw errors * etc, but cannot cause a segfault or memory unsafe behavior.

label: code-design

33033. Error; error value is in heap->lj.value1.

33034. -> [... key val toJSON val key]

33035. callstack

33036. read 3 bytes into 't', padded by zero

33037. XXX: remove this feature entirely? it would only matter for * emergency GC. Disable for lowest memory builds.

33038. * Computed values (e.g. INFINITY)

33039. DUK_TOK_DIV_EQ

33040. already set

33041. input 'd' in Windows UTC, 100ns units

33042. Keep current size

33043. 50x heap size

33044. Dynamic buffer with 'curr_alloc' pointing to a dynamic area allocated using * heap allocation primitives. Also used for external buffers when low memory * options are not used.
33045. [name this]
33046. [... holder name val enum obj_key new_elem]
33047. patched later
33048. [... regexp_object]
33049. **comment:** * Note: the description in E5 Section 15.10.2.6 has a typo, it * contains 'A' twice and lacks 'a'; the intent is [0-9a-zA-Z_].
 label: documentation
33050. This was the last term, and q_last was * updated to match this term at loop top.
33051. x <- b^y; use t1 and t2 as temps
33052. prev_token.start_offset points to the closing brace here; when skipping * we're going to reparse the closing brace to ensure semicolon insertion * etc work as expected.
33053. more initializers
33054. ref.holder is global object, holder is the object with the * conflicting property.
33055. essentially tracks digit position of lowest 'f' digit
33056. **comment:** Prevent any side effects on the string table and the caller provided * str/blen arguments while interning is in progress. For example, if * the caller provided str/blen from a dynamic buffer, a finalizer might * resize that dynamic buffer, invalidating the call arguments.
 label: code-design
33057. binding power "levels" (see doc/compiler.rst)
33058. Enter message processing loop for sending Status notifys and * to finish a pending detach.
33059. result reg
33060. strings are only tracked by stringtable
33061. to avoid relookups
33062. abandoning requires a props allocation resize and * 'rechecks' the valstack, invalidating any existing * valstack value pointers!
33063. If object has a . toJSON() property, we can't be certain * that it wouldn't mutate any value arbitrarily, so bail * out of the fast path. * * If an object is a Proxy we also can't avoid side effects * so abandon.
33064. Reviving an object using a heap pointer is a dangerous API * operation: if the application doesn't guarantee that the * pointer target is always reachable, difficult-to-diagnose * problems may ensue. Try to validate the 'ptr' argument to * the extent possible.
33065. ':'
33066. **comment:** Unlike in duk_hex_encode() 'dst' is not necessarily aligned by 2. * For platforms where unaligned accesses are not allowed, shift 'dst' * ahead by 1 byte to get alignment and then DUK_MEMMOVE() the result * in place. The faster encoding loop makes up the difference. * There's always space for one extra byte because a terminator always * follows the hex data and that's been accounted for by the caller.
 label: code-design
33067. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
33068. **comment:** * 'nregs' registers are allocated on function entry, at most 'nargs' * are initialized to arguments, and the rest to undefined. Arguments * above 'nregs' are not mapped to registers. All registers in the * active stack range must be initialized because they are GC reachable. * 'nargs' is needed so that if the function is given more than 'nargs' * arguments, the additional arguments do not 'clobber' registers * beyond 'nregs' which must be consistently initialized to undefined. * * Usually there is no need to know which registers are mapped to * local variables. Registers may be allocated to variable in any * way (even including gaps). However, a register-variable mapping * must be the same for the duration of the function execution and * the register cannot be used for anything else. * * When looking up variables by name, the '_Varmap' map is used. * When an activation closes, registers mapped to arguments are * copied into the environment record based on the same map. The * reverse map (from register to variable) is not currently needed * at run time, except for debugging, so it is not maintained.
 label: code-design
33069. Slice starting point is beyond current length.
33070. ... target is extensible
33071. duk_unicode_support.c
33072. **comment:** XXX: easier check with less code?
 label: code-design
33073. Used when precision is undefined; also used for NaN (-> "NaN"), * and +/- infinity (-> "Infinity", "-Infinity").
33074. x <- x * y, use t as temp
33075. [... v1 v2 str] -> [... str v2]
33076. Zero 'count' is also allowed to make call sites easier.
33077. **comment:** XXX: size optimize
 label: code-design
33078. l= 0 => post-update for array 'length' (used when key is an array index)
33079. **comment:** has inner functions which may slow access (XXX: this can be optimized by looking at the inner functions)
 label: code-design
33080. **comment:** XXX: need to determine LHS type, and check that it is LHS compatible
 label: code-design
33081. false
33082. * Built-in strings
33083. * Clear (reachable) flags of finalize_list * * We could mostly do in the sweep phase when we move objects from the * heap into the finalize_list. However, if a finalizer run is skipped * during a mark-and-sweep, the objects on the finalize_list will be marked * reachable during the next mark-and-sweep. Since they're already on the * finalize_list, no-one will be clearing their REACHABLE flag so we do it * here. (This now overlaps with the sweep handling in a harmless way.)
33084. * Variant 3
33085. * On second pass, skip the function.
33086. OK for NULL.
33087. Note: key issue here is to re-lookup the base pointer on every attempt. * The pointer being reallocated may change after every mark-and-sweep.
33088. -> [... key val toJSON val]
33089. **comment:** XXX: It would be nice to build the string directly but ToUint16() * coercion is needed so a generic helper would not be very * helpful (perhaps coerce the value stack first here and then * build a string from a duk_tval number sequence in one go?).
 label: code-design
33090. Note: getprop may invoke any getter and invalidate any * duk_tval pointers, so this must be done first.
33091. [obj trap_result res_arr propname]
33092. split1: prefer direct execution (no jump)
33093. in resumer's context
33094. Conceptually we'd extract the plain underlying buffer * or its slice and then do a type mask check below to * see if we should reject it. Do the mask check here * instead to avoid making a copy of the buffer slice.
33095. 'try'
33096. -> [in_val]
33097. **comment:** Currently all built-in native functions are strict. * This doesn't matter for many functions, but e.g. * String.prototype.charAt (and other string functions) * rely on being strict so that their 'this' binding is * not automatically coerced.
 label: code-design
33098. **comment:** * Helper for handling an Ecmascript-to-Ecmascript call or an Ecmascript * function (initial) Duktape.Thread.resume(). * * Compared to normal calls handled by duk_handle_call(), there are a * bunch of differences: * * - the call is never protected * - there is no C recursion depth increase (hence an "ignore recursion * limit" flag is not applicable) * - instead of making the call, this helper just performs the thread * setup and returns; the bytecode executor then restarts execution * internally * - ecmascript functions are never 'vararg' functions (they access * varargs through the 'arguments' object) * * The callstack of the target contains an earlier Ecmascript call in case * of an Ecmascript-to-Ecmascript call (whose idx_retval is updated), or * is empty in case of an initial

Duktape.Thread.resume(). * * The first thing to do here is to figure out whether an ecma-to-ecma * call is actually possible. It's not always the case if the target is * a bound function; the final function may be native. In that case, * return an error so caller can fall back to a normal call path.

label: code-design

33099. Note: 0xff != DUK_BC_B_MAX

33100. * Unions and structs for self tests

33101. Note: masking is done for 'i' to deal with negative numbers correctly

33102. * For property layout 1, tweak e_size to ensure that the whole entry * part (key + val + flags) is a suitable multiple for alignment * (platform specific). * * Property layout 2 does not require this tweaking and is preferred * on low RAM platforms requiring alignment.

33103. don't run finalizers; queue finalizable objects back to heap_allocated

33104. val1 = count

33105. * Function declaration, function expression, or (non-standard) * function statement. * * The E5 specification only allows function declarations at * the top level (in "source elements"). An ExpressionStatement * is explicitly not allowed to begin with a "function" keyword * (E5 Section 12.4). Hence any non-error semantics for such * non-top-level statements are non-standard. Duktape semantics * for function statements are modelled after V8, see * test-dev-func-decl-outside-top.js.

33106. Get minimum array part growth for a certain size.

33107. **comment:** XXX: very slow - better to bulk allocate a gap, and copy * from args_array directly (we know it has a compact array * part, etc).

label: code-design

33108. enumerate'

33109. DUK_TOK_RSHIFT

33110. roughly 256 bytes

33111. **comment:** For now, just use duk_safe_call() to wrap duk_new(). We can't * simply use a protected duk_handle_call() because there's post * processing which might throw. It should be possible to ensure * the post processing never throws (except in internal errors and * out of memory etc which are always allowed) and then remove this * wrapper.

label: code-design

33112. object is now reachable

33113. flags: "im"

33114. Copy trap result keys into the enumerator object.

33115. * Mark-and-sweep flags * * These are separate from heap level flags now but could be merged. * The heap structure only contains a 'base mark-and-sweep flags' * field and the GC caller can impose further flags.

33116. Success path.

33117. * If entering a 'catch' block which requires an automatic * catch variable binding, create the lexical environment. * * The binding is mutable (= writable) but not deletable. * Step 4 for the catch production in E5 Section 12.14; * no value is given for CreateMutableBinding 'D' argument, * which implies the binding is not deletable.

33118. t1 <- (* r B)

33119. DUK_BUFOBJ_UINT8ARRAY

33120. Accept any duk_hbufferobject, though we're only normally * called for Duktape.Buffer values.

33121. Lenient: allow function declarations outside top level in * non-strict mode but reject them in strict mode.

33122. assertions: env must be closed in the same thread as where it runs

33123. EOF (-1) will be cast to an unsigned value first * and then re-cast for the switch. In any case, it * will match the default case (syntax error).

33124. refzero not running -> must be empty

33125. **comment:** XXX: differentiate null pointer from empty string?

label: code-design

33126. Normal object which doesn't get automatically coerced to a * primitive value. Functions are checked for specially. The * primitive value coercions for Number, String, Pointer, and * Boolean can't result in functions so suffices to check here.

33127. DUK_USE_MARK_AND_SWEEP

33128. * Coercion and fast path processing

33129. y == +Infinity

33130. caller required to know

33131. * Function name (if any) * * We don't check for prohibited names here, because we don't * yet know whether the function will be strict. Function body * parsing handles this retroactively. * * For function expressions and declarations function name must * be an Identifier (excludes reserved words). For setter/getter * it is a PropertyName which allows reserved words and also * strings and numbers (e.g. "{ get 1) { ... } }").

33132. **comment:** XXX: There are no format strings in duk_config.h yet, could add * one for formatting duk_int64_t. For now, assumes "%lld" and that * "long long" type exists. Could also rely on C99 directly but that * won't work for older MSVC.

label: code-design

33133. toLocaleString'

33134. * Manipulate value stack so that exactly 'num_stack_rets' return * values are at 'idx_rebase' in every case, assuming there are * 'rc' return values on top of stack. * * This is a bit tricky, because the called C function operates in * the same activation record and may have e.g. popped the stack * empty (below idx_rebase).

33135. Exotic behaviors are only enabled for arguments objects * which have a parameter map (see E5 Section 10.6 main * algorithm, step 12). * * In particular, a non-strict arguments object with no * mapped formals does *NOT* get exotic behavior, even * for e.g. "caller" property. This seems counterintuitive * but seems to be the case.

33136. buffer is automatically zeroed

33137. num_args

33138. DUK_OP_EXTRA, sub-operation in A

33139. insert the required matches (qmin) by copying the atom

33140. DUK_USE_INTEGER_BE

33141. out_clamped

33142. adv = 2 default OK

33143. There is no need to decref anything else than 'env': if 'env' * becomes unreachable, refzero will handle decref'ing its prototype.

33144. '-0'

33145. module table remains registered to modLoaded, minimize its size

33146. * Detach handling

33147. write back

33148. [... filename (temps) func]

33149. **comment:** It's not strictly necessary to track the current size, but * it is useful for writing robust native code.

label: code-design

33150. * Second pass parsing.

33151. **comment:** Note: cannot portably debug print a function pointer, hence 'func' not printed!

label: code-design

33152. * Arbitrary byteswap for potentially unaligned values * * Used to byteswap pointers e.g. in debugger code.

33153. **comment:** XXX: There's some overlap with duk_js_closure() here, but * seems difficult to share code. Ensure that the final function * looks the same as created by duk_js_closure().

label: code-design

33154. **comment:** It is important not to call this if the last byte read was an EOM. * Reading ahead in this scenario would cause unnecessary blocking if * another message is not available.

label: code-design

33155. DUK_USE_EXEC_TIMEOUT_CHECK

33156. * Misc defines

33157. E5.1 Section 15.1.3.4: uriUnescaped

33158. Combined size of separators already overflows

33159. -> [... res]
33160. [... this name message]
33161. "/" and not in regexp mode
33162. DUK_USE_BYTECODE_DUMP_SUPPORT
33163. **comment:** may indirectly slow access through a direct eval
 label: code-design
33164. no incref needed
33165. variable binding register if register-bound (otherwise < 0)
33166. How much to advance before next loop; note that next loop * will advance by 1 anyway, so -1 from the total escape * length (e.g. len("\uXXXXX") - 1 = 6 - 1). As a default, * 1 is good.
33167. **comment:** 'sce' points to the wrong entry here, but is no longer used
 label: code-design
33168. Note: this check relies on the fact that even a zero-size string * has a non-NUL pointer.
33169. [... varname val this] (because throw_flag == 1, always resolved)
33170. No need to reinit setjmp() catchpoint, as call handling * will store and restore our state.
33171. embed: duk_hbuffer ptr
33172. i >= 0 always true, unsigned
33173. Maximum expansion per input byte is 6: * - invalid UTF-8 byte causes "\uXXXX" to be emitted (6/1 = 6). * - 2-byte UTF-8 encodes as "\uXXXX" (6/2 = 3). * - 4-byte UTF-8 encodes as "\Uxxxxxxxx" (10/4 = 2.5).
33174. Shared helper to implement Object.getPrototypeOf and the ES6 * Object.prototype.__proto__ getter. ** http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__
33175. xxx -> DUK_HBUFFEROBJECT_ELEM_INT16
33176. vararg function, careful arg handling (e.g. thisArg may not be present)
33177. implicit this value always undefined for * declarative environment records.
33178. **comment:** * IsCallable() (E5 Section 9.11) ** XXX: API equivalent is a separate implementation now, and this has * currently no callers.
 label: code-design
33179. Extract 'top' bits of currvil; note that the extracted bits do not need * to be cleared, we just ignore them on next round.
33180. duk_unicode_ids_noabmp[]
33181. Return value of 'sz' or more indicates output was (potentially) * truncated.
33182. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers).
33183. Encoding/decoding flags
33184. Shared helper to implement ES6 Object.setPrototypeOf and * Object.prototype.__proto__ setter. ** [http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.setPrototypeOf](http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__)
33185. no need to write 'out_bufdata'
33186. new_used / size <= 1 / DIV <=> new_used <= size / DIV
33187. Register the module table early to modLoaded[] so that we can * support circular references even in modSearch(). If an error * is thrown, we'll delete the reference.
33188. label handling
33189. Expression
33190. **comment:** This is performance critical because it appears in every DECREF.
 label: code-design
33191. code_idx = entry_top + 0
33192. -> [... lval rval]
33193. * Standard algorithm succeeded without errors, check for exotic post-behaviors. ** Arguments exotic behavior in E5 Section 10.6 occurs after the standard * [[DefineOwnProperty]] has completed successfully. ** Array exotic behavior in E5 Section 15.4.5.1 is implemented partly * prior to the default [[DefineOwnProperty]], but: * - for an array index key (e.g. "10") the final 'length' update occurs here * - for 'length' key the element deletion and 'length' update occurs here
33194. * All defined array-indexed properties are in the array part * (we assume the array part is comprehensive), and all array * entries are writable, configurable, and enumerable. Thus, * nothing can prevent array entries from being deleted.
33195. **comment:** * Helpers for creating and querying pc2line debug data, which * converts a bytecode program counter to a source line number. ** The run-time pc2line data is bit-packed, and documented in: ** doc/function-objects.rst
 label: documentation
33196. Represent a null pointer as 'null' to be consistent with * the JX format variant. Native '%op' format for a NULL * pointer may be e.g. '(nil)'.
33197. eat 'throw'
33198. always grow the array, no sparse / abandon support here
33199. leave formals on stack for later use
33200. since index non-negative
33201. tzoffset
33202. duk_has_prop() popped the second 'name'
33203. varname
33204. * E5 Section 15.10.2.6 "IsWordChar" abstract operation. Assume * x < 0 for characters read outside the string.
33205. -----XXXX
33206. stack top: new time value, return 1 to allow tail calls
33207. basereg
33208. E5 Section 15.4.5.1, step 3, steps a - i are implemented here, j - n at the end
33209. jump for 'catch' case
33210. Module object containing module.exports, etc
33211. 'prototype' property for all built-in objects (which have it) has attributes: * [[Writable]] = false, * [[Enumerable]] = false, * [[Configurable]] = false
33212. stack[0] = start * stack[1] = deleteCount * stack[2...nargs-1] = items * stack[nargs] = ToObject(this) -3 * stack[nargs+1] = ToUint32(length) -2 * stack[nargs+2] = result array -1
33213. -> [... escaped_source bytecode]
33214. * Write to array part? ** Note: array abandoning requires a property resize which uses * 'rechecks' valstack for temporaries and may cause any existing * valstack pointers to be invalidated. To protect against this, * tv_obj, tv_key, and tv_val are copies of the original inputs.
33215. copy existing entries as is
33216. 5: toLocaleTimeString
33217. mark-and-sweep: children not processed
33218. Catches EOF of JSON input.
33219. radix
33220. **comment:** XXX: when optimizing for guaranteed property slots, use a guaranteed * slot for internal value; this call can then access it directly.
 label: code-design
33221. no wrapping
33222. **comment:** * Create and throw an error (originating from Duktape internally) ** Push an error object on top of the stack, possibly throw augmenting * the error, and finally longjmp. ** If an error occurs while we're dealing with the current error, we might * enter an infinite recursion loop. This is prevented by detecting a * "double fault" through the heap->handling_error flag; the recursion * then stops at the second level.
 label: code-design
33223. if property key begins with underscore, encode it with * forced quotes (e.g. "_Foo") to distinguish it from encoded * internal properties (e.g. \xffBar -> _Bar).
33224. object or array
33225. no need to check now: both success and error are OK
33226. Note: allow backtracking from p == ptr_end

33227. Relookup if relocated
33228. module.filename for .fileName, default to resolved ID if * not present.
33229. -> [... toLocaleString ToObject(val)]
33230. DUK_TOK_LBRACKET already eaten, current token is right after that
33231. **comment:** * Array needs to grow, but we don't want it becoming too sparse. * If it were to become sparse, abandon array part, moving all * array entries into the entries part (for good). ** Since we don't keep track of actual density (used vs. size) of * the array part, we need to estimate somehow. The check is made * in two parts: ** - Check whether the resize need is small compared to the * current size (relatively); if so, resize without further * checking (essentially we assume that the original part is * "dense" so that the result would be dense enough). ** - Otherwise, compute the resize using an actual density * measurement based on counting the used array entries.
label: code-design
33232. Restore stack top after unbalanced code paths.
33233. position of highest digit changed
33234. * duk_heap allocation and freeing.
33235. Careful with borrow condition: * - If borrow not subtracted: 0x12345678 - 0 - 0xffffffff = 0x12345679 (> 0x12345678) * - If borrow subtracted: 0x12345678 - 1 - 0xffffffff = 0x12345678 (== 0x12345678)
33236. no statement value (unlike function expression)
33237. Three digit octal escape, digits validated.
33238. **comment:** * Addition operator is different from other arithmetic * operations in that it also provides string concatenation. * Hence it is implemented separately. ** There is a fast path for number addition. Other cases go * through potentially multiple coercions as described in the * E5 specification. It may be possible to reduce the number * of coercions, but this must be done carefully to preserve * the exact semantics. ** E5 Section 11.6.1. ** Custom types also have special behavior implemented here.
label: code-design
33239. Quite approximate but should be useful for little and big endian.
33240. For X <op>= Y we need to evaluate the pre-op * value of X before evaluating the RHS: the RHS * can change X, but when we do <op> we must use * the pre-op value.
33241. Count is incremented before DUK__DRAGON4_OUTPUT_PREINC() call * on purpose, which is taken into account by the macro.
33242. NUL also comes here. Comparison order matters, 0x20 * is most common whitespace.
33243. retval for success path
33244. * Wrapping duk_safe_call() will mangle the stack, just return stack top
33245. * Node.js Buffer.prototype: toJSON()
33246. We'll need to interrupt early so recompute the init * counter to reflect the number of bytecode instructions * executed so that step counts for e.g. debugger rate * limiting are accurate.
33247. 'fileName'
33248. * Convert duk_compiler_func to a function template and add it * to the parent function table.
33249. parse args starting from "next temp"
33250. This seems like a good overall approach. Fast path for ASCII in 4 byte * blocks.
33251. * Mark-and-sweep garbage collection.
33252. **comment:** * Object compaction (emergency only). ** Object compaction is a separate step after sweeping, as there is * more free memory for it to work with. Also, currently compaction * may insert new objects into the heap allocated list and the string * table which we don't want to do during a sweep (the reachability * flags of such objects would be incorrect). The objects inserted * are currently: ** - a temporary duk_hbuffer for a new properties allocation * - if array part is abandoned, string keys are interned * * The object insertions go to the front of the list, so they do not * cause an infinite loop (they are not compacted).
label: code-design
33253. **comment:** unnecessary div for last bye
label: code-design
33254. buffer values are coerced first to string here
33255. sanity limit for function pointer size
33256. [sep ToObject(this) len sep str0 ... str(count-1)]
33257. DUK_TOK_YIELD
33258. **comment:** XXX: is valstack top best place for argument?
label: code-design
33259. reset bytecode buffer but keep current size; pass 2 will * require same amount or more.
33260. duk_hobject_run_finalizer() sets
33261. [source template closure]
33262. Successful return: restore jmpbuf and return to caller.
33263. negative -> no atom matched on previous round
33264. [... name]
33265. Note: may be NULL if first call
33266. unresolvable, no stack changes
33267. **comment:** The DataView .buffer property is ordinarily set to the argument * which is an ArrayBuffer. We accept any duk_hbufferobject as * an argument and .buffer will be set to the argument regardless * of what it is. This may be a bit confusing if the argument * is e.g. a DataView or another TypedArray view. ** XXX: Copy .buffer property from a DataView/TypedArray argument? * Create a fresh ArrayBuffer for Duktape.Buffer and Node.js Buffer * arguments? See: test-bug-dataview-buffer-prop.js.
label: code-design
33268. DUK_USE_DATE_NOW_TIME
33269. XXX: valstack handling is awkward. Add a valstack helper which * avoids dup():ing; valstack_copy(src, dst)?
33270. On entry, item at idx_func is a bound, non-lightweight function, * but we don't rely on that below.
33271. * Reference counting implementation.
33272. use strict equality instead of non-strict equality
33273. This assert depends on the input parts representing time inside * the Ecmascript range.
33274. controls for minimum array part growth
33275. -> [... key]
33276. * Returns non-zero if a key and/or value was enumerated, and: ** [enum] -> [key] (get_value == 0) * [enum] -> [key value] (get_value == 1) ** Returns zero without pushing anything on the stack otherwise.
33277. allow basic ASCII whitespace
33278. Heap allocated: return heap pointer which is NOT useful * for the caller, except for debugging.
33279. activation is a direct eval call
33280. current lex_env of the activation (created for catcher)
33281. **comment:** Figure out all active breakpoints. A breakpoint is * considered active if the current function's fileName * matches the breakpoint's fileName, AND there is no * inner function that has matching line numbers * (otherwise a breakpoint would be triggered both * inside and outside of the inner function which would * be confusing). Example: ** function foo() { * print('foo'); * function bar() { <-.. breakpoints in these * print('bar'); | lines should not affect * } <-' foo() execution * bar(); * } ** We need a few things that are only available when * debugger support is enabled: (1) a line range for * each function, and (2) access to the function * template to access the inner functions (and their * line ranges). ** It's important to have a narrow match for active * breakpoints so that we don't enter checked execution * when that's not necessary. For instance, if we're * running inside a certain function and there's * breakpoint outside in (after the call site), we * don't want to slow down execution of the function.
label: code-design
33282. start byte offset of token in lexer input
33283. * Note: lj.value1 is 'value', lj.value2 is 'resume'. * This differs from YIELD.
33284. curr_token = get/set name

33285. Initial '[' has been checked and eaten by caller.
33286. Thread may have changed, e.g. YIELD converted to THROW.
33287. Lookup to encode one byte directly into 2 characters: * * def genhextab(bswap): * for i in xrange(256): * t = chr(i).encode('hex') * if bswap: * t = t[1] + t[0] * print('0x' + t.encode('hex') + 'U') * print('big endian'); genhextab(False) * print('little endian'); genhextab(True)
33288. * 64-bit arithmetic * * There are some platforms/compilers where 64-bit types are available * but don't work correctly. Test for known cases.
33289. be paranoid for 32-bit time values (even avoiding negative ones)
33290. * Node.js Buffer.byteLength()
33291. emit instruction; could return PC but that's not needed in the majority * of cases.
33292. Fast write calls which assume you control the spare beforehand. * Multibyte write variants exist and use a temporary write pointer * because byte writes alias with anything: with a stored pointer * explicit pointer load/stores get generated (e.g. gcc -Os).
33293. All the flags fit in 16 bits, so will fit into duk_bool_t.
33294. flags: "g"
33295. For cross-checking during development: ensure dispatch count * matches cumulative interrupt counter init value sums.
33296. valstack should not need changes
33297. end of _Varmap
33298. **comment:** Certain boxed types are required to go through * automatic unboxing. Rely on internal value being * sane (to avoid infinite recursion).
 label: code-design
33299. [-0x80000000,0x7fffffff]
33300. DUK_HOBJECT_FLAG_EXOTIC_STRINGOBJ varies
33301. 'Arguments'
33302. [key getter this] -> [key retval]
33303. Retry allocation after mark-and-sweep for this * many times. A single mark-and-sweep round is * not guaranteed to free all unreferenced memory * because of finalization (in fact, ANY number of * rounds is strictly not enough).
33304. only functions can have arguments
33305. E5 Section 9.3.1
33306. * GETVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is GetValue'd] * 8.7.1 GetValue (V) * 8.12.1 [[GetOwnProperty]] (P) * 8.12.2 [[GetProperty]] (P) * 8.12.3 [[Get]] (P) * * If 'throw' is true, always leaves two values on top of stack: [val this]. * * If 'throw' is false, returns 0 if identifier cannot be resolved, and the * stack will be unaffected in this case. If identifier is resolved, returns * 1 and leaves [val this] on top of stack. * * Note: the 'strict' flag of a reference returned by GetIdentifierReference * is ignored by GetValue. Hence we don't take a 'strict' parameter. * * The 'throw' flag is needed for implementing 'typeof' for an unreferenced * identifier. An unreferenced identifier in other contexts generates a * ReferenceError.
33307. **comment:** XXX: top of stack must contain value, which helper doesn't touch, * rework to use tv_val directly?
 label: code-design
33308. * Executor interrupt handling * * The handler is called whenever the interrupt countdown reaches zero * (or below). The handler must perform whatever checks are activated, * e.g. check for cumulative step count to impose an execution step * limit or check for breakpoints or other debugger interaction. * * When the actions are done, the handler must reinits the interrupt * init and counter values. The 'init' value must indicate how many * bytecode instructions are executed before the next interrupt. The * counter must interface with the bytecode executor loop. Concretely, * the new init value is normally one higher than the new counter value. * For instance, to execute exactly one bytecode instruction the init * value is set to 1 and the counter to 0. If an error is thrown by the * interrupt handler, the counters are set to the same value (e.g. both * to 0 to cause an interrupt when the next bytecode instruction is about * to be executed after error handling). * * Maintaining the init/counter value properly is important for accurate * behavior. For instance, executor step limit needs a cumulative step * count which is simply computed as a sum of 'init' values. This must * work accurately even when single stepping.
33309. 0xd0...0xdf
33310. tv1 is -below- valstack_bottom
33311. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT32
33312. [obj Properties]
33313. * First attempt
33314. not found
33315. Loop check.
33316. XXX: just assert non-NULL values here and make caller arguments * do the defaulting to the default implementations (smaller code)?
33317. reserved words: additional future reserved words in strict mode
33318. * String table resize check. * * Note: this may silently (and safely) fail if GC is caused by an * allocation call in stringtable resize_hash(). Resize_hash() * will prevent a recursive call to itself by setting the * DUK_MS_FLAG_NO_STRINGTABLE_RESIZE in heap->mark_and_sweep_base_flags.
33319. Sync and NULL early.
33320. 12 to 21
33321. **comment:** XXX: we should never shrink here; when we error out later, we'd * need to potentially grow the value stack in error unwind which could * cause another error.
 label: code-design
33322. -> pushes fixed buffer
33323. **comment:** XXX: improve check; check against nregs, not against top
 label: code-design
33324. byte sizes will match
33325. max # of values initialized in one MPUTARR set
33326. not reached
33327. -> [... this]
33328. * Local defines and forward declarations.
33329. **comment:** prev value must be unused, no decref
 label: code-design
33330. Calling duk_debugger_cooperate() while Duktape is being * called into is not supported. This is not a 100% check * but prevents any damage in most cases.
33331. Note: careful with indices like '-x'; if 'x' is zero, it refers to bottom
33332. x >= y --> not (x < y)
33333. yes, must set array length explicitly
33334. * Post resize assertions.
33335. Note: for dstlen=0, dst may be NULL
33336. !DUK_USE_TRACEBACKS
33337. -> [func funcname env funcname]
33338. Constants for duk_lexer_ctx.buf.
33339. Difference is in 100ns units, convert to milliseconds w/o fractions
33340. * "IdentifierPart" production check.
33341. prev exists and is not a letter
33342. Note: no recursion issue, we can trust 'map' to behave
33343. if fails, e_size will be zero = not an issue, except performance-wise
33344. **comment:** XXX: use a NULL error handler for the finalizer call?
 label: code-design
33345. **comment:** workaround: max nbis = 24 now
 label: code-design
33346. DUK_USE_HEX_FASTPATH
33347. DUK_UTIL_H_INCLUDED
33348. 'Infinity'

33349. We know _Varmap only has own properties so walk property * table directly. We also know _Varmap is dense and all * values are numbers; assert for these. GC and finalizers * shouldn't affect _Varmap so side effects should be fine.

33350. debugger state, only relevant when attached

33351. highest value of temp_reg (temp_max - 1 is highest used reg)

33352. * Init compiler and lexer contexts

33353. Input values are signed 48-bit so we can detect overflow * reliably from high bits or just a comparison.

33354. pre/post opcode values have constraints,

33355. allocated heap objects

33356. re-read to avoid spill / fetch

33357. LRU: move our entry to first

33358. argument/function declaration shadows 'arguments'

33359. Proxy target

33360. unconditional block

33361. **comment:** XXX: Using a string return value forces a string intern which is * not always necessary. As a rough performance measure, hex encode * time for tests/perf/test-hex-encode.js dropped from ~35s to ~15s * without string coercion. Change to returning a buffer and let the * caller coerce to string if necessary? **label:** code-design

33362. * Node.js Buffer.prototype.copy()

33363. DUK_VERBOSE_ERRORS

33364. skip ''

33365. XXX: additional conditions when to close variables? we don't want to do it * unless the environment may have "escaped" (referenced in a function closure). * With delayed environments, the existence is probably good enough of a check.

33366. When debugger is enabled, we need to recheck the activation * status after returning. This is now handled by call handling * and heap->dbg_force_restart.

33367. 'byteLength'

33368. * Parsing implementation. * * JSON lexer is now separate from duk_lexer.c because there are numerous * small differences making it difficult to share the lexer. * The parser here works with raw bytes directly; this works because all * JSON delimiters are ASCII characters. Invalid xUTF-8 encoded values * inside strings will be passed on without normalization; this is not a * compliance concern because compliant inputs will always be valid * CESU-8 encodings.

33369. * Detect iteration statements; if encountered, establish an * empty label.

33370. skip ')

33371. * Poppers

33372. DUK_HOBJECT_H_INCLUDED

33373. always at least the header

33374. A -> target register * B -> bytecode (also contains flags) * C -> escaped source

33375. Fast path, decode as is.

33376. * "/" followed by something in regexp mode. See E5 Section 7.8.5. * * RegExp parsing is a bit complex. First, the regexp body is delimited * by forward slashes, but the body may also contain forward slashes as * part of an escape sequence or inside a character class (delimited by * square brackets). A mini state machine is used to implement these. * * Further, an early (parse time) error must be thrown if the regexp * would cause a run-time error when used in the expression new RegExp(...). * Parsing here simply extracts the (candidate) regexp, and also accepts * invalid regular expressions (which are delimited properly). The caller * (compiler) must perform final validation and regexp compilation. * * RegExp first char may not be '/' (single line comment) or '*' (multi- * line comment). These have already been checked above, so there is no * need below for special handling of the first regexp character as in * the E5 productions. * * About unicode escapes within regexp literals: * * E5 Section 7.8.5 grammar does NOT accept \uHHHH escapes. * However, Section 6 states that regexps accept the escapes, * see paragraph starting with "In string literals...". * The regexp grammar, which sees the decoded regexp literal * (after lexical parsing) DOES have a \uHHHH unicode escape. * So, for instance: * * \u1234/* * should first be parsed by the lexical grammar as: * * \'u' RegularExpressionBackslashSequence * '1' RegularExpressionNonTerminator * '2' RegularExpressionNonTerminator * '3' RegularExpressionNonTerminator * '4' RegularExpressionNonTerminator * * and the escape itself is then parsed by the regexp engine. * This is the current implementation. * * Minor spec inconsistency: * * E5 Section 7.8.5 RegularExpressionBackslashSequence is: * * \ RegularExpressionNonTerminator * * while Section A.1 RegularExpressionBackslashSequence is: * * \ NonTerminator * * The latter is not normative and a typo. *

33377. 'break'

33378. in yielder's context

33379. * E5 Section 15.4.5.1, steps 3.k - 3.n. The order at the end combines * the error case 3.l.iii and the success case 3.m-3.n. * * Note: 'length' is always in entries part, so no array abandon issues for * 'writable' update.

33380. * Avoid a GC if GC is already running. This can happen at a late * stage in a GC when we try to e.g. resize the stringtable * or compact objects.

33381. Reject attempt to change virtual properties: not part of the * standard algorithm, applies currently to e.g. virtual index * properties of buffer objects (which are virtual but writable). * (Cannot "force" modification of a virtual property.)

33382. MakeDay

33383. Push 'this' binding, check that it is a Date object; then push the * internal time value. At the end, stack is: [... this timeval]. * Returns the time value. Local time adjustment is done if requested.

33384. Cannot use duk_to_string() on the buffer because it is usually * larger than 'len'. Also, 'buf' is usually a stack buffer.

33385. non-leap year: sunday, monday, ...

33386. activation has tail called one or more times

33387. extended types: custom encoding

33388. reg

33389. -> [... enum key val]

33390. EOF

33391. XXX: optimize value stack operation

33392. * Proxy object handling

33393. * Then decode the builtins init data (see genbuiltins.py) to * init objects

33394. mark as complex (capture handling)

33395. string is a reserved word (non-strict)

33396. s <- b^(1-e) * 2

33397. * Object inspection commands: GetHeapObjInfo, GetObjPropDesc, * GetObjPropDescRange

33398. * Unreachable object about to be swept. Finalize target refcounts * (objects which the unreachable object points to) without doing * refzero processing. Recursive decrefs are also prevented when * refzero processing is disabled. * * Value cannot be a finalizable object, as they have been made * temporarily reachable for this round.

33399. Map DUK_FLX_xxx to byte size.

33400. Note that we currently parse -bytes-, not codepoints. * All non-ASCII extended UTF-8 will encode to bytes >= 0x80, * so they'll simply pass through (valid UTF-8 or not).

33401. * Get/require

33402. key, getter and setter, now reachable through object

33403. **comment:** * Finalize objects in the finalization work list. Finalized * objects are queued back to heap_allocated with FINALIZED set. * * Since finalizers may cause arbitrary side effects, they are * prevented during string table and object property allocation * resizing using the DUK_MS_FLAG_NO_FINALIZERS flag in * heap->mark_and_sweep_base_flags. In this case the objects * remain in the finalization work list after mark-and-sweep * exits and they may be finalized on the next pass. * * Finalization currently happens inside "MARKANDSWEET_RUNNING" * protection (no mark-and-sweep may be triggered by the * finalizers). As a side effect: * * 1) an out-of-memory error inside a finalizer will not * cause a mark-and-sweep and may cause the finalizer * to fail unnecessarily * * 2) any temporary objects whose refcount decreases to zero * during finalization will not be put into refzero_list; * they can only be collected by another mark-and-sweep * * This is not optimal, but since the sweep for this phase has * already happened, this is probably good enough for now. **label:** code-design

33404. * A Dragon4 number-to-string variant, based on: * * Guy L. Steele Jr., Jon L. White: "How to Print Floating-Point Numbers * Accurately" * * Robert G. Burger, R. Kent Dybvig: "Printing Floating-Point Numbers * Quickly and Accurately" * * The current algorithm is based on Figure 1 of the Burger-Dybvig paper, * i.e.

the base implementation without logarithm estimation speedups * (these would increase code footprint considerably). Fixed-format output * does not follow the suggestions in the paper; instead, we generate an * extra digit and round-with-carry. ** The same algorithm is used for number parsing (with b=10 and B=2) * by generating one extra digit and doing rounding manually. ** See doc/number-conversion.rst for limitations.

33405. holder will be set to the target object, not the actual object * where the property was found (see duk_get_identifier_reference()).

33406. 'Array'

33407. * INITSET/INITGET are only used to initialize object literal keys. * The compiler ensures that there cannot be a previous data property * of the same name. It also ensures that setter and getter can only * be initialized once (or not at all).

33408. extended utf-8 not allowed for URIs

33409. * E5 Section 15.7.2.1 requires that the constructed object * must have the original Number.prototype as its internal * prototype. However, since Number.prototype is non-writable * and non-configurable, this doesn't have to be enforced here: * The default object (bound to 'this') is OK, though we have * to change its class. * * Internal value set to ToNumber(arg) or +0; if no arg given, * ToNumber(undefined) = NaN, so special treatment is needed * (above). String internal value is immutable.

33410. bytecode limits

33411. guaranteed to find an empty slot

33412. Because 'day since epoch' can be negative and is used to compute weekday * using a modulo operation, add this multiple of 7 to avoid negative values * when year is below 1970 epoch. Ecmascript time values are restricted to * +/- 100 million days from epoch, so this adder fits nicely into 32 bits. * Round to a multiple of 7 (= floor(100000000 / 7) * 7) and add margin.

33413. Allow trailing garbage (e.g. treat "123foo" as "123")

33414. **comment:** XXX: Are steps 6 and 7 in E5 Section 15.11.4.4 duplicated by * accident or are they actually needed? The first ToString() * could conceivably return 'undefined'.

label: code-design

33415. **comment:** Size sanity check. Should not be necessary because caller is * required to check this, but we don't want to cause a segfault * if the size wraps either in duk_size_t computation or when * storing the size in a 16-bit field.

label: code-design

33416. steps 3.h and 3.i

33417. Note: toJSON() is a generic function which works even if 'this' * is not a Date. The sole argument is ignored.

33418. module (argument)

33419. valstack space that suffices for all local calls, including recursion * of other than Duktape calls (getters etc)

33420. current paren level allows 'in' token

33421. **comment:** XXX: fast-int-to-double

label: code-design

33422. * ArrayBuffer.isView()

33423. Magically bound variable cannot be an accessor. However, * there may be an accessor property (or a plain property) in * place with magic behavior removed. This happens e.g. when * a magic property is redefined with defineProperty(). * Cannot assert for "not accessor" here.

33424. * We could clear the book-keeping variables for the topmost activation, * but don't do so now.

33425. * Debugging enabled

33426. * E5 Section 7.4, allow SourceCharacter (which is any 16-bit * code point).

33427. **comment:** XXX: duk_set_length

label: code-design

33428. **comment:** XXX: which lists should participate? to be finalized?

label: code-design

33429. XXXXXX-- -----

33430. * Bitstream decoder.

33431. * The handling here is a bit tricky. If a previous ']' has been processed, * we have a pending split1 and a pending jump (for a previous match). These * need to be back-patched carefully. See docs for a detailed example.

33432. Step 16: update length; note that the final length may be above 32 bit range * (but we checked above that this isn't the case here)

33433. **comment:** XXX: 'internal error' is a bit of a misnomer

label: code-design

33434. **comment:** In compatible mode and standard JSON mode, output * something useful for non-BMP characters. This won't * roundtrip but will still be more or less readable and * more useful than an error.

label: code-design

33435. A debugger forced interrupt check is not needed here, as * problematic safe calls are not caused by side effects.

33436. Real world behavior for map(): trailing non-existent * elements don't invoke the user callback, but are still * counted towards result 'length'.

33437. Ensure argument name is not a reserved word in current * (final) strictness. Formal argument parsing may not * catch reserved names if strictness changes during * parsing. ** We only need to do this in strict mode because non-strict * keyword are always detected in formal argument parsing.

33438. retval (sub-atom char length) unused, tainted as complex above

33439. no pre-checks now, assume a previous yield() has left things in * tip-top shape (longjmp handler will assert for these).

33440. * Peephole optimizer for finished bytecode. ** Does not remove opcodes; currently only straightens out unconditional * jump chains which are generated by several control structures.

33441. should not happen

33442. Avoid doing an actual write callback with length == 0, * because that's reserved for a write flush.

33443. DUK_TOK_REGEXP

33444. * Peephole optimize JUMP chains.

33445. * Execution timeout check

33446. Recursive value reviver, implements the Walk() algorithm. No C recursion * check is done here because the initial parsing step will already ensure * there is a reasonable limit on C recursion depth and hence object depth.

33447. type to represent a reg/const reference during compilation

33448. TypedArray views need an automatic ArrayBuffer which must be * provided as .buffer property of the view. Just create a new * ArrayBuffer sharing the same underlying buffer. ** The ArrayBuffer offset is always set to zero, so that if one * accesses the ArrayBuffer at the view's .byteOffset, the value * matches the view at index 0.

33449. **comment:** Indent helper. Calling code relies on js_ctx->recursion_depth also being * directly related to indent depth.

label: code-design

33450. DUK_TOK_PUBLIC

33451. E5 Sections 11.8.4, 11.8.5; x >= y --> not (x < y)

33452. 'get'

33453. **comment:** function name (borrowed reference), ends up in _name

label: code-design

33454. repl_value

33455. quotes

33456. 'undefined' already on stack top

33457. 'res' is used for "left", and 'tmp' for "right"

33458. def_value

33459. Delete elements required by a smaller length, taking into account * potentially non-configurable elements. Returns non-zero if all * elements could be deleted, and zero if all or some elements could * not be deleted. Also writes final "target length" to 'out_result_len'. * This is the length value that should go into the 'length' property * (must be set by the caller). Never throws an error.

33460. * duk_handle_call_protected() and duk_handle_call_unprotected(): * call into a Duktape/C or an Ecmascript function from any state. ** Input stack (thr): * * [func this arg1 ... argN] * * Output stack (thr): * * [retval] (DUK_EXEC_SUCCESS) * [errobj] (DUK_EXEC_ERROR (normal error), protected call) * * Even when executing a protected call an error may be thrown in rare cases * such as an insane num_stack_args argument. If there is no catchpoint for * such errors, the

fatal error handler is called. ** The error handling path should be error free, even for out-of-memory * errors, to ensure safe sandboxing. (As of Duktape 1.4.0 this is not * yet the case, see XXX notes below.)

33461. **comment:** XXX: skip null filename?

label: code-design

33462. avoid multiply

33463. New require() function for module, updated resolution base

33464. **comment:** * Function pointers ** Printing function pointers is non-portable, so we do that by hex printing * bytes from memory.

label: code-design

33465. "-Infinity", '-' has been eaten

33466. make absolute

33467. 'act' already set above

33468. DUK_BUFOBJ_UINT8CLAMPEDARRAY

33469. **comment:** XXX: turkish / azeri, lowercase rules

label: code-design

33470. **comment:** Coerce like a string. This makes sense because addition also treats * buffers like strings.

label: code-design

33471. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).

33472. no comma

33473. strict outer context

33474. jump for 'finally' case or end (if no finally)

33475. * Structs

33476. DUK__L0 -> '\ char * DUK__L1 ... DUK__L5 -> more lookup

33477. from next pc

33478. If the value happens to be 0xFFFFFFFF, it's not a valid array index * but will then match DUK__NO_ARRAY_INDEX.

33479. use 'left' as a temp

33480. JSON

33481. binary arithmetic; DUK_OP_ADD, DUK_OP_EQ, other binary ops

33482. Innermost fast path processes 4 valid base-64 characters at a time * but bails out on whitespace, padding chars ('=') and invalid chars. * Once the slow path segment has been processed, we return to the * inner fast path again. This handles e.g. base64 with newlines * reasonably well because the majority of a line is in the fast path.

33483. Key and value indices are either (-2, -1) or (-1, -2). Given idx_key, * idx_val is always (idx_key ^ 0x01).

33484. * Pointer built-ins

33485. Zero length string is not accepted without quotes

33486. [key] -> []

33487. No activation, no variable access. Could also pretend * we're in the global program context and read stuff off * the global object.

33488. current function being compiled (embedded instead of pointer for more compact access)

33489. Note: src_data may be NULL if input is a zero-size * dynamic buffer.

33490. Note: 'this' may be bound to any value, not just an object

33491. **comment:** XXX: Not needed for now, so not implemented. Note that * function pointers may have different size/layout than * a void pointer.

label: requirement

33492. out of spec, don't care

33493. **comment:** XXX: idx_val would fit into 16 bits, but using duk_small_uint_t * might not generate better code due to casting.

label: code-design

33494. used by e.g. duk_regexp_executor.c, string built-ins

33495. **comment:** XXX: could also copy from template, but there's no way to have any * other value here now (used code has no access to the template).

label: code-design

33496. Share yield longjmp handler.

33497. curr_token slot2 (matches 'lex' slot2_idx)

33498. Error message doesn't matter: the error is ignored anyway.

33499. Have a calling activation, check for direct eval (otherwise * assume indirect eval.

33500. **comment:** Compact but lots of churn.

label: code-design

33501. Output shuffling: only one output register is realistically possible. ** (Zero would normally be an OK marker value: if the target register * was zero, it would never be shuffled. But with DUK_USE_SHUFFLE_TORTURE * this is no longer true, so use -1 as a marker instead.)

33502. **comment:** actually used

label: code-design

33503. rule match

33504. * Helper structs

33505. terminating conditions

33506. see below

33507. allow automatic semicolon even without lineterm (compatibility)

33508. Outside any activation -> put to global.

33509. +0 = break, +1 = continue

33510. try locale specific formatter; if it refuses to format the * string, fall back to an ISO 8601 formatted value in local * time.

33511. -> [val arr]

33512. * Byte matching for back-references would be OK in case- * sensitive matching. In case-insensitive matching we need * to canonicalize characters, so back-reference matching needs * to be done with codepoints instead. So, we just decode * everything normally here, too. ** Note: back-reference index which is 0 or higher than * NCapturingParens (= number of capturing parens in the * -entire- regexp) is a compile time error. However, a * backreference referring to a valid capture which has * not matched anything always succeeds! See E5 Section * 15.10.2.9, step 5, sub-step 3.

33513. this is needed for regexp mode

33514. jump to next case clause

33515. pop plain buffer, now reachable through h_bufres

33516. * PUTPROP: Ecmascript property write. ** Unlike Ecmascript primitive which returns nothing, returns 1 to indicate * success and 0 to indicate failure (assuming throw is not set). ** This is an extremely tricky function. Some examples: * * * Currently a decref may trigger a GC, which may compact an object's * property allocation. Consequently, any entry indices (e_idx) will * be potentially invalidated by a decref. * * * Exotic behaviors (strings, arrays, arguments object) require, * among other things: * * - Preprocessing before and postprocessing after an actual property * write. For example, array index write requires pre-checking the * array 'length' property for access control, and may require an * array 'length' update after the actual write has succeeded (but * not if it fails). * * - Deletion of multiple entries, as a result of array 'length' write. * * * Input values are taken as pointers which may point to the valstack. * If valstack is resized because of the put (this may happen at least * when the array part is abandoned), the pointers can be invalidated. * (We currently make a copy of all of the input values to avoid issues.)

33517. add AD*2^16

33518. Proxy handler

33519. **comment:** * Tailcall handling ** Although the callstack entry is reused, we need to explicitly unwind * the current activation (or simulate an unwind). In particular, the * current activation must be closed, otherwise something like * test-bug-reduce-judofy.js results. Also catchstack needs be unwound * because there may be non-error-catching label entries in valid tail calls.

label: code-design

33520. **comment:** Should not be required because the code below always sets both high * and low parts, but at least gcc-4.4.5 fails to deduce this correctly * (perhaps because the low part is set (seemingly) conditionally in a * loop), so this is here to avoid the bogus warning.
label: code-design

33521. Impose a maximum string length for now. Restricted artificially to * ensure adding a heap header length won't overflow size_t. The limit * should be synchronized with DUK_HBUFFER_MAX_BYTELEN. ** E5.1 makes provisions to support strings longer than 4G characters. * This limit should be eliminated on 64-bit platforms (and increased * closer to maximum support on 32-bit platforms).

33522. relookup exports from module.exports in case it was changed by modSearch

33523. * Indexed read/write helpers (also used from outside this file)

33524. adv = 2 - 1 default OK

33525. **comment:** XXX: the key in 'key in obj' is string coerced before we're called * (which is the required behavior in E5/E5.1/E6) so the key is a string * here already.
label: code-design

33526. **comment:** XXX: side effect handling is quite awkward here
label: code-design

33527. act->func

33528. **comment:** Set stack top within currently allocated range, but don't reallocate. * This is performance critical especially for call handling, so whenever * changing, profile and look at generated code.
label: code-design

33529. **comment:** Neutered checks not necessary here: neutered buffers have * zero 'length' so we'll effectively skip them.
label: code-design

33530. Call setup checks callability.

33531. **comment:** Flags for intermediate value coercions. A flag for using a forced reg * is not needed, the forced_reg argument suffices and generates better * code (it is checked as it is used).
label: code-design

33532. **comment:** Non-ASCII slow path (range-by-range linear comparison), very slow
label: code-design

33533. Contract, either: * - Push value on stack and return 1 * - Don't push anything on stack and return 0

33534. 0: not IdentifierStart or IdentifierPart * 1: IdentifierStart and IdentifierPart * -1: IdentifierPart only

33535. DUK_BUFOBJ_UINT32ARRAY

33536. * Restore 'caller' property for non-strict callee functions.

33537. ($\geq e 0$) AND (not ($= f (\text{expt } b (- p 1))$)) * * be < ($\text{expt } b e$) == $b^e * r \leftarrow (* f \text{ be } 2) == 2 * f * b^e$ [if $b==2 \rightarrow f * b^{(e+1)}$] * s <- 2 * m+ <- be == $b^e * m - <- be == b^e * k <- 0 * B <- B * \text{low_ok} <- \text{round} * \text{high_ok} <- \text{round}$

33538. A -> unused (reserved for flags, for consistency with DUK_OP_CALL) * B -> target register and start reg: constructor, arg1, ..., argN * (for DUK_OP_NEWI, 'b' is indirect) * C -> num args (N)

33539. 0x60...0x6f

33540. ≥ 0

33541. mimic semantics for strings

33542. Parse a single statement. ** Creates a label site (with an empty label) automatically for iteration * statements. Also "peels off" any label statements for explicit labels.

33543. DUK_FLD_8BIT

33544. [key]

33545. normal key/value

33546. mm <- b^e

33547. steps 4.a and 4.b are tricky

33548. The external string struct is defined even when the feature is inactive.

33549. fixed arg count

33550. A -> flags * B -> return value reg/const * C -> currently unused

33551. **comment:** Because there are quite many call sites, pack error code (require at most * 8-bit) into a single argument.
label: code-design

33552. Raw helper to extract internal information / statistics about a value. * The return values are version specific and must not expose anything * that would lead to security issues (e.g. exposing compiled function * 'data' buffer might be an issue). Currently only counts and sizes and * such are given so there should not be a security impact.

33553. -> [...] copy template newobj]

33554. XXX: This causes recursion up to inner function depth * which is normally not an issue, e.g. mark-and-sweep uses * a recursion limiter to avoid C stack issues. Avoiding * this would mean some sort of a work list or just refusing * to serialize deep functions.

33555. * Size-optimized pc->line mapping.

33556. return as is

33557. value1 -> resume value, value2 -> resume thread, iserror -> error/normal

33558. function: create binding for func name (function templates only, used for named function expressions)

33559. The low 8 bits map directly to duk_hobject.h DUK_PROPDESC_FLAG_xxx. * The remaining flags are specific to the debugger.

33560. **comment:** XXX: copy from caller?
label: code-design

33561. always for register bindings

33562. **comment:** XXX: optimize: allocate an array part to the necessary size (upwards * estimate) and fill in the values directly into the array part; finally * update 'length'.
label: code-design

33563. heap->dbg_udata: keep

33564. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

33565. IdentifierStart production with Letter, ASCII, and non-BMP excluded

33566. **comment:** * Custom formatter for debug printing, allowing Duktape specific data * structures (such as tagged values and heap objects) to be printed with * a nice format string. Because debug printing should not affect execution * state, formatting here must be independent of execution (see implications * below) and must not allocate memory. ** Custom format tags begin with a '%!' to safely distinguish them from * standard format tags. The following conversions are supported: * * %!T tagged value (duk_tval *) * %!O heap object (duk_heapobj *) * %!I decoded bytecode instruction * %!C bytecode instruction opcode name (arg is long) * * Everything is serialized in a JSON-like manner. The default depth is one * level, internal prototype is not followed, and internal properties are not * serialized. The following modifiers change this behavior: * * @ print pointers * # print binary representations (where applicable) * d deep traversal of own properties (not prototype) * p follow prototype chain (useless without 'd') * i include internal properties (other than prototype) * x hexdump buffers * h heavy formatting * * For instance, the following serializes objects recursively, but does not * follow the prototype chain nor print internal properties: "%!do". ** Notes: * * Standard sprintf return value semantics seem to vary. This * implementation returns the number of bytes it actually wrote * (excluding the null terminator). If retval == buffer size, * output was truncated (except for corner cases). ** Output format is intentionally different from Ecmascript * formatting requirements, as formatting here serves debugging * of internals. ** Depth checking (and updating) is done in each type printer * separately, to allow them to call each other freely. ** Some pathological structures might take ages to print (e.g. * self recursion with 100 properties pointing to the object * itself). To guard against these, each printer also checks * whether the output buffer is full; if so, early exit. ** Reference loops are detected using a loop stack.
label: code-design

33567. dummy value used as marker

33568. never here, but fall through

33569. **comment:** * duk_handle_safe_call(): make a "C protected call" within the * current activation. ** The allowed thread states for making a call are the same as for * duk_handle_call_xxx(). * * Error handling is similar to duk_handle_call_xxx(); errors may be thrown * (and result in a fatal error) for insane arguments.
label: code-design

33570. must never longjmp
33571. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside object 'a_size'.
33572. Must set 'length' explicitly when using duk_xdef_prop_xxx() to * set the values.
33573. heapdr size and additional allocation size, followed by * type specific stuff (with varying value count)
33574. stack top contains 'true'
33575. callbacks and udata; dbg_read_cb != NULL is used to indicate attached state
33576. '\xffHandler'
33577. Note: string is a terminal heap object, so no depth check here
33578. 0x80-0x8f
33579. * Internal helper for defining an accessor property, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes. This is called * very rarely, so the implementation first sets a value to undefined * and then changes the entry to an accessor (this is to save code space).
33580. Debug print which visualizes the qsort partitioning process.
33581. XXX: this placeholder is not always correct, but use for now. * It will fail in corner cases; see test-dev-func-cons-args.js.
33582. r <- (2 * f) * b^e
33583. unreferenced with some options
33584. always terminates led()
33585. (reference is valid as long activation exists)
33586. don't report __FILE__ / __LINE__ as fileName/lineNumber
33587. **comment:** Note: don't bail out early, we must read all the ranges from * bytecode. Another option is to skip them efficiently after * breaking out of here. Prefer smallest code.
label: code-design
33588. Pause for all step types: step into, step over, step out. * This is the only place explicitly handling a step out.
33589. awkward; we assume there is space for this
33590. Peek ahead in the stream one byte.
33591. * Breakpoint management
33592. * Compact an object
33593. slot B is both a target and a source (used by extraops like DUK_EXTRAOP_INSTOF)
33594. alias for above
33595. handle comma and closing brace
33596. shrink case; leave some spare
33597. strictness is not inherited, intentional
33598. only Array.prototype matches
33599. Parse a function-like expression, assuming that 'comp_ctx->curr_func' is * correctly set up. Assumes that curr_token is just after 'function' (or * 'set'/'get' etc).
33600. function: always register bound
33601. **comment:** XXX: fast primitive to set a bunch of values to UNDEFINED
label: code-design
33602. Buffer size needed for duk_bi_date_format_timeval(). * Accurate value is 32 + 1 for NUL termination: * >>> len('+123456-01-23T12:34:56.123+12:34') * 32 *
Include additional space to be safe.
33603. We want to compare the slice/view areas of the arguments. * If either slice/view is invalid (underlying buffer is shorter) * ensure equals() is false, but otherwise the only thing that * matters is to be memory safe.
33604. This seems reasonable overall.
33605. Loop prevention
33606. 'throw'
33607. we're running inside the caller's activation, so no change in call/catch stack or valstack bottom
33608. Assume that the native representation never contains a closing * parenthesis.
33609. varargs marker
33610. entry level reached
33611. Current strictness flag: affects API calls.
33612. Because value stack init policy is 'undefined above top', * we don't need to write, just assert.
33613. '*'
33614. call is from executor, so we know we have a jmpbuf
33615. Yield/resume book-keeping.
33616. finally part will catch
33617. Quite lenient, e.g. allow empty as zero, but don't allow trailing * garbage.
33618. IEEE requires that NaNs compare false
33619. Load inner functions to value stack, but don't yet copy to buffer.
33620. [start end]
33621. **comment:** Try to make do with a stack buffer to avoid allocating a temporary buffer. * This works 99% of the time which is quite nice.
label: code-design
33622. This testcase fails when Emscripten-generated code runs on Firefox. * It's not an issue because the failure should only affect packed * duk_tval representation, which is not used with Emscripten.
33623. **comment:** XXX: keys is an internal object with all keys to be processed * in its (gapless) array part. Because nobody can touch the keys * object, we could iterate its array part directly (keeping in mind * that it can be reallocated).
label: code-design
33624. Return value is like write(), number of bytes written. * The return value matters because of code like: * "off += buf.copy(...)".
33625. Exponent without digits (e.g. "1e" or "1e+"). If trailing garbage is * allowed, ignore exponent part as garbage (= parse as "1", i.e. exp 0).
33626. DUK_TOK_THROW
33627. **comment:** Note: not a typo, "object" is returned for a null value
label: documentation
33628. atom_char_length, atom_start_offset, atom_start_offset reflect the * atom matched on the previous loop. If a quantifier is encountered * on this loop, these are needed to handle the quantifier correctly. * new_atom_char_length etc are for the atom parsed on this round; * they're written to atom_char_length etc at the end of the round.
33629. Original idiom used, minimal code size.
33630. **comment:** 'fun' is quite rarely used, so no local for it
label: code-design
33631. **comment:** XXX: fastint
label: code-design
33632. Note: in integer arithmetic, (x / 4) is same as floor(x / 4) for non-negative * values, but is incorrect for negative ones.
33633. normal
33634. Careful with wrapping (left shifting idx would be unsafe).
33635. **comment:** XXX: for negative input offsets, 'offset' will be a large * positive value so the result here is confusing.
label: code-design
33636. no overflow
33637. "+11:22:0"
33638. borrowed: function being executed; for bound function calls, this is the final, real function, NULL for lightfuncs
33639. **comment:** * Augment error (throw time), unless alloc/double error
label: code-design

33640. -> [... ToObject(this) ToUInt32(length) arg[i]]
33641. The specification has quite awkward order of coercion and * checks for toPrecision(). The operations below are a bit * reordered, within constraints of observable side effects.
33642. caller
33643. match followed by capture(s)
33644. return input buffer, converted to a Duktape.Buffer object * if called as a constructor (no change if called as a * function).
33645. We could use a switch-case for the class number but it turns out * a small if-else ladder on class masks is better. The if-ladder * should be in order of relevancy.
33646. * New length is smaller than old length, need to delete properties above * the new length. * * If array part exists, this is straightforward: array entries cannot * be non-configurable so this is guaranteed to work. * * If array part does not exist, array-indexed values are scattered * in the entry part, and some may not be configurable (preventing length * from becoming lower than their index + 1). To handle the algorithm * in E5 Section 15.4.5.1, step 1 correctly, we scan the entire property * set twice.
33647. 'ptr' is evaluated both as LHS and RHS.
33648. * Ecmascript [[Class]]
33649. **comment:** Quick reject of too large or too small exponents. This check * would be incorrect for zero (e.g. "0e1000" is zero, not Infinity) * so zero check must be above.
label: code-design
33650. finalize_list will always be processed completely
33651. **comment:** XXX: exposed duk_debug_read_pointer
label: code-design
33652. -> [... key val replacer holder]
33653. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside bufferobject length.
33654. open scope information, for compiled functions only
33655. Handle empty separator case: it will always match, and always * triggers the check in step 13.c.iii initially. Note that we * must skip to either end of string or start of first codepoint, * skipping over any continuation bytes! * * Don't allow an empty string to match at the end of the input.
33656. temp reg handling
33657. * Parse function body
33658. x <- y - z
33659. 'Pointer'
33660. relookup if changed
33661. **comment:** * Augment an error at creation time with _Tracedata/fileName/lineNumber * and allow a user error handler (if defined) to process/replace the error. * The error to be augmented is at the stack top. * * thr: thread containing the error value * thr_callstack: thread which should be used for generating callstack etc. * c_filename: C __FILE__ related to the error * c_line: C __LINE__ related to the error * noblame_fileline: if true, don't fileName/line as error source, otherwise use traceback * (needed because user code filename/line are reported but internal ones * are not) * * XXX: rename noblame_fileline to flags field; combine it to some existing * field (there are only a few call sites so this may not be worth it).
label: code-design
33662. throw_flag
33663. * Else, must be one of: * - ExpressionStatement, possibly a directive (String) * - LabelledStatement (Identifier followed by ':') * * Expressions beginning with 'function' keyword are covered by a case * above (such expressions are not allowed in standard E5 anyway). * Also expressions starting with '{' are interpreted as block * statements. See E5 Section 12.4. * * Directive detection is tricky; see E5 Section 14.1 on directive * prologue. A directive is an expression statement with a single * string literal and an explicit or automatic semicolon. Escape * characters are significant and no parens etc are allowed: * * 'use strict'; // valid 'use strict' directive * 'use\u0020strict'; // valid directive, not a 'use strict' directive * ('use strict'); // not a valid directive * * The expression is determined to consist of a single string literal * based on duk__expr_nud() and duk__expr_led() call counts. The string literal * of a 'use strict' directive is determined to lack any escapes based * num_escapes count from the lexer. Note that other directives may be * allowed to contain escapes, so a directive with escapes does not * terminate a directive prologue. * * We rely on the fact that the expression parser will not emit any * code for a single token expression. However, it will generate an * intermediate value which we will then successfully ignore. * * A similar approach is used for labels.
33664. The act->prev_caller should only be set if the entry for 'caller' * exists (as it is only set in that case, and the property is not * configurable), but handle all the cases anyway.
33665. break/continue without label
33666. **comment:** XXX: the best typing needs to be validated by perf measurement: * e.g. using a small type which is the cast to a larger duk_idx_t * may be slower than declaring the variable as a duk_idx_t in the * first place.
label: code-design
33667. **comment:** XXX: conversion errors should not propagate outwards. * Perhaps values need to be coerced individually?
label: code-design
33668. XXX: here we need to know if 'left' is left-hand-side compatible. * That information is no longer available from current expr parsing * state; it would need to be carried into the 'left' ivalue or by * some other means.
33669. non-global regexp: lastIndex never updated on match
33670. Note: undefined from Section 11.8.5 always results in false * return (see e.g. Section 11.8.3) - hence special treatment here.
33671. If there's no catch block, rc_varname will be 0 and duk__patch_trycatch() * will replace the LDCONST with a NOP. For any actual constant (including * constant 0) the DUK__CONST_MARKER flag will be set in rc_varname.
33672. exposed because lexer needs these too
33673. empty match (may happen with empty separator) -> bump and continue
33674. As with all inspection code, we rely on the debug client providing * a valid, non-stale pointer: there's no portable way to safely * validate the pointer here.
33675. Set up curr_pc for opcode dispatch.
33676. [... pattern flags escaped_source]
33677. **comment:** The 'magic' field allows an opaque 16-bit field to be accessed by the * Duktape/C function. This allows, for instance, the same native function * to be used for a set of very similar functions, with the 'magic' field * providing the necessary non-argument flags / values to guide the behavior * of the native function. The value is signed on purpose: it is easier to * convert a signed value to unsigned (simply AND with 0xffff) than vice * versa. * * Note: cannot place nargs/magic into the heapdr flags, because * duk_hobject takes almost all flags already (and needs the spare).
label: code-design
33678. Get minimum entry part growth for a certain size.
33679. valstack index of start of args (arg1) (relative to entry valstack_bottom)
33680. [ToObject(this) item1 ... itemN arr item(i) item(i)[j]]
33681. thr->heap->lj.value2 is 'thread', will be wiped out at the end
33682. break/continue with label (label cannot be a reserved word, production is 'Identifier'
33683. property access
33684. Coerce all finite parts with ToInteger(). ToInteger() must not * be called for NaN/Infinity because it will convert e.g. NaN to * zero. If ToInteger() has already been called, this has no side * effects and is idempotent. * * Don't read dparts[DUK_DATE_IDX_WEEKDAY]; it will cause Valgrind * issues if the value is uninitialized.
33685. internal property functions
33686. Report thrown value to client coerced to string
33687. **comment:** ArrayBuffer argument is handled specially above; the rest of the * argument variants are handled by shared code below.
label: code-design
33688. DUK_USE_USER_INITJS
33689. Range limited to [0, 0x7fffffff] range, i.e. range that can be * represented with duk_int32_t. Use this when the method doesn't * handle the full 32-bit unsigned range correctly.
33690. 110x xxxx 10xx xxxx
33691. Sanity limits for stack sizes.

33692. **comment:** Linear scan: more likely because most objects are small. * This is an important fast path. ** XXX: this might be worth inlining for property lookups.
label: code-design

33693. XXX: will need a force flag if garbage collection is triggered * explicitly during paused state.

33694. Array length is larger than 'asize'. This shouldn't * happen in practice. Bail out just in case.

33695. No net refcount change.

33696. * Assert context is valid: non-NULL pointer, fields look sane. ** This is used by public API call entrypoints to catch invalid 'ctx' pointers * as early as possible; invalid 'ctx' pointers cause very odd and difficult to * diagnose behavior so it's worth checking even when the check is not 100%.

33697. * Setup value stack: clamp to 'nargs', fill up to 'nregs'

33698. This should not happen because DUK_TAG_OBJECT case checks * for this already, but check just in case.

33699. result array is already at the top of stack

33700. args go here as a comma expression in parens

33701. [...] source? filename? &comp_args] (depends on flags)

33702. no extra padding

33703. Start position (inclusive) and end position (exclusive)

33704. tentative, checked later

33705. slot A is a source (default: target)

33706. indexed by recursion_depth

33707. * Fixed buffer helper useful for debugging, requires no allocation * which is critical for debugging.

33708. Order of unwinding is important

33709. * Final sigma context specific rule. This is a rather tricky * rule and this handling is probably not 100% correct now. * The rule is not locale/language specific so it is supported.

33710. tracks maximum initialized index + 1

33711. **comment:** first register that is a temporary (below: variables)
label: code-design

33712. A -> register of target object * B -> first register of value data (start_index, value1, value2, ..., valueN) * C -> number of key/value pairs (N)

33713. Just skip, leaving zeroes in the result.

33714. [...] arr]

33715. Note: successive characters could be joined into string matches * but this is not trivial (consider e.g. '/xyz+'); see docs for * more discussion.

33716. Coerce an duk_alue to a register or constant; result register may * be a temp or a bound register. ** The duk_alue argument ('x') is converted into a regconst as a * side effect.

33717. bytes in source

33718. no voluntary gc

33719. A -> result reg * B -> object reg * C -> key reg/const

33720. An object may have FINALIZED here if it was finalized by mark-and-sweep * on a previous run and refcount then decreased to zero. We won't run the * finalizer again here.

33721. **comment:** XXX: allow object to be a const, e.g. in 'foo'.toString()? * On the other hand, DUK_REGCONSTP() is slower and generates * more code.
label: code-design

33722. refcount

33723. regexp res_obj is at index 4

33724. stack[0] = callback fn * stack[1] = initialValue * stack[2] = object (coerced this) * stack[3] = length (not needed, but not popped above) * stack[4] = accumulator

33725. **comment:** Node.js Buffer variable width integer field. We don't really * care about speed here, so aim for shortest algorithm.
label: code-design

33726. The spec algorithm first does "R = ToString(separator)" before checking * whether separator is undefined. Since this is side effect free, we can * skip the ToString() here.

33727.]'

33728. NULL with zero length represents an empty string; NULL with higher * length is also now treated like an empty string although it is * a bit dubious. This is unlike duk_push_string() which pushes a * 'null' if the input string is a NULL.

33729. -> [func funcname env funcname func]

33730. true, because v[1] has at least one bit set

33731. bp to use when parsing a top level Expression

33732. Caller will finish the marking process if we hit a recursion limit.

33733. y

33734. **comment:** * E5 Section 7.6: ** IdentifierStart: * UnicodeLetter * \$ * _ * \ UnicodeEscapeSequence ** IdentifierStart production has one multi-character production: ** \ UnicodeEscapeSequence ** The '\ character is -not- matched by this function. Rather, the caller * should decode the escape and then call this function to check whether the * decoded character is acceptable (see discussion in E5 Section 7.6). ** The "UnicodeLetter" alternative of the production allows letters * from various Unicode categories. These can be extracted with the * "src/extract_chars.py" script. ** Because the result has hundreds of Unicode codepoint ranges, matching * for any values >= 0x80 are done using a very slow range-by-range scan * and a packed range format. ** The ASCII portion (codepoints 0x00 ... 0x7f) is fast-pathed below because * it matters the most. The ASCII related ranges of IdentifierStart are: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z'] * 0x0024 ['\$'] * 0x005f['_']
label: code-design

33735. Must be an object, otherwise TypeError (E5.1 Section 8.10.5, step 1).

33736. TypeError if fails

33737. * Entries part

33738. * Delayed env creation check

33739. insert 'undefined' values at idx_rcbase to get the * return values to idx_rebase

33740. xxx -> DUK_HBUFFEROBJECT_ELEM_INT8

33741. guarantees entry_callstack_top - 1 >= 0

33742. no code needs to be emitted, the regs already have values

33743. magic: 0=getter call, 1=Object.getPrototypeOf

33744. PC is unsigned. If caller does PC arithmetic and gets a negative result, * it will map to a large PC which is out of bounds and causes a zero to be * returned.

33745. * Debug connection write primitives

33746. **comment:** XXX: join these ops (multiply-accumulate), but only if * code footprint decreases.
label: code-design

33747. module.name for .name, default to last component if * not present.

33748. * Exposed debug macros: debugging enabled

33749. Object.defineProperty() calls [[DefineOwnProperty]] with Throw=true

33750. 0xe0-0xef

33751. compact the prototype

33752. XXX: default priority for infix operators is duk_expr_lbp(tok) -> get it here?

33753. read header

33754. prop index in 'hash part', < 0 if not there

33755. This is based on Rhino EquivalentYear() algorithm: *
<https://github.com/mozilla/rhino/blob/f99cc11d616f0cdda2c42bde72b3484df6182947/src/org.mozilla/javascript/NativeDate.java>

33756. **comment:** Clamping to zero makes the API more robust to calling code * calculation errors.
label: code-design

33757. Surround with parentheses like in JX, ensures NULL pointer * is distinguishable from null value ("(null)" vs "null").

33758. Attempt to write 'stack', 'fileName', 'lineNumber' works as if * user code called Object.defineProperty() to create an overriding * own property. This allows user code to overwrite .fileName etc * intuitively as e.g. "err.fileName = 'dummy'" as one might expect. * See <https://github.com/svaarala/duktape/issues/387>.

33759. object is a native function (duk_hnativefunction)
33760. [... error func fileName lineNumber]
33761. 21 bits
33762. **comment:** XXX: thread selection for mark-and-sweep is currently a hack. * If we don't have a thread, the entire mark-and-sweep is now * skipped (although we could just skip finalizations).
label: code-design
33763. embed: duk_hstring ptr
33764. enumeration
33765. * Free memory
33766. Maximum value check ensures 'nbytes' won't wrap below.
33767. Use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to an existing one).
33768. currently, even for Array.prototype
33769. unsigned intentionally
33770. DUK_TOK_SNEQ
33771. only outer_lex_env matters, as functions always get a new * variable declaration environment.
33772. No need to assert, buffer size maximum is 0xffff.
33773. default is explicit index read/write copy
33774. not enumerable
33775. * Exposed data
33776. avoid problems if p == h->prototype
33777. DUK_USE_PROVIDE_DEFAULT_ALLOC_FUNCTIONS
33778. Output shuffle needed after main operation
33779. For native calls must be NULL so we don't sync back
33780. **comment:** XXX: very basic optimization -> duk_get_prop_stridx_top
label: code-design
33781. omitted
33782. toGMTString'
33783. * Object related * * Note: seal() and freeze() are accessible through Ecmascript bindings, * and are not exposed through the API.
33784. * ISO 8601 subset parser.
33785. Internal keys are prefixed with 0xFF in the stringtable * (which makes them invalid UTF-8 on purpose).
33786. without explicit non-BMP support, assume non-BMP characters * are always accepted as identifier characters.
33787. [res]
33788. dash is already 0
33789. Object flag. There are currently 26 flag bits available. Make sure * this stays in sync with debugger object inspection code.
33790. This may only happen if built-ins are being "torn down". * This behavior is out of specification scope.
33791. Not possible because array object 'length' is present * from its creation and cannot be deleted, and is thus * caught as an existing property above.
33792. Note: strictness is not inherited from the current Duktape/C * context. Otherwise it would not be possible to compile * non-strict code inside a Duktape/C activation (which is * always strict now). See tests/api/test-eval-strictness.c * for discussion.
33793. A -> flags * B -> base register for call (base -> func, base+1 -> this, base+2 -> arg1 ... base+2+N-1 -> argN) * (for DUK_OP_CALLI, 'b' is indirect) * C -> nargs
33794. * Wrapper for jmp_buf. * * This is used because jmp_buf is an array type for backward compatibility. * Wrapping jmp_buf in a struct makes pointer references, sizeof, etc, * behave more intuitively. * * http://en.wikipedia.org/wiki/Setjmp.h#Member_types
33795. * This is a bit tricky to implement portably. The result depends * on the timestamp (specifically, DST depends on the timestamp). * If e.g. UNIX APIs are used, they'll have portability issues with * very small and very large years. * * Current approach: * * - Stay within portable UNIX limits by using equivalent year mapping. * Avoid year 1970 and 2038 as some conversions start to fail, at * least on some platforms. Avoiding 1970 means that there are * currently DST discrepancies for 1970. * * - Create a UTC and local time breakdowns from 't'. Then create * a time_t using gmtime() and localtime() and compute the time * difference between the two. * * Equivalent year mapping (E5 Section 15.9.1.8): * * If the host environment provides functionality for determining * daylight saving time, the implementation of ECMAScript is free * to map the year in question to an equivalent year (same * leap-year-ness and same starting week day for the year) for which * the host environment provides daylight saving time information. * The only restriction is that all equivalent years should produce * the same result. * * This approach is quite reasonable but not entirely correct, e.g. * the specification also states (E5 Section 15.9.1.8): * * The implementation of ECMAScript should not try to determine * whether the exact time was subject to daylight saving time, but * just whether daylight saving time would have been in effect if * the _current daylight saving time algorithm_ had been used at the * time. This avoids complications such as taking into account the * years that the locale observed daylight saving time year round. * * Since we rely on the platform APIs for conversions between local * time and UTC, we can't guarantee the above. Rather, if the platform * has historical DST rules they will be applied. This seems to be the * general preferred direction in Ecmascript standardization (or at least * implementations) anyway, and even the equivalent year mapping should * be disabled if the platform is known to handle DST properly for the * full Ecmascript range. * * The following has useful discussion and links: * * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
33796. keep val_highest
33797. **comment:** XXX: limit to quoted strings only, to save keys from being cluttered?
label: code-design
33798. DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT
33799. can't get value, may be accessor
33800. Relookup and initialize dispatch loop variables. Debugger check.
33801. * Longjmp state, contains the information needed to perform a longjmp. * Longjmp related values are written to value1, value2, and iserror.
33802. leave result on stack top
33803. Resolved, normalized absolute module ID
33804. marker; not actual tagged value
33805. re-lookup curr char on first round
33806. [... formals arguments map mappedNames]
33807. borrowed: full duk_rval for function being executed; for lightfuncs
33808. Shared handler to minimize parser size. Cause will be * hidden, unfortunately, but we'll have an offset which * is often quite enough.
33809. Skip dvalue.
33810. **comment:** XXX: inefficient; block remove primitive
label: code-design
33811. If duk_ivalue_toplain_raw() allocates a temp, forget it and * restore next temp state.
33812. DUK_HOBJECT_FLAG_CONSTRUCTABLE varies
33813. [... pattern flags]
33814. custom; implies DUK_HOBJECT_IS_THREAD
33815. [thisArg arg1 ... argN]
33816. for array and object literals
33817. validation of the regexp is caller's responsibility
33818. regnum is sane
33819. * To avoid creating a heavy intermediate value for the list of ranges, * only the start token ('[or '[^') is parsed here. The regexp * compiler parses the ranges itself.
33820. **comment:** XXX: Currently function source code is not stored, as it is not * required by the standard. Source code should not be stored by * default (user should enable it explicitly), and the source should * probably be compressed with a trivial text compressor; average * compression of 20-30% is quite easy to achieve even with a trivial * compressor (RLE + backwards lookup). * * Debugging needs source code to be useful: sometimes input code is * not found in files as it may be generated and then eval()'d, given * by dynamic C code, etc. * * Other issues: * * - Need tokenizer indices for start and end to substring * - Always normalize function declaration part? * - If we keep _Formals, only need to store body
label: code-design
33821. Preinc/predec for var-by-name, slow path.

33822. tm_isdst is both an input and an output to mktime(), use 0 to * avoid DST handling in mktime(): * - https://github.com/svaarala/duktape/issues/406 * - http://stackoverflow.com/questions/8558919/mktime-and-tm-isdst

33823. Small variant; roughly 150 bytes smaller than the fast variant.

33824. -> [... closure template env funcname]

33825. * Free the heap. * * Frees heap-related non-heap-tracked allocations such as the * string intern table; then frees the heap allocated objects; * and finally frees the heap structure itself. Reference counts * and GC markers are ignored (and not updated) in this process, * and finalizers won't be called. * * The heap pointer and heap object pointers must not be used * after this call.

33826. rescue no longer supported

33827. res->mark_and_sweep_trigger_counter == 0 -> now causes immediate GC; which is OK

33828. _Pc2line

33829. Sanity checks for string and token defines

33830. number of escapes and line continuations (for directive prologue)

33831. MakeTime

33832. **comment:** XXX: this is possible without resorting to the value stack
label: code-design

33833. ignored

33834. NULL for lightfunc

33835. lookup shorthands (note: assume context variable is named 'lex_ctxt')

33836. Other callbacks are optional.

33837. truncated in case pass 3 needed

33838. With constants and inner functions on value stack, we can now * atomically finish the function 'data' buffer, bump refcounts, * etc. * * Here we take advantage of the value stack being just a duk_tval * array: we can just memcpy() the constants as long as we incref * them afterwards.

33839. DUK_USE_NONSTD_FUNC_SOURCE_PROPERTY

33840. Leave 'getter' on stack

33841. Called as a function, pattern has [[Class]] "RegExp" and * flags is undefined -> return object as is.

33842. Q...f

33843. x < y

33844. set exotic behavior only after we're done

33845. **comment:** Default function to write a formatted log line. Writes to stderr, * appending a newline to the log line. * * The argument is a buffer whose visible size contains the log message. * This function should avoid coercing the buffer to a string to avoid * string table traffic.
label: code-design

33846. Current size is about 152 bytes.

33847. **comment:** These base values are never used, but if the compiler doesn't know * that DUK_ERROR() won't return, these are needed to silence warnings. * On the other hand, scan-build will warn about the values not being * used, so add a DUK_UNREF.
label: code-design

33848. hash size relative to entries size: for value X, approx. hash_prime(e_size + e_size / X)

33849. while repl

33850. Note: we don't parse back exponent notation for anything else * than radix 10, so this is not an ambiguous check (e.g. hex * exponent values may have 'e' either as a significand digit * or as an exponent separator). * * If the exponent separator occurs twice, 'e' will be interpreted * as a digit (= 14) and will be rejected as an invalid decimal * digit.

33851. DUK_HEAPHDR_HAS_FINALIZED may be set if we're doing a * refzero finalization and mark-and-sweep gets triggered * during the finalizer.

33852. Setup state. Initial ivalue is 'undefined'.

33853. 'object'

33854. If we processed any debug messages breakpoints may have * changed; restart execution to re-check active breakpoints.

33855. 1 1 1 <32 bits>

33856. **comment:** Leading / trailing whitespace is sometimes accepted and * sometimes not. After white space trimming, all valid input * characters are pure ASCII.
label: code-design

33857. patch in range count later

33858. * If tracebacks are disabled, 'fileName' and 'lineNumber' are added * as plain own properties. Since Error.prototype has accessors of * the same name, we need to define own properties directly (cannot * just use e.g. duk_put_prop_stridx). Existing properties are not * overwritten in case they already exist.

33859. 0: toString

33860. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small.
label: code-design

33861. UTF-8 encoded bytes escaped as %xx%xx%xx... -> 3 * nbytes. * Codepoint range is restricted so this is a slightly too large * but doesn't matter.

33862. lowest mantissa for this exponent

33863. * Traceback handling when tracebacks disabled. * * The fileName / lineNumber stubs are now necessary because built-in * data will include the accessor properties in Error.prototype. If those * are removed for builds without tracebacks, these can also be removed. * 'stack' should still be present and produce a ToString() equivalent: * this is useful for user code which prints a stacktrace and expects to * see something useful. A normal stacktrace also begins with a ToString() * of the error so this makes sense.

33864. all we need to know

33865. ASSIGNMENT EXPRESSION

33866. IdentityEscape

33867. Must decrease recursion depth before returning.

33868. note: mixing len into seed improves hashing when skipping

33869. For join(), nargs is 1. For toLocaleString(), nargs is 0 and * setting the top essentially pushes an undefined to the stack, * thus defaulting to a comma separator.

33870. Return with final function pushed on stack top.

33871. Note: 'e' and 'E' are also accepted here.

33872. %c', passed concretely as int

33873. thread resumed another thread (active but not running)

33874. or a non-catching entry

33875. next char

33876. -> [... key val]

33877. add F

33878. **comment:** return a specific NaN (although not strictly necessary)
label: code-design

33879. * Variant 1

33880. global regexp: lastIndex updated on match

33881. **comment:** pre-check how many atom copies we're willing to make (atom_copies not needed below)
label: requirement

33882. Current size (not counting a dynamic buffer's "spare").

33883. +2 = catcher value, catcher lj_type

33884. parsing in "directive prologue", recognize directives

33885. non-strict: non-deletable, writable

33886. DUK_USE_64BIT_OPS

33887. simulate alloc failure on every realloc (except when mark-and-sweep is running)

33888. 7

33889. **comment:** alloc temp just in case, to update max temp
label: code-design

33890. * An array must have a 'length' property (E5 Section 15.4.5.2). * The special array behavior flag must only be enabled once the * length property has been added. *
* The internal property must be a number (and preferably a * fastint if fastint support is enabled).

33891. * Create a hstring and insert into the heap. The created object * is directly garbage collectable with reference count zero. * * The caller must place the interned string into the stringtable * immediately (without chance of a longjmp); otherwise the string * is lost.

33892. Note: this is correct even for default clause statements: * they participate in 'fall-through' behavior even if the * default clause is in the middle.

33893. exposed, used by e.g. duk.bi_date.c

33894. * Parsing of ints and floats

33895. flags: "gi"

33896. For TypedArrays 'undefined' return value is specified * by ES6 (matches V8).

33897. * Return target object

33898. side effects

33899. [... result]

33900. only objects have finalizers

33901. interpret e.g. '0x' and '0xg' as a NaN (= parse error)

33902. **comment:** * Get and call the finalizer. All of this must be wrapped * in a protected call, because even getting the finalizer * may trigger an error (getter may throw one, for instance).
label: code-design

33903. **comment:** * String/JSON conversions * * Human readable conversions are now basically ISO 8601 with a space * (instead of 'T') as the date/time separator. This is a good baseline * and is platform independent. * * A shared native helper to provide many conversions. Magic value contains * a set of flags. The helper provides: * * toString() * toDateString() * toTimeString() * toLocaleString() * toLocaleDateString() * toLocaleTimeString() * toUTCString() * toISOString() * * Notes: * * - Date.prototype.toGMTString() and Date.prototype.toUTCString() are * required to be the same Ecmascript function object (!), so it is * omitted from here. * * - Date.prototype.toUTCString(): E5.1 specification does not require a * specific format, but result should be human readable. The * specification suggests using ISO 8601 format with a space (instead * of 'T') separator if a more human readable format is not available. * * - Date.prototype.toISOString(): unlike other conversion functions, * toISOString() requires a RangeError for invalid date values.
label: code-design

33904. * Thread support.

33905. bytecode start offset of the atom parsed in this loop * (allows quantifiers to copy the atom bytecode)

33906. temp copy, write back for next loop

33907. char loop

33908. basic platform types

33909. we currently assume virtual properties are not configurable (as none of them are)

33910. Stage 1: find highest preventing non-configurable entry (if any). * When forcing, ignore non-configurability.

33911. **comment:** XXX: keep property attributes or tweak them here? * Properties will now be non-configurable even when they're * normally configurable for the global object.
label: code-design

33912. address

33913. * Property already exists. Steps 5-6 detect whether any changes need * to be made.

33914. may be NULL

33915. * Parse Ecmascript source InputElementDiv or InputElementRegExp * (E5 Section 7), skipping whitespace, comments, and line terminators. * * Possible results are: * (1) a token * (2) a line terminator (skipped) * (3) a comment (skipped) * (4) EOF * * White space is automatically skipped from the current position (but * not after the input element). If input has already ended, returns * DUK_TOK_EOF indefinitely. If a parse error occurs, uses an DUK_ERROR() * macro call (and hence a longjmp through current heap longjmp context). * Comments and line terminator tokens are automatically skipped. * * The input element being matched is determined by regexp_mode; if set, * parses a InputElementRegExp, otherwise a InputElementDiv. The * difference between these are handling of productions starting with a * forward slash. * * If strict_mode is set, recognizes additional future reserved words * specific to strict mode, and refuses to parse octal literals. * * The matching strategy below is to (currently) use a six character * lookup window to quickly determine which production is the -longest- * matching one, and then parse that. The top-level if-else clauses * match the first character, and the code blocks for each clause * handle -all- alternatives for that first character. Ecmascript * specification uses the "longest match wins" semantics, so the order * of the if-clauses matters. * * Misc notes: * * * Ecmascript numeric literals do not accept a sign character. * Consequently e.g. "-1.0" is parsed as two tokens: a negative * sign and a positive numeric literal. The compiler performs * the negation during compilation, so this has no adverse impact. * * * There is no token for "undefined": it is just a value available * from the global object (or simply established by doing a reference * to an undefined value). * * * Some contexts want Identifier tokens, which are IdentifierNames * excluding reserved words, while some contexts want IdentifierNames * directly. In the latter case e.g. "while" is interpreted as an * identifier name, not a DUK_TOK WHILE token. The solution here is * to provide both token types: DUK_TOK WHILE goes to 't' while * DUK_TOK_IDENTIFIER goes to 't_nores', and 'slot1' always contains * the identifier / keyword name. * * * Directive prologue needs to identify string literals such as * "use strict" and 'use strict', which are sensitive to line * continuations and escape sequences. For instance, "use\0020strict" * is a valid directive but is distinct from "use strict". The solution * here is to decode escapes while tokenizing, but to keep track of the * number of escapes. Directive detection can then check that the * number of escapes is zero. * * * Multi-line comments with one or more internal LineTerminator are * treated like a line terminator to comply with automatic semicolon * insertion.

33916. IEEE double is approximately 16 decimal digits; print a couple extra

33917. * Resizing and hash behavior

33918. String sanitizer which escapes ASCII control characters and a few other * ASCII characters, passes Unicode as is, and replaces invalid UTF-8 with * question marks. No errors are thrown for any input string, except in out * of memory situations.

33919. **comment:** Try to optimize X <op>= Y for reg-bound * variables. Detect side-effect free RHS * narrowly by seeing whether it emits code. * If not, rewind the code emitter and overwrite * the unnecessary temp reg load.
label: code-design

33920. [obj key undefined]

33921. **comment:** XXX: can be optimized for smaller footprint esp. on 32-bit environments
label: code-design

33922. * isArray()

33923. DUK_USE_HOBJECT_HASH_PART

33924. trailing elisions?

33925. * Simple commands

33926. XXX: The TypeError is currently not applied to bound * functions because the 'strict' flag is not copied by * bind(). This may or may not be correct, the specification * only refers to the value being a "strict mode Function * object" which is ambiguous.

33927. [... global val] -> [... global]

33928. length in codepoints (must be E5 compatible)

33929. The smallest fastint is no longer 48-bit when * negated. Positive zero becomes negative zero * (cannot be represented) when negated.

33930. Decode helper. Return zero on error.

33931. XXX: specify array size, as we know it

33932. **comment:** Numbers are normalized to big (network) endian. We can * (but are not required) to use integer dvalues when there's * no loss of precision. * * XXX: share check with other code; this check is slow but * reliable and doesn't require careful exponent/mantissa * mask tricks as in the fastint downgrade code.
label: code-design

33933. * Use static helpers which can work with math.h functions matching * the following signatures. This is not portable if any of these math * functions is actually a macro. * * Typing here is intentionally 'double' wherever values interact with * the standard library APIs.

33934. 1111 0xxx; 4 bytes

33935. start array index of current MPUTARR set

33936. cp == -1 (EOF) never matches and causes return value 0

33937. **comment:** XXX: the handling of character range detection is a bit convoluted. * Try to simplify and make smaller.
label: code-design

33938. DUK_OP_CALL flags in A

33939. * Shared helper for non-bound func lookup. ** Returns duk_hobject * to the final non-bound function (NULL for lightfunc).
33940. Must be ≥ 0 and multiple of element size.
33941. A stubbed built-in is useful for e.g. compilation torture testing with BCC.
33942. Decode a plain string consisting entirely of identifier characters. * Used to parse plain keys (e.g. "foo: 123").
33943. note that we can't reliably pop anything here
33944. [... put_value]
33945. 5
33946. ref.value and ref.this_binding invalidated here
33947. nret
33948. object is a compiled function (duk_hcompiledfunction)
33949. Push a new closure on the stack. ** Note: if fun_temp has NEWENV, i.e. a new lexical and variable declaration * is created when the function is called, only outer_lex_env matters * (outer_var_env is ignored and may or may not be same as outer_lex_env).
33950. Need a short reg/const, does not have to be a mutable temp.
33951. delete Duktape.modLoaded[resolved_id]
33952. keep current valstack_top
33953. Disabled until fixed, see above.
33954. 'message'
33955. Just transfer the refcount from act->prev_caller to tv_caller, * so no need for a refcount update. This is the expected case.
33956. Suggested step-by-step method from documentation of RtTimeToSecondsSince1970: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928(v=vs.85).aspx)
33957. * Migrate array to start of entries if requested. ** Note: from an enumeration perspective the order of entry keys matters. * Array keys should appear wherever they appeared before the array abandon * operation.
33958. 0x70...0x7f
33959. Maximum iteration count for computing UTC-to-local time offset when * creating an Ecmascript time value from local parts.
33960. **comment:** XXX: depend on available temps?
 label: code-design
33961. default case
33962. %u; only 16 bits are guaranteed
33963. MultiplicativeExpression
33964. Trim white space (= allow leading and trailing whitespace)
33965. * Call user provided module search function and build the wrapped * module source code (if necessary). The module search function * can be used to implement pure Ecmascript, pure C, and mixed * Ecmascript/C modules. ** The module search function can operate on the exports table directly * (e.g. DLL code can register values to it). It can also return a * string which is interpreted as module source code (if a non-string * is returned the module is assumed to be a pure C one). If a module * cannot be found, an error must be thrown by the user callback. ** Because Duktape.modLoaded[] already contains the module being * loaded, circular references for C modules should also work * (although expected to be quite rare).
33966. One particular problem case is where an object has been * queued for finalization but the finalizer hasn't yet been * executed. ** Corner case: we're running in a finalizer for object X, and * user code calls duk_push_heapptr() for X itself. In this * case X will be in finalize_list, and we can detect the case * by seeing that X's FINALIZED flag is set (which is done before * the finalizer starts executing).
33967. does not modify tv_x
33968. 26: setSeconds
33969. patch pending jump and split
33970. traceback depth doesn't take into account the filename/line * special handling above (intentional)
33971. DUK_USE_ERRTHROW || DUK_USE_ERRCREATE
33972. only needed by debugger for now
33973. Compute day number of the first day of a given year.
33974. ignore_loop
33975. * Begin
33976. Output #1: resolved absolute name
33977. and even number
33978. * DELPROP: Ecmascript property deletion.
33979. normal: implicit leading 1-bit
33980. may throw an error
33981. * Encoding constants, must match genbuiltins.py
33982. 'RegExp'
33983. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize?
 label: code-design
33984. <<
33985. **comment:** XXX: double evaluation of DUK_HCOMPILEDFUNCTION_GET_DATA()
 label: code-design
33986. ensure full memcmp() fits in while
33987. unreachable
33988. push operation normalizes NaNs
33989. getter/setter
33990. x == 0x00 (EOF) causes syntax_error
33991. XXX: range assert
33992. **comment:** XXX: move masks to js_ctx? they don't change during one * fast path invocation.
 label: code-design
33993. flags: "i"
33994. * Init stringtable: probe variant
33995. Catches both doubles and cases where only one argument is a fastint
33996. key coercion (unless already coerced above)
33997. **comment:** XXX: There's quite a bit of overlap with buffer creation handling in * duk_bi_buffer.c. Look for overlap and refactor.
 label: code-design
33998. Limit checks for bytecode byte size and line number.
33999. The necessary #includes are in place in duk_config.h.
34000. **comment:** XXX: this could be optimized
 label: code-design
34001. duk_handle_return() is guaranteed never to throw, except * for potential out-of-memory situations which will then * propagate out of the executor longjmp handler.
34002. **comment:** * XXX: if duk_handle_call() took values through indices, this could be * made much more sensible. However, duk_handle_call() needs to fudge * the 'this' and 'func' values to handle bound function chains, which * is now done "in-place", so this is not a trivial change.
 label: code-design
34003. Number serialization has a significant impact relative to * other fast path code, so careful fast path for fastints.
34004. * Compiler intermediate values ** Intermediate values describe either plain values (e.g. strings or * numbers) or binary operations which have not yet been coerced into * either a left-hand-side or right-hand-side role (e.g. object property).
34005. **comment:** Silence a few global unused warnings here.
 label: code-design
34006. -> [...] arr num]

34007. tzoffset seconds are dropped; 16 bits suffice for * time offset in minutes
34008. thread has yielded
34009. Lightfuncs are always strict.
34010. '\xffArgs'
34011. **comment:** XXX: for array instances we could take a shortcut here and assume * Array.prototype doesn't contain an array index property.
 label: code-design
34012. PropertyName -> IdentifierName | StringLiteral | NumericLiteral
34013. -1 = top callstack entry, callstack[callstack_top - 1] * -callstack_top = bottom callstack entry, callstack[0]
34014. Further state-dependent pre-checks
34015. E5 Section 8.12.8
34016. is_decl
34017. at least one activation, ours
34018. s < b^(-e) * 2
34019. use p_src_base from now on
34020. 'taint' result as complex
34021. no issues with memcmp() zero size, even if broken
34022. * Disjunction struct: result of parsing a disjunction
34023. * Emit initializers in sets of maximum max_init_pairs keys. * Setter/getter is handled separately and terminates the * current set of initializer values. Corner cases such as * single value initializers do not have special handling now.
34024. * E5 Section 7.3: CR LF is detected as a single line terminator for * line numbers. Here we also detect it as a single line terminator * token.
34025. We don't need to sync back thr->ptr_curr_pc here because * the bytecode executor always has a setjmp catchpoint which * does that before errors propagate to here.
34026. Previous entry was inside visited[], nothing to do.
34027. **comment:** XXX: optimize temp reg use
 label: code-design
34028.toFixed()
34029. Detect zero special case.
34030. Number and (minimum) size of bigints in the nc_ctx structure.
34031. We come here for actual aborts (like encountering . toJSON()) * but also for recursion/loop errors. Bufwriter size can be * kept because we'll probably need at least as much as we've * allocated so far.
34032. * Preliminaries: trim, sign, Infinity check * * We rely on the interned string having a NUL terminator, which will * cause a parse failure wherever it is encountered. As a result, we * don't need separate pointer checks. * * There is no special parsing for 'NaN' in the specification although * 'Infinity' (with an optional sign) is allowed in some contexts. * Some contexts allow plus/minus sign, while others only allow the * minus sign (like JSON.parse()). * * Automatic hex number detection (leading '0x' or '0X') and octal * number detection (leading '0' followed by at least one octal digit) * is done here too.
34033. duk_unicode_idp_m_ids_noabmp[]
34034. '(?:)'
34035. * Process replacer/proplist (2nd argument to JSON.stringify)
34036. may be a string constant
34037. return total chars written excluding terminator
34038. [(builtin objects) name]
34039. new_free / size <= 1 / DIV <=> new_free <= size / DIV
34040. This shouldn't happen; call sites should avoid looking up * _Finalizer "through" a Proxy, but ignore if we come here * with a Proxy to avoid finalizer re-entry.
34041. [... holder name val enum obj_key]
34042. * Create a new function object based on a "template function" which contains * compiled bytecode, constants, etc, but lacks a lexical environment. * * ECMAScript requires that each created closure is a separate object, with * its own set of editable properties. However, structured property values * (such as the formal arguments list and the variable map) are shared. * Also the bytecode, constants, and inner functions are shared. * * See E5 Section 13.2 for detailed requirements on the function objects; * there are no similar requirements for function "templates" which are an * implementation dependent internal feature. Also see function-objects.rst * for a discussion on the function instance properties provided by this * implementation. * * Notes: * * * Order of internal properties should match frequency of use, since the * properties will be linearly scanned on lookup (functions usually don't * have enough properties to warrant a hash part). * * * The created closure is independent of its template; they do share the * same 'data' buffer object, but the template object itself can be freed * even if the closure object remains reachable.
34043. E5 Section 11.13.1 (and others) step 4 never matches for prop writes -> no check
34044. exit() afterwards to satisfy "noreturn"
34045. string compare is the default (a bit oddly)
34046. '%xx%xx...%xx', p points to char after first '%'
34047. **comment:** XXX: need a toplain_ignore() which will only coerce a value to a temp * register if it might have a side effect. Side-effect free values do not * need to be coerced.
 label: code-design
34048. 2^4 -> 1/16 = 6.25% spare
34049. * reduce(), reduceRight()
34050. * Macros for accessing size fields
34051. * Resolve module identifier into canonical absolute form.
34052. leap year: sunday, monday, ...
34053. even after a detach and possible reattach
34054. tval
34055. **comment:** integer mixed endian not really used now
 label: code-design
34056. only advance if not tainted
34057. tail insert: don't disturb head in case refzero is running
34058. indexOf: NaN should cause pos to be zero. * lastIndexOf: NaN should cause pos to be +Infinity * (and later be clamped to len).
34059. 0.000...
34060. leave stack unbalanced on purpose
34061. [... func this <bound args> arg1 ... argN]
34062. Bitfield for each DUK_HBUFFEROBJECT_ELEM_xxx indicating which element types * are compatible with a blind byte copy for the TypedArray set() method (also * used for TypedArray constructor). Array index is target buffer elem type, * bitfield indicates compatible source types. The types must have same byte * size and they must be coercion compatible.
34063. **comment:** Run fake finalizer. Avoid creating unnecessary garbage.
 label: code-design
34064. * Table for hex encoding bytes
34065. [... buf func] -> [... func]
34066. build result as: (r << 32) + s: start with (BD + E + F)
34067. We only get here when doing non-standard JSON encoding
34068. Without heap pointer compression duk_hbuffer_dynamic and duk_hbuffer_external * have the same layout so checking for fixed vs. dynamic (or external) is enough.
34069. [target] -> [enum]
34070. Handle both full and partial slice (as long as covered).
34071. envrec: (declarative) record is closed

34072. **comment:** Log frontend shared helper, magic value indicates log level. Provides * frontend functions: trace(), debug(), info(), warn(), error(), fatal(). * This needs to have small footprint, reasonable performance, minimal * memory churn, etc.
label: code-design

34073. XXX: bump preventcount by one for the duration of this call?

34074. for

34075. 15: getUTCDay

34076. need_bytes may be zero

34077. register binding lookup is based on varmap (even in first pass)

34078. prototype: the only internal property lifted outside 'e' as it is so central

34079. 3 entries actually needed below

34080. end of input and last char has been processed

34081. [start length str]

34082. array

34083. fast paths for space and tab

34084. Significand ('f') padding.

34085. **comment:** not needed
label: requirement

34086. Regardless of whether property is found in entry or array part, * it may have arguments exotic behavior (array indices may reside * in entry part for abandoned / non-existent array parts).

34087. add a new pending match jump for latest finished alternative

34088. exclusive

34089. DUK_USE_BUILTIN_INITJS

34090. * If selftests enabled, run them as early as possible

34091. ivalue/ispec helpers

34092. * Determine the effective 'this' binding and coerce the current value * on the valstack to the effective one (in-place, at idx_this). * * The current this value in the valstack (at idx_this) represents either: * - the caller's requested 'this' binding; or * - a 'this' binding accumulated from the bound function chain * * The final 'this' binding for the target function may still be * different, and is determined as described in E5 Section 10.4.3. * * For global and eval code (E5 Sections 10.4.1 and 10.4.2), we assume * that the caller has provided the correct 'this' binding explicitly * when calling, i.e.: * * - global code: this=global object * - direct eval: this=copy from eval() caller's this binding * - other eval: this=global object * * Note: this function may cause a recursive function call with arbitrary * side effects, because ToObject() may be called.

34093. idx_this = idx_func + 1

34094. **comment:** XXX: for real world code, could just ignore array inheritance * and only look at array own properties.
label: code-design

34095. not needed, as we exit right away

34096. XXX: return type should probably be duk_size_t, or explicit checks are needed for * maximum size.

34097. relookup, may have changed

34098. leading digit + fractions

34099. Select appropriate escape format automatically, and set 'tmp' to a * value encoding both the escape format character and the nybble count: * * (nybble_count << 16) | (escape_char1) | (escape_char2)

34100. tc1 = true, tc2 = true

34101. See comments below on MakeTime why these are volatile.

34102. -> [val obj val]

34103. Store lexer position, restoring if quantifier is invalid.

34104. no need to compact since we already did that in duk_abandon_array_checked() * (regardless of whether an array part existed or not).

34105. marker; not actual tagged type

34106. **comment:** XXX: expensive check (also shared elsewhere - so add a shared internal API call?)
label: code-design

34107. For NaN/inf, the return value doesn't matter.

34108. curr char

34109. [regexp string]

34110. **comment:** NULL not needed here
label: code-design

34111. duk_pcall_prop() may itself throw an error, but we're content * in catching the obvious errors (like toLogString() throwing an * error).

34112. special result value handling

34113. [... Logger clog res]

34114. [... put_value varname]

34115. intermediate join to avoid valstack overflow

34116. ptr before mark-and-sweep

34117. end-of-input breaks

34118. reserve a jumps slot after instr before target spilling, used for NEXTENUM

34119. a label site has been emitted by duk_parse_stmt() automatically * (it will also emit the ENDLABEL).

34120. * Variable access

34121. **comment:** function: function must not be tail called
label: code-design

34122. third arg: absolute index (to entire valstack) of idx_bottom of new activation

34123. **comment:** * HASPROP variant used internally. * * This primitive must never throw an error, callers rely on this. * In particular, don't throw an error for prototype loops; instead, * pretend like the property doesn't exist if a prototype sanity limit * is reached. * * Does not implement proxy behavior: if applied to a proxy object, * returns key existence on the proxy object itself.
label: code-design

34124. 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.

34125. !DUK_USE_HSTRING_CLEN

34126. DUK_ERR_EVAL: no macros needed

34127. DUK_USE_PREFER_SIZE

34128. Inref copies, keep originals.

34129. object is a buffer object (duk_hbufferobject) (always exotic)

34130. recheck that the property still exists

34131. jump from true part to end

34132. **comment:** XXX: fastint fast path would be very useful here
label: code-design

34133. XXX: happens e.g. when evaluating: String(Buffer.prototype).

34134. 'name'

34135. eat identifier

34136. * Init stringtable: fixed variant

34137. Copy values through direct validated reads and writes.

34138. break matches always

34139. * "length" maps to number of formals (E5 Section 13.2) for function * declarations/expressions (non-bound functions). Note that 'nargs' * is NOT necessarily equal to the number of arguments.

34140. -1 for opcode

34141. * Regexp instance check, bytecode check, input coercion. ** See E5 Section 15.10.6.
34142. **comment:** hash part size or 0 if unused
label: code-design
34143. **comment:** XXX: this duplicates functionality in duk_regex.c where a similar loop is * required anyway. We could use that BUT we need to update the regexp compiler * 'nranges' too. Work this out a bit more cleanly to save space.
label: code-design
34144. helper to insert a (non-string) heap object into heap allocated list
34145. * Internal API calls which have (stack and other) semantics similar * to the public API.
34146. retval is directly usable
34147. **comment:** * Error throwing helpers ** The goal is to provide verbose and configurable error messages. Call * sites should be clean in source code and compile to a small footprint. * Small footprint is also useful for performance because small cold paths * reduce code cache pressure. Adding macros here only makes sense if there * are enough call sites to get concrete benefits.
label: code-design
34148. digit count
34149. * Hobject property set/get functionality. ** This is very central functionality for size, performance, and compliance. * It is also rather intricate; see hobject-algorithms.rst for discussion on * the algorithms and memory-management.rst for discussion on refcounts and * side effect issues. ** Notes: ** - It might be tempting to assert "refcount nonzero" for objects * being operated on, but that's not always correct: objects with * a zero refcount may be operated on by the refcount implementation * (finalization) for instance. Hence, no refcount assertions are made. ** - Many operations (memory allocation, identifier operations, etc) * may cause arbitrary side effects (e.g. through GC and finalization). * These side effects may invalidate duk_tval pointers which point to * areas subject to reallocation (like value stack). Heap objects * themselves have stable pointers. Holding heap object pointers or * duk_tval copies is not problematic with respect to side effects; * care must be taken when holding and using argument duk_tval pointers. ** - If a finalizer is executed, it may operate on the same object * we're currently dealing with. For instance, the finalizer might * delete a certain property which has already been looked up and * confirmed to exist. Ideally finalizers would be disabled if GC * happens during property access. At the moment property table realloc * disables finalizers, and all DECREFs may cause arbitrary changes so * handle DECREF carefully. ** - The order of operations for a DECREF matters. When DECREF is executed, * the entire object graph must be consistent; note that a refzero may * lead to a mark-and-sweep through a refcount finalizer.
34150. **comment:** * Compact the closure, in most cases no properties will be added later. * Also, without this the closures end up having unused property slots * (e.g. in Duktape 0.9.0, 8 slots would be allocated and only 7 used). * A better future solution would be to allocate the closure directly * to correct size (and setup the properties directly without going * through the API).
label: code-design
34151. 2/3 -> 1/8 = 12.5% min growth
34152. * Reference count updates ** Note: careful manipulation of refcounts. The top is * not updated yet, so all the activations are reachable * for mark-and-sweep (which may be triggered by decref). * However, the pointers are NULL so this is not an issue.
34153. 'hex'
34154. borrowed label name
34155. * String table algorithm: closed hashing with a probe sequence ** This is the default algorithm and works fine for environments with * minimal memory constraints.
34156. Longjmp state is kept clean in success path
34157. Longjmp callers are required to sync-and-null thr->ptr_curr_pc * before longjmp.
34158. Would indicate corrupted lists.
34159. Return invalid index; if caller uses this without checking * in another API call, the index won't map to a valid stack * entry.
34160. **comment:** XXX: faster internal way to get this
label: code-design
34161. old value is number: no refcount
34162. Using classification has smaller footprint than direct comparison.
34163. DUK_USE_ROM_GLOBAL_CLONE || DUK_USE_ROM_GLOBAL_INHERIT
34164. leading byte of match string
34165. **comment:** * Formal argument list ** We don't check for prohibited names or for duplicate argument * names here, because we don't yet know whether the function will * be strict. Function body parsing handles this retroactively.
label: code-design
34166. canonicalized by compiler
34167. [source template closure this]
34168. initial index
34169. 5 heap flags
34170. DUK_TOK_FUNCTION
34171. exponent for 'f'
34172. * Duktape includes (other than duk_features.h) ** The header files expect to be included in an order which satisfies header * dependencies correctly (the headers themselves don't include any other * includes). Forward declarations are used to break circular struct/typedef * dependencies.
34173. always 1 arg
34174. helpers
34175. * Single source autogenerated distributable for Duktape 1.8.0. ** Git commit 0a70d7e4c5227c84e3fed5209828973117d02849 (v1.8.0). * Git branch v1.8-maintenance. ** See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.
34176. ignore result
34177. DUK_JMPBUF_H_INCLUDED
34178. Newline allows module last line to contain a // comment.
34179. 'static'
34180. * sort() ** Currently qsort with random pivot. This is now really, really slow, * because there is no fast path for array parts. ** Signed indices are used because qsort() leaves and degenerate cases * may use a negative offset.
34181. extended types: compatible encoding
34182. **comment:** not allowed in strict mode, regardless of whether resolves; * in non-strict mode DELVAR handles both non-resolving and * resolving cases (the specification description is a bit confusing).
label: code-design
34183. valstack limit caller has check, prevents wrapping
34184. Note that byte order doesn't affect this test: all bytes in * 'test' will be 0xFF for two's complement.
34185. Note: number has no explicit tag (in 8-byte representation)
34186. * Calendar helpers ** Some helpers are used for getters and can operate on normalized values * which can be represented with 32-bit signed integers. Other helpers are * needed by setters and operate on un-normalized double values, must watch * out for non-finite numbers etc.
34187. Unlike year, the other parts fit into 16 bits so %d format * is portable.
34188. reg/const for case value
34189. Equivalent year mapping, used to avoid DST trouble when platform * may fail to provide reasonable DST answers for dates outside the * ordinary range (e.g. 1970-2038). An equivalent year has the same * leap-year-ness as the original year and begins on the same weekday * (Jan 1). ** The year 2038 is avoided because there seem to be problems with it * on some platforms. The year 1970 is also avoided as there were * practical problems with it; an equivalent year is used for it too, * which breaks some DST computations for 1970 right now, see e.g. * test-bi-date-tzoffset-brute-fis.js.
34190. signed integer encoding needed to work with UTF-8
34191. Here val would be potentially invalid if we didn't make * a value copy at the caller.
34192. '&'
34193. jump is inserted here (variant 3)
34194. * Closing environment records. ** The environment record MUST be closed with the thread where its activation * is. In other words (if 'env' is open): ** - 'thr' must match _env.thread * - 'func' must match _env.callee * - 'regbase' must match _env.regbase ** These are not looked up from the env to minimize code size. *

* XXX: should access the own properties directly instead of using the API
34195. resume; [... initial_func undefined(= this) resume_value]
34196. ignore reclimit, not constructor
34197. '\xffCallee'
34198. is normalized
34199. Variant for writing duk_tvals so that any heap allocated values are * written out as tagged heap pointers.
34200. Coerce like boolean
34201. mark-and-sweep marking reached a recursion limit and must use multi-pass marking
34202. [(builtin objects)]
34203. **comment:** XXX: tv_func is not actually needed
 label: requirement
34204. * Source filename (or equivalent), for identifying thrown errors.
34205. '__proto__'
34206. **comment:** XXX: this is probably a useful shared helper: for a * duk_hbufferobject, get a validated buffer pointer/length.
 label: code-design
34207. parent env is the prototype
34208. get varenv for varname (callee's declarative lexical environment)
34209. Use a new environment and there's an 'arguments' object. * We need to initialize it right now.
34210. The timevalue must be in valid Ecmascript range, but since a local * time offset can be applied, we need to allow a +/- 24h leeway to * the value. In other words, although the UTC time is within the * Ecmascript range, the local part values can be just outside of it.
34211. Note: can be safely scanned as bytes (undecoded)
34212. =====
34213. No activation matches, use undefined for both .fileName and * .lineNumber (matches what we do with a _Tracedata based * no-match lookup).
34214. also goes into flags
34215. UnaryExpression
34216. **comment:** XXX: return something more useful, so that caller can throw?
 label: code-design
34217. Tailcalls are handled by back-patching the TAILCALL flag to the * already emitted instruction later (in return statement parser). * Since A and C have a special meaning here, they cannot be "shuffled".
34218. all virtual properties are non-configurable and non-writable
34219. [obj trap_result res_arr]
34220. **comment:** * The string conversion here incorporates all the necessary Ecmascript * semantics without attempting to be generic. nc_ctx->digits contains * nc_ctx->count digits (>= 1), with the topmost digit's 'position' * indicated by nc_ctx->k as follows: * * digits="123" count=3 k=0 --> 0.123 * digits="123" count=3 k=1 --> 1.23 * digits="123" count=3 k=5 --> 12300 * digits="123" count=3 k=-1 --> 0.0123 * * Note that the identifier names used for format selection are different * in Burger-Dybvig paper and Ecmascript specification (quite confusingly * so, because e.g. 'k' has a totally different meaning in each). See * documentation for discussion. * * Ecmascript doesn't specify any specific behavior for format selection * (e.g. when to use exponent notation) for non-base-10 numbers. * * The bigint space in the context is reused for string output, as there * is more than enough space for that (>1kB at the moment), and we avoid * allocating even more stack.
 label: code-design
34221. depth check is done when printing an actual type
34222. Preinc/predec for object properties.
34223. * duk_re_range_callback for generating character class ranges. * * When ignoreCase is false, the range is simply emitted as is. * We don't, for instance, eliminate duplicates or overlapping * ranges in a character class. * * When ignoreCase is true, the range needs to be normalized through * canonicalization. Unfortunately a canonicalized version of a * continuous range is not necessarily continuous (e.g. [x-{}] is * continuous but [X-{}] is not). The current algorithm creates the * canonicalized range(s) space efficiently at the cost of compile * time execution time (see doc/regexp.rst for discussion). * * Note that the ctx->nranges is a context-wide temporary value * (this is OK because there cannot be multiple character classes * being parsed simultaneously).
34224. init function state: init valstack allocations
34225. **comment:** Currently about 7*152 = 1064 bytes. The space for these * duk_bignums is used also as a temporary buffer for generating * the final string. This is a bit awkward; a union would be * more correct.
 label: code-design
34226. Check statement type based on the first token type. * * Note: expression parsing helpers expect 'curr_tok' to * contain the first token of the expression upon entry.
34227. **comment:** * The 'in' operator requires an object as its right hand side, * throwing a TypeError unconditionally if this is not the case. * * However, lightfuncs need to behave like fully fledged objects * here to be maximally transparent, so we need to handle them * here.
 label: code-design
34228. * Number-to-string conversion. The semantics of these is very tightly * bound with the Ecmascript semantics required for call sites.
34229. **comment:** XXX: inject test
 label: code-design
34230. **comment:** SCANBUILD: with suitable dmin/dmax limits 'd' is unused
 label: code-design
34231. 0xff => 255 - 256 = -1; 0x80 => 128 - 256 = -128
34232. DUK_USE_PARANOI_ERROR
34233. Does not assume that jump_pc contains a DUK_OP_JUMP previously; this is intentional * to allow e.g. an INVALID opcode be overwritten with a JUMP (label management uses this).
34234. A 32-bit unsigned integer formats to at most 32 digits (the * worst case happens with radix == 2). Output the digits backwards, * and use a memmove() to get them in the right place.
34235. Without ROM objects "needs refcount update" == is heap allocated.
34236. **comment:** Note: 'act' is dangerous here because it may get invalidated at many * points, so we re-lookup it multiple times.
 label: code-design
34237. **comment:** XXX: Add a flag to reject an attempt to re-attach? Otherwise * the detached callback may immediately reattach.
 label: code-design
34238. side effects -> don't use tv_x, tv_y after
34239. [... enum]
34240. if gap is empty, behave as if not given at all
34241. When JX/JC not in use, the type mask above will avoid this case if needed.
34242. DUK_TOK_LT
34243. no value
34244. * HASVAR: check identifier binding from a given environment record * without traversing its parents. * * This primitive is not exposed to user code as such, but is used * internally for e.g. declaration binding instantiation. * * See E5 Sections: * 10.2.1.1 HasBinding(N) * 10.2.1.2 HasBinding(N) * * Note: strictness has no bearing on this check. Hence we don't take * a 'strict' parameter.
34245. hash probe sequence
34246. * Make the C call
34247. DUK_USE_NONSTD_FUNC_STMT
34248. has access to 'this' binding
34249. **comment:** Create a temporary enumerator to get the (non-duplicated) key list; * the enumerator state is initialized without being needed, but that * has little impact.
 label: code-design
34250. **comment:** XXX: optimize for string inputs: no need to coerce to a buffer * which makes a copy of the input.
 label: code-design

34251. initial size guess
34252. Buffer writes are often integers.
34253. env[funcname] = closure
34254. Safe to call multiple times.
34255. Default variants. Selection depends on speed/size preference. * Concretely: with gcc 4.8.1 -Os x64 the difference in final binary * is about +1kB for _FAST variants.
34256. formatted result limited
34257. [] -> [res]
34258. may be NaN
34259. 'undefined', artifact of lookup
34260. assume array part is comprehensive (contains all array indexed elements * or none of them); hence no need to check the entries part here.
34261. lJ value1: value
34262. pushes function template
34263. Identifier, i.e. don't allow reserved words
34264. **comment:** Parse an inner function, adding the function template to the current function's * function table. Return a function number to be used by the outer function. * * Avoiding O(depth^2) inner function parsing is handled here. On the first pass, * compile and register the function normally into the 'funcs' array, also recording * a lexer point (offset/line) to the closing brace of the function. On the second * pass, skip the function and return the same 'fnum' as on the first pass by using * a running counter. * * An unfortunate side effect of this is that when parsing the inner function, almost * nothing is known of the outer function, i.e. the inner function's scope. We don't * need that information at the moment, but it would allow some optimizations if it * were used.
label: code-design
34265. [arg toLogString]
34266. out_token->lineterm set by caller
34267. note: any entries above the catchstack top are garbage and not zeroed
34268. Error code also packs a tracedata related flag.
34269. enumerator must have no keys deleted
34270. Stack size increases or stays the same.
34271. known to be number; in fact an integer
34272. avail_bytes += need_bytes
34273. **comment:** XXX: this illustrates that a C catchpoint implemented using duk_safe_call() * is a bit heavy at the moment. The wrapper compiles to ~180 bytes on x64. * Alternatives would be nice.
label: code-design
34274. temp variable to pass constants and flags to shared code
34275. char length of the atom parsed in this loop
34276. -> [... fn x y]
34277. **comment:** ToBoolean() does not require any operations with side effects so * we can do it efficiently. For footprint it would be better to use * duk_js_tobool()and then push+replace to the result slot.
label: code-design
34278. -> [...]
34279. topmost activation idx_retval is considered garbage, no need to init
34280. * Delayed activation environment record initialization (for functions * with NEWENV). * * The non-delayed initialization is handled by duk_handle_call().
34281. ref.value, ref.this.binding invalidated here by getprop call
34282. * Field accessor macros
34283. [... buf]
34284. idx_base and idx_base+1 get completion value and type
34285. integer number in range
34286. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor). * * Using duk_put_prop() works incorrectly with '__proto__' * if the own property with that name has been deleted. This * does not happen normally, but a clever reviver can trigger * that, see complex reviver case in: test-bug-json-parse-__proto__.js.
label: code-design
34287. Force interrupt right away if we're paused or in "checked mode". * Step out is handled by callstack unwind.
34288. check stack before interning (avoid hanging temp)
34289. The extra (+4) is tight.
34290. Send version identification and flush right afterwards. Note that * we must write raw, unframed bytes here.
34291. DUK_TOK_STATIC
34292. No pointer compression because pointer is potentially outside of * Duktape heap.
34293. 'null'
34294. pc2line
34295. always normalized
34296. JUMP L1 omitted
34297. **comment:** writable, not configurable
label: code-design
34298. * Init/assert flags, copying them where appropriate. Some flags * (like NEWENV) are processed separately below.
34299. big endian
34300. entry_top + 3
34301. must match bytecode defines now; build autogenerate?
34302. Although NAMEBINDING is not directly needed for using * function instances, it's needed by bytecode dump/load * so copy it too.
34303. ''
34304. * Automatically generated by genbuiltins.py, do not edit!
34305. * Debug connection read primitives
34306. Note: when parsing a formal list in non-strict context, e.g. * "implements" is parsed as an identifier. When the function is * later detected to be strict, the argument list must be rechecked * against a larger set of reserved words (that of strict mode). * This is handled by duk_parse_func_body(). Here we recognize * whatever tokens are considered reserved in current strictness * (which is not always enough).
34307. **comment:** XXX: optimize with more direct internal access
label: code-design
34308. __proto__ setter returns 'undefined' on success unlike the * setPrototypeOf() call which returns the target object.
34309. 0x00-0x0f
34310. * String comparison (E5 Section 11.8.5, step 4), which * needs to compare codepoint by codepoint. * * However, UTF-8 allows us to use strcmp directly: the shared * prefix will be encoded identically (UTF-8 has unique encoding) * and the first differing character can be compared with a simple * unsigned byte comparison (which strcmp does). * * This will not work properly for non-xutf-8 strings, but this * is not an issue for compliance.
34311. DUK_TOK_BOR_EQ
34312. Argument must be a string, e.g. a buffer is not allowed.
34313. Note: val is a stable duk_tval pointer. The caller makes * a value copy into its stack frame, so 'tv_val' is not subject * to side effects here.
34314. number of formal arguments
34315. register, constant, or value
34316. Note: intentionally allow leading zeroes here, as the * actual parser will check for them.
34317. Impose a maximum buffer length for now. Restricted artificially to * ensure resize computations or adding a heap header length won't * overflow size_t and that a signed duk_int_t can hold a buffer * length. The limit should be synchronized with DUK_HSTRING_MAX_BYTELEN.
34318. h_code: held in bw_code

34319. * Forward declarations.
34320. heap pointer comparison suffices
34321. Note: 'count' is currently not adjusted by rounding (i.e. the * digits are not "chopped off". That shouldn't matter because * the digit position (absolute or relative) is passed on to the * convert-and-push function.
34322. **comment:** The variable environment for magic variable bindings needs to be * given by the caller and recorded in the arguments object. * * See E5 Section 10.6, the creation of setters/getters. * * The variable environment also provides access to the callee, so * an explicit (internal) callee property is not needed.
label: code-design
34323. All components default to 0 except day-of-month which defaults * to 1. However, because our internal day-of-month is zero-based, * it also defaults to zero here.
34324. Backtrack.
34325. **comment:** XXX: code duplication
label: code-design
34326. arg count
34327. **comment:** Switch to caller's setjmp() catcher so that if an error occurs * during error handling, it is always propagated outwards instead * of causing an infinite loop in our own handler.
label: code-design
34328. [... number] -> [... string]
34329. 'data' (and everything in it) is reachable through h_res now
34330. Fast path.
34331. called as a normal function: return new Date().toString()
34332. -----XX
34333. must be computed after realloc
34334. 'join'
34335. Get the value represented by an duk_ispec to a register or constant. * The caller can control the result by indicating whether or not: * * (1) a constant is allowed (sometimes the caller needs the result to * be in a register) * * (2) a temporary register is required (usually when caller requires * the register to be safely mutable; normally either a bound * register or a temporary register are both OK) * * (3) a forced register target needs to be used * * Bytecode may be emitted to generate the necessary value. The return * value is either a register or a constant.
34336. **comment:** XXX: FP_ZERO check can be removed, the else clause handles it * correctly (preserving sign).
label: code-design
34337. **comment:** XXX: the string shouldn't appear twice, but we now loop to the * end anyway; if fixed, add a looping assertion to ensure there * is no duplicate.
label: code-design
34338. re-lookup first char on first loop
34339. [... buf loop (proplist)]
34340. DUK_BUFOBJ_INT16ARRAY
34341. No net refcount changes.
34342. DUK_USE_DATE_FMT_STRFTIME
34343. eat the right paren
34344. **comment:** A straightforward 64-byte lookup would be faster * and cleaner, but this is shorter.
label: code-design
34345. For this to work, DATEMSK must be set, so this is not very * convenient for an embeddable interpreter.
34346. Fixed buffer; data follows struct, with proper alignment guaranteed by * struct size.
34347. **comment:** * Call handling. * * Main functions are: * * - duk_handle_call_unprotected(): unprotected call to Ecmascript or * Duktape/C function * - duk_handle_call_protected(): protected call to Ecmascript or * Duktape/C function * - duk_handle_safe_call(): make a protected C call within current * activation * - duk_handle_ecma_call_setup(): Ecmascript-to-Ecmascript calls * (not always possible), including tail calls and coroutine resume * * See 'execution.rst'. * * Note: setjmp() and local variables have a nasty interaction, * see execution.rst; non-volatile locals modified after setjmp() * call are not guaranteed to keep their value.
label: code-design
34348. -1 if disjunction is complex, char length if simple
34349. noblame_fileline
34350. Backtrack to previous slash or start of buffer.
34351. func limits
34352. This is a rare property helper; it sets the global thrower (E5 Section 13.2.3) * setter/getter into an object property. This is needed by the 'arguments' * object creation code, function instance creation code, and Function.prototype.bind().
34353. **comment:** Workaround for some exotic platforms where NAN is missing * and the expression (0.0 / 0.0) does NOT result in a NaN. * Such platforms use the global 'duk_computed_nan' which must * be initialized at runtime. Use 'volatile' to ensure that * the compiler will actually do the computation and not try * to do constant folding which might result in the original * problem.
label: code-design
34354. A -> target register (A, A+1) for call setup * (for DUK_OP_CSREGI, 'a' is indirect) * B -> register containing target function (not type checked here)
34355. Return the Buffer to allow chaining: b.fill(0x11).fill(0x22, 3, 5).toString()
34356. **comment:** code_idx: not needed
label: requirement
34357. keep func->h_funcs; inner functions are not reparsed to avoid O(depth^2) parsing
34358. load factor min 25%
34359. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
34360. Note: peek cannot currently trigger a detach * so the dbg_detaching == 0 assert outside the * loop is correct.
34361. Assume interrupt init/counter are properly initialized here.
34362. * Parse a case or default clause.
34363. **comment:** XXX: when this error is caused by a nonexistent * file given to duk_peval_file() or similar, the * error message is not the best possible.
label: code-design
34364. impose a reasonable exponent limit, so that exp * doesn't need to get tracked using a bigint.
34365. marker value; in E5 2^32-1 is not a valid array index (2^32-2 is highest valid)
34366. maxval
34367. valstack space allocated especially for proxy lookup which does a * recursive property lookup
34368. Accept ASCII strings which conform to identifier requirements * as being emitted without key quotes. Since we only accept ASCII * there's no need for actual decoding: 'p' is intentionally signed * so that bytes >= 0x80 extend to negative values and are rejected * as invalid identifier codepoints.
34369. Careful overflow handling. When multiplying by 10: * - 0x19999998 x 10 = 0xfffffff0: no overflow, and adding * 0..9 is safe. * - 0x19999999 x 10 = 0xffffffff: no overflow, adding * 0..5 is safe, 6...9 overflows. * - 0x1999999a x 10 = 0x10000004: always overflow.
34370. The Str(key, holder) operation. * * Stack policy: [... key] -> [...]
34371. * Bitstream encoder
34372. -> [sep ToObject(this) len str sep]
34373. Decrement PC if that was requested, this requires a PC sync.
34374. Array properties have exotic behavior but they are concrete, * so no special handling here. * * Arguments exotic behavior (E5 Section 10.6, [[GetOwnProperty]]) * is only relevant as a post-check implemented below; hence no * check here.
34375. XXX: request a "last statement is terminal" from duk_parse_stmt() and duk_parse_stmts(); * we could avoid the last RETURN if we could ensure there is no way to get here * (directly or via a jump)
34376. [... key_obj key key flags]
34377. no side effects
34378. INACTIVE: no activation, single function value on valstack
34379. Equality may be OK but >length not. Checking * this explicitly avoids some overflow cases * below.

34380. this cannot initiate a detach
34381. done so that duk_mark_heaphdr() works correctly
34382. if highest bit of a register number is set, it refers to a constant instead
34383. **comment:** Slow variants, call to a helper to reduce code size. * Can be used explicitly when size is always more important than speed.
label: code-design
34384. DUK_USE_REFERENCE_COUNTING
34385. Caller ensures space for at least DUK_JSON_MAX_ESC_LEN.
34386. byte offset to buf
34387. * The catch variable must be updated to reflect the new allocated * register for the duration of the catch clause. We need to store * and restore the original value for the varmap entry (if any).
34388. **comment:** unused now
label: code-design
34389. **comment:** stack[0] = callback * stack[1] = thisArg * stack[2] = object * stack[3] = ToUInt32(length) (unused, but avoid unnecessary pop) * stack[4] = result array (or undefined)
label: code-design
34390. still in prologue
34391. heap->dbg_processing: keep on purpose to avoid debugger re-entry in detaching state
34392. fixed-format termination conditions
34393. DUK_TOK_ELSE
34394. * Add line number to a compiler error.
34395. array of function templates: [func1, offset1, line1, func2, offset2, line2] * offset/line points to closing brace to allow skipping on pass 2
34396. Ensure copy is covered by underlying buffers.
34397. longjmp state
34398. * Note: * * - Intentionally attempt (empty) match at char_offset == k_input->clen * * - Negative char_offsets have been eliminated and char_offset is duk_uint32_t * -> no need or use for a negative check
34399. Expression, terminates at a ')'
34400. can't happen when keeping current stack size
34401. Special handling for CALL/NEW/MPUTOBJ/MPUTARR shuffling. * For each, slot B identifies the first register of a range * of registers, so normal shuffling won't work. Instead, * an indirect version of the opcode is used.
34402. DUK_TOK_FOR
34403. Miscellaneous.
34404. -> [... retval]
34405. If a setter/getter is missing (undefined), the descriptor must * still have the property present with the value 'undefined'.
34406. -> [timeval this]
34407. **comment:** Clean up stack
label: code-design
34408. **comment:** dump all allocated entries, unused entries print as 'unused', * note that these may extend beyond current 'length' and look * a bit funny.
label: code-design
34409. unwind to 'yield' caller
34410. How much stack to require on entry to object/array decode
34411. (e.g. k=3, digits=2 -> "12X")
34412. Note: changing from writable to non-writable is OK
34413. unwinds valstack, updating refcounts
34414. For JX, expressing the whole unsigned 32-bit range matters.
34415. r <- 2 * f
34416. currently active breakpoints: NULL term, borrowed pointers
34417. conversions, coercions, comparison, etc
34418. [key val] -> [key]
34419. use stack allocated buffer to ensure reachability in errors (e.g. intern error)
34420. **comment:** * Manually optimized double-to-fastint downgrade check. * * This check has a large impact on performance, especially for fastint * slow paths, so must be changed carefully. The code should probably be * optimized for the case where the result does not fit into a fastint, * to minimize the penalty for "slow path code" dealing with fractions etc. * * At least on one tested soft float ARM platform double-to-int64 coercion * is very slow (and sometimes produces incorrect results, see self tests). * This algorithm combines a fastint compatibility check and extracting the * integer value from an IEEE double for setting the tagged fastint. For * other platforms a more naive approach might be better. * * See doc/fastint.rst for details.
label: code-design
34421. * Misc support functions
34422. ch is a literal character here or -1 if parsed entity was * an escape such as "\s".
34423. Code emission flags, passed in the 'opcode' field. Opcode + flags * fit into 16 bits for now, so use duk_small_uint.t.
34424. array of formal argument names (-> _Formals)
34425. [... func this arg1 ... argN]
34426. * Simple atom * * If atom_char_length is zero, we'll have unbounded execution time for e.g. * /0*x/.exec('x'). We can't just skip the match because it might have some * side effects (for instance, if we allowed captures in simple atoms, the * capture needs to happen). The simple solution below is to force the * quantifier to match at most once, since the additional matches have no effect. * * With a simple atom there can be no capture groups, so no captures need * to be reset.
34427. key encountered as a getter
34428. **comment:** XXX: typing
label: code-design
34429. catches EOF (NUL)
34430. 'act' is no longer accessed, scanbuild fix
34431. **comment:** XXX: Array size is known before and (2 * re_ctx.nsaved) but not taken * advantage of now. The array is not compacted either, as regexp match * objects are usually short lived.
label: code-design
34432. Ecmascript activation + Duktape.Thread.yield() activation
34433. **comment:** Presence of 'fun' is config based, there's a marginal performance * difference and the best option is architecture dependent.
label: code-design
34434. [... RegExp]
34435. DUK_TOK_DECREMENT
34436. [... errhandler undefined errval]
34437. **comment:** The target for this LDCONST may need output shuffling, but we assume * that 'pc_ldconst' will be the LDCONST that we can patch later. This * should be the case because there's no input shuffling. (If there's * no catch clause, this LDCONST will be replaced with a NOP.)
label: code-design
34438. no wrap assuming h_bufobj->length is valid
34439. Nominal size check.
34440. register bound variables are non-configurable -> always false
34441. -> [obj trap handler target]
34442. reasonable output estimate
34443. Note: DST adjustment is determined using UTC time. * If 'd' is NaN, tzoffset will be 0.
34444. * Character and charcode access
34445. nargs == 2 so we can pass a callstack level to eval().

34446. First pass parse is very lenient (e.g. allows '1.2.3') and extracts a * string for strict number parsing.
34447. normal valued properties
34448. '\xffLexenv'
34449. Module loading failed. Node.js will forget the module * registration so that another require() will try to load * the module again. Mimic that behavior.
34450. **comment:** Note: for an accessor without getter, falling through to * check for "caller" exotic behavior is unnecessary as * "undefined" will never activate the behavior. But it does * no harm, so we'll do it anyway.
 label: code-design
34451. e.g. 12.3 -> digits="123" k=2 -> 1.23e1
34452. **comment:** XXX: there is room to use a shared helper here, many built-ins * check the 'this' type, and if it's an object, check its class, * then get its internal value, etc.
 label: code-design
34453. variable map for pass 2 (identifier -> register number or null (unmapped))
34454. buffer pointer is to an externally allocated buffer
34455. 'Uint32Array'
34456. 'with'
34457. **comment:** * (Re)initialize the temporary byte buffer. May be called extra times * with little impact.
 label: code-design
34458. callstack index of related activation
34459. non-standard
34460. may be safe, or non-safe depending on flags
34461. activation prevents yield (native call or "new")
34462. **comment:** XXX: expensive, but numconv now expects to see a string
 label: code-design
34463. DUK_FLD_VARINT; not relevant here
34464. **comment:** * Note: use indirect realloc variant just in case mark-and-sweep * (finalizers) might resize this same buffer during garbage * collection.
 label: code-design
34465. eat 'try'
34466. 1110 xxxx 10xx xxxx 10xx xxxx
34467. Note: it's nice if size is 2^N (at least for 32-bit platforms).
34468. Quite heavy assert: check valstack policy. Improper * shuffle instructions can write beyond valstack_top/end * so this check catches them in the act.
34469. Expects 'this' at top of stack on entry.
34470. The token in the switch has already been eaten here
34471. currently all labels accept a break, so no explicit check for it now
34472. [arg result]
34473. got lineterm preceding non-whitespace, non-lineterm token
34474. intentional overlap with earlier memzero
34475. Allow exponent
34476. '\xffSource'
34477. read-only object, in code section
34478. **comment:** Slice and accessor information. * * Because the underlying buffer may be dynamic, these may be * invalidated by the buffer being modified so that both offset * and length should be validated before every access. Behavior * when the underlying buffer has changed doesn't need to be clean: * virtual 'length' doesn't need to be affected, reads can return * zero/NaN, and writes can be ignored. * * Note that a data pointer cannot be precomputed because 'buf' may * be dynamic and its pointer unstable.
 label: code-design
34479. Shared object part.
34480. **comment:** XXX: there are language sensitive rules for the ASCII range. * If/when language/locale support is implemented, they need to * be implemented here for the fast path. There are no context * sensitive rules for ASCII range.
 label: code-design
34481. DUK_OP_RETURN flags in A
34482. DUK_TOK_ADD
34483. allocate to reg only (not const)
34484. **comment:** not strictly necessary, but avoids "uninitialized variable" warnings
 label: code-design
34485. * Augment an error at throw time; allow a user error handler (if defined) * to process/replace the error. The error to be augmented is at the * stack top.
34486. **comment:** This macro works when a regconst field is 9 bits, [0,0x1ff]. Adding * DUK_LIKELY/DUK_UNLIKELY increases code footprint and doesn't seem to * improve performance on x64 (and actually harms performance in some tests).
 label: code-design
34487. requested number of output digits; 0 = free-format
34488. * Shared readfield and writefield methods * * The readfield/writefield methods need support for endianness and field * types. All offsets are byte based so no offset shifting is needed.
34489. Gap in array; check for inherited property, * bail out if one exists. This should be enough * to support gappy arrays for all practical code.
34490. **comment:** XXX: could improve bufwriter handling to write multiple codepoints * with one ensure call but the relative benefit would be quite small.
 label: code-design
34491. **comment:** * Delete references to given hstring from the heap string cache. * * String cache references are 'weak': they are not counted towards * reference counts, nor serve as roots for mark-and-sweep. When an * object is about to be freed, such references need to be removed.
 label: code-design
34492. DUK_TOK_EOF
34493. **comment:** XXX: signalling the need to shrink check (only if unwound)
 label: code-design
34494. XXX: duk_dup_unvalidated(ctx, -2) etc.
34495. fill new entries with -unused- (required, gc reachable)
34496. -> [O toString O]
34497. accept anything, expect first value (EOF will be * caught by key parsing below.
34498. unknown, ignore
34499. * Note: fmax() does not match the E5 semantics. E5 requires * that if -any- input to Math.max() is a NaN, the result is a * NaN. fmax() will return a NaN only if - both- inputs are NaN. * Same applies to fmin(). * * Note: every input value must be coerced with ToNumber(), even * if we know the result will be a NaN anyway: ToNumber() may have * side effects for which even order of evaluation matters.
34500. [... v1 v2 name filename]
34501. #DUK_USE_PANIC_HANDLER
34502. **comment:** XXX: inlined DECREF macro would be nice here: no NULL check, * refzero queueing but no refzero algorithm run (= no pointer * instability), inline code.
 label: code-design
34503. expected ival
34504. **comment:** A 16-bit listlen makes sense with 16-bit heap pointers: there * won't be space for 64k strings anyway.
 label: code-design
34505. * Finalize refcounts for heap elements just about to be freed. * This must be done for all objects before freeing to avoid any * stale pointer dereferences. * * Note that this must deduce the set of objects to be freed * identically to duk_sweep_heap().
34506. no sce, or sce scan not best

34507. caller must have decref'd values above new_a_size (if that is necessary)
34508. [... buf loop (proplist) (gap)]
34509. **comment:** XXX: duk_hobject_hasprop() would be correct for * non-Proxy objects too, but it is about ~20-25% * slower at present so separate code paths for * Proxy and non-Proxy now.
label: code-design
34510. Coercion may be needed, the helper handles that by pushing the * tagged values to the stack.
34511. checked by caller
34512. 'Int16Array'
34513. res->strs16[] is zeroed and zero decodes to NULL, so no NULL inits.
34514. entry part must not contain any configurable properties, or * writable properties (if is_frozen).
34515. thread; minimizes argument passing
34516. step 4.d
34517. **comment:** * The Number constructor uses ToNumber(arg) for number coercion * (coercing an undefined argument to NaN). However, if the * argument is not given at all, +0 must be used instead. To do * this, a vararg function is used.
label: code-design
34518. one operator (= assign)
34519. no need to init reg, it will be undefined on entry
34520. string objects must not created without internal value
34521. eat backslash on entry
34522. * Cached module check. * * If module has been loaded or its loading has already begun without * finishing, return the same cached value ('exports'). The value is * registered when module load starts so that circular references can * be supported to some extent.
34523. * Setup a preliminary activation and figure out nargs/nregs. * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
34524. **comment:** XXX: Rework to use an always-inline function?
label: code-design
34525. **comment:** * Common heap header * * All heap objects share the same flags and refcount fields. Objects other * than strings also need to have a single or double linked list pointers * for insertion into the "heap allocated" list. Strings are held in the * heap-wide string table so they don't need link pointers. * * Technically, 'h_refcount' must be wide enough to guarantee that it cannot * wrap (otherwise objects might be freed incorrectly after wrapping). This * means essentially that the refcount field must be as wide as data pointers. * On 64-bit platforms this means that the refcount needs to be 64 bits even * if an 'int' is 32 bits. This is a bit unfortunate, and compromising on * this might be reasonable in the future. * * Heap header size on 32-bit platforms: 8 bytes without reference counting, * 16 bytes with reference counting.
label: code-design
34526. We're a varargs function because we need to detect whether * initialValue was given or not.
34527. cannot be strict (never mapped variables)
34528. Avoid recursive re-enter; enter when we're attached or * detaching (to finish off the pending detach).
34529. DUK_DEBUGGER_H_INCLUDED
34530. Maximum length of CommonJS module identifier to resolve. Length includes * both current module ID, requested (possibly relative) module ID, and a * slash in between.
34531. **comment:** * Property count check. This is the only point where we ensure that * we don't get more (allocated) property space that we can handle. * There aren't hard limits as such, but some algorithms fail (e.g. * finding next higher prime, selecting hash part size) if we get too * close to the 4G property limit. * * Since this works based on allocation size (not actually used size), * the limit is a bit approximate but good enough in practice.
label: code-design
34532. This loop intentionally does not ensure characters are valid * ([0-9a-fA-F]) because the hex decode call below will do that.
34533. 0 if no used entries
34534. * Basic double / byte union memory layout.
34535. copy values (no overlap even if to_ctx == from_ctx; that's not * allowed now anyway)
34536. Use match charlen instead of bytelen, just in case the input and * match codepoint encodings would have different lengths.
34537. **comment:** Count actually used array part entries and array minimum size. * NOTE: 'out_min_size' can be computed much faster by starting from the * end and breaking out early when finding first used entry, but this is * not needed now.
label: code-design
34538. if new_p == NULL, all of these pointers are NULL
34539. -> [... obj retval/error]
34540. [...] holder key] -> [...] holder]
34541. 'this' binding is just before current activation's bottom
34542. **comment:** * Error handling macros, assertion macro, error codes. * * There are three level of 'errors': * * 1. Ordinary errors, relative to a thread, cause a longjmp, catchable. * 2. Fatal errors, relative to a heap, cause fatal handler to be called. * 3. Panic errors, unrelated to a heap and cause a process exit. * * Panics are used by the default fatal error handler and by debug code * such as assertions. By providing a proper fatal error handler, user * code can avoid panics in non-debug builds.
label: code-design
34543. Arguments can be: [source? filename? &comp_args] so that * nargs is 1 to 3. Call site encodes the correct nargs count * directly into flags.
34544. a
34545. Allow leading zeroes (e.g. "0123" -> "123")
34546. * Default clause.
34547. d <- (quotient (* r B) s) (in range 0...B-1)
34548. **comment:** XXX: There used to be a shared log buffer here, but it was removed * when dynamic buffer spare was removed. The problem with using * bufwriter is that, without the spare, the buffer gets passed on * as an argument to the raw() call so it'd need to be resized * (reallocated) anyway. If raw() call convention is changed, this * could be made more efficient.
label: code-design
34549. Commands initiated by debug client.
34550. next loop requires a comma
34551. 0x40...0x4f
34552. 'string'
34553. match == search string, by definition
34554. **comment:** Shared handling for encode/decode argument. Fast path handling for * buffer and string values because they're the most common. In particular, * avoid creating a temporary string or buffer when possible.
label: code-design
34555. 32-bit x 11-bit = 43-bit, fits accurately into a double
34556. Opcode only emitted by compiler when debugger * support is enabled. Ignore it silently without * debugger support, in case it has been loaded * from precompiled bytecode.
34557. eat closing brace
34558. Behavior for constructor and non-constructor call is * the same except for augmenting the created error. When * called as a constructor, the caller (duk_new()) will handle * augmentation; when called as normal function, we need to do * it here.
34559. Encode into a single opcode (18 bits can encode 1-2 bytes + length indicator)
34560. defined(DUK_USE_DATE_NOW_WINDOWS) || defined(DUK_USE_DATE_TZO_WINDOWS)
34561. without finally, the second jump slot is used to jump to end of stmt
34562. slot B is a target (default: source)
34563. fastint constants etc
34564. * Unicode codepoints above U+FFFF are encoded as surrogate * pairs here. This ensures that all CESU-8 codepoints are * 16-bit values as expected in ECMAScript. The surrogate * pairs always get a 3-byte encoding (each) in CESU-8. * See: http://en.wikipedia.org/wiki/Surrogate_pair * * 20-bit codepoint, 10 bits (A and B) per surrogate pair: * * x = 0b00000000 0000AAAA AAAAABBBBBBBB * sp1 = 0b110110AA AAAAAAAA (0xd800 + ((x >> 10) & 0x3ff)) * sp2 =

0b110111BB BBBB BBBB (0xdc00 + (x & 0x3ff)) * * Encoded into CESU-8: * * sp1 -> 0b11101101 (0xe0 + ((sp1 >> 12) & 0x0f)) * -> 0b1010AAAA (0x80 + ((sp1 >> 6) & 0x3f)) * -> 0b10AAAAAA (0x80 + (sp1 & 0x3f)) * sp2 -> 0b11101101 (0xe0 + ((sp2 >> 12) & 0x0f)) * -> 0b1011BBBB (0x80 + ((sp2 >> 6) & 0x3f)) * -> 0b10BBBBBB (0x80 + (sp2 & 0x3f)) * * Note that 0x10000 must be subtracted first. The code below * avoids the sp1, sp2 temporaries which saves around 20 bytes * of code.

34565. Duktape.modLoaded

34566. We could do this operation without caller updating bw_ctx->ptr, * but by writing it back here we can share code better.

34567. Note: if encoding ends by hitting end of input, we don't check that * the encoding is valid, we just assume it is.

34568. **comment:** XXX: call method tail call?

label: code-design

34569. w.../

34570. always provideThis=true

34571. Note: only numbered indices are relevant, so arr_idx fast reject * is good (this is valid unless there are more than 4**32-1 arguments).

34572. * Thread state and calling context checks

34573. token type (with reserved word identification)

34574. **comment:** XXX: there's no explicit recursion bound here now. For the average * qsort recursion depth O(log n) that's not really necessary: e.g. for * 2**32 recursion depth would be about 32 which is OK. However, qsort * worst case recursion depth is O(n) which may be a problem.

label: code-design

34575. verified at beginning

34576. **comment:** Fall back to the initial (original) Object.toString(). We don't * currently have pointers to the built-in functions, only the top * level global objects (like "Array") so this is now done in a bit * of a hacky manner. It would be cleaner to push the (original) * function and use duk_call_method().

label: code-design

34577. XXX: duk_push_uint_string()

34578. Built-in providers

34579. **comment:** XXX: could add flags for "is valid CESU-8" (EcmaScript compatible strings), * "is valid UTF-8", "is valid extended UTF-8" (internal strings are not, * regexp bytecode is), and "contains non-BMP characters". These are not * needed right now.

label: code-design

34580. pop final_cons

34581. **comment:** When ptr == NULL, the format argument is unused.

label: code-design

34582. To convert a heap stridx to a token number, subtract * DUK_STRIDX_START_RESERVED and add DUK_TOK_START_RESERVED.

34583. ANSI C typing

34584. These pointers are at the start of the struct so that they pack * nicely. Mixing pointers and integer values is bad on some * platforms (e.g. if int is 32 bits and pointers are 64 bits).

34585. 1=little endian

34586. * "IdentifierStart" production check.

34587. * Compiler state

34588. **comment:** XXX: this is awkward as we use an internal method which doesn't handle * extensibility etc correctly. Basically we'd want to do a [[DefineOwnProperty]] * or Object.defineProperty() here.

label: code-design

34589. **comment:** * Entries part is a bit more complex

label: code-design

34590. * On first pass, perform actual parsing. Remember valstack top on entry * to restore it later, and switch to using a new function in comp_ctx.

34591. nop: insert top to top

34592. h_match is borrowed, remains reachable through match_obj

34593. 0 = uppercase, 32 = lowercase (= 'a' - 'A')

34594. catch jump

34595. 3: toLocaleString

34596. not reachable

34597. leave 'cat' as top catcher (also works if catchstack exhausted)

34598. +1 for opcode

34599. 'varname' is in stack in this else branch, leaving an unbalanced stack below, * but this doesn't matter now.

34600. Not halfway, round to nearest.

34601. remove in_val

34602. idx_argbase

34603. Strict by default.

34604. fixed precision and zero padding would be required

34605. Ensuring (reserving) space.

34606. flags: "m"

34607. 'setPrototypeOf'

34608. return 1 to allow callers to tail call

34609. * Not found (even in global object)

34610. * Relocate memory with garbage collection, using a callback to provide * the current allocated pointer. This variant is used when a mark-and-sweep * (e.g. finalizers) might change the original pointer.

34611. 'number'

34612. label id allocation (running counter)

34613. **comment:** * Shared error messages: declarations and macros * * Error messages are accessed through macros with fine-grained, explicit * error message distinctions. Concrete error messages are selected by the * macros and multiple macros can map to the same concrete string to save * on code footprint. This allows flexible footprint/verbosity tuning with * minimal code impact. There are a few limitations to this approach: * (1) switching between plain messages and format strings doesn't work * conveniently, and (2) conditional strings are a bit awkward to handle. * * Because format strings behave differently in the call site (they need to * be followed by format arguments), they have a special prefix (DUK_STR_FMT_ * and duk_str_fmt_). * * On some compilers using explicit shared strings is preferable; on others * it may be better to use straight literals because the compiler will combine * them anyway, and such strings won't end up unnecessarily in a symbol table.

label: code-design

34614. **comment:** XXX: how to figure correct size?

label: code-design

34615. **comment:** * See the following documentation for discussion: * * doc/execution.rst: control flow details * * Try, catch, and finally "parts" are Blocks, not Statements, so * they must always be delimited by curly braces. This is unlike e.g. * the if statement, which accepts any Statement. This eliminates any * questions of matching parts of nested try statements. The Block * parsing is implemented inline here (instead of calling out). * * Finally part has a 'let scoped' variable, which requires a few kinks * here.

label: code-design

34616. op_flags

34617. # argument registers target function wants (< 0 => never for ecma calls)

34618. Different calling convention than above used because the helper * is shared.

34619. equality not actually possible

34620. **comment:** XXX: make this leniency dependent on flags or strictness?

label: code-design

34621. [... func this arg1 ... argN envobj]

34622. 32: setDate

34623. slot C is a target (default: source)
34624. -> [... lval rval new_rval]
34625. Sealed and frozen objects cannot gain any more properties, * so this is a good time to compact them.
34626. Parse a single variable declaration (e.g. "i" or "i=10"). A leading 'var' * has already been eaten. These is no return value in 'res', it is used only * as a temporary. * * When called from 'for-in' statement parser, the initializer expression must * not allow the 'in' token. The caller supply additional expression parsing * flags (like DUK_EXPR_FLAG_REJECT_IN) in 'expr_flags'. * * Finally, out_rc_varname and out_reg_varbind are updated to reflect where * the identifier is bound: * * If register bound: out_reg_varbind >= 0, out_rc_varname == 0 (ignore) * If not register bound: out_reg_varbind < 0, out_rc_varname >= 0 * * These allow the caller to use the variable for further assignment, e.g. * as is done in 'for-in' parsing.
34627. sanity
34628. deal with negative values
34629. Unlike for negative arguments, some call sites * want length to be clamped if it's positive.
34630. create bound function object
34631. maximum bytecode copies for {n,m} quantifiers
34632. [... lval rval]
34633. **comment:** Note: Boolean prototype's internal value property is not writable, * but duk_xdef_prop_stridx() disregards the write protection. Boolean * instances are immutable. * * String and buffer special behaviors are already enabled which is not * ideal, but a write to the internal value is not affected by them.
label: code-design
34634. use duk_double_union as duk_tval directly
34635. [... obj finalizer obj heapDestruct] -> [... obj retval]
34636. **comment:** * Assignments are right associative, allows e.g. * a = 5; * a += b = 9; // same as a += (b = 9) * -> expression value 14, a = 14, b = 9 * * Right associativity is reflected in the BP for recursion, * "-1" ensures assignment operations are allowed. * * XXX: just use DUK_BP_COMMA (i.e. no need for 2-step bp levels)?
label: code-design
34637. don't increase 'count'
34638. module.exports = exports
34639. simulate alloc failure on every alloc (except when mark-and-sweep is running)
34640. must relookup act in case of side effects
34641. **comment:** XXX: take advantage of val being unsigned, no need to mask
label: code-design
34642. status booleans
34643. identifier handling
34644. duk_unicode_caseconv_lc()
34645. DUK_TOK_PROTECTED
34646. target
34647. reserved words: future reserved words
34648. Ensure space for maximum multi-character result; estimate is overkill.
34649. * Getters. * * Implementing getters is quite easy. The internal time value is either * NaN, or represents milliseconds (without fractions) from Jan 1, 1970. * The internal time value can be converted to integer parts, and each * part will be normalized and will fit into a 32-bit signed integer. * * A shared native helper to provide all getters. Magic value contains * a set of flags and also packs the date component index argument. The * helper provides: * * getFullYear() * getUTCFullYear() * getMonth() * getUTCMonth() * getDate() * getUTCDate() * getDay() * getUTCDay() * getHours() * getUTCHours() * getMinutes() * getUTCMinutes() * getSeconds() * getUTCSeconds() * getMilliseconds() * getUTCMilliseconds() * getYear() * * Notes: * * - Date.prototype.getDate(): 'date' means day-of-month, and is * zero-based in internal calculations but public API expects it to * be one-based. * * - Date.prototype.getTime() and Date.prototype.valueOf() have identical * behavior. They have separate function objects, but share the same C * function (duk_bi_date_prototype_value_of).
34650. * Pre resize assertions.
34651. In-place unary operation.
34652. Note: expect that caller has already eaten the left paren
34653. implicit leading one
34654. **comment:** * Array index and length * * Array index: E5 Section 15.4 * Array length: E5 Section 15.4.5.1 steps 3.c - 3.d (array length write) * * The DUK_HSTRING_GET_ARRIDX_SLOW() and DUK_HSTRING_GET_ARRIDX_FAST() macros * call duk_js_to_arrayindex_string_helper().
label: code-design
34655. * Internal helpers for managing object 'length'
34656. biased exp == 0 -> denormal, exp -1022
34657. Oxdeadbeef in decimal
34658. strings up to this length are not cached
34659. DUK_HEAPHDR_H_INCLUDED
34660. [... key_obj key val]
34661. Checking callability of the immediate target * is important, same for constructability. * Checking it for functions down the bound * function chain is not strictly necessary * because .bind() should normally reject them. * But it's good to check anyway because it's * technically possible to edit the bound function * chain via internal keys.
34662. **comment:** * Three possible element formats: * 1)PropertyName : AssignmentExpression * 2) getPropertyName () { FunctionBody } * 3) setPropertyName (PropertySetParameterList) { FunctionBody } * * PropertyName can be IdentifierName (includes reserved words), a string * literal, or a number literal. Note that IdentifierName allows 'get' and * 'set' too, so we need to look ahead to the next token to distinguish: * * { get : 1 } * * and * * { get foo() { return 1 } } * { get get() { return 1 } } // 'get' as getter propertyname * * Finally, a trailing comma is allowed. * * Key name is coerced to string at compile time (and ends up as a * string constant) even for numeric keys (e.g. "{1:foo}"). * These could be emitted using e.g. LDINT, but that seems hardly * worth the effort and would increase code size.
label: code-design
34663. keep func->h_argnames; it is fixed for all passes
34664. num entries of new func at entry
34665. One digit octal escape, digit validated.
34666. Use result value as is.
34667. step 15.a
34668. formal arg limits
34669. Use the approach described in "Remarks" of FileTimeToLocalFileTime: * http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx
34670. [offset noAssert], when ftype != DUK_FLD_VARINT
34671. bufwriter for temp accumulation
34672. normal comparison; known: * - both x and y are not NaNs (but one of them can be) * - both x and y are not zero (but one of them can be) * - x and y may be denormal or infinite
34673. -1 if invalid
34674. convenience helpers
34675. * Use Object.defineProperty() helper for the actual operation.
34676. Ensure compact use of temps.
34677. * Object handling: property access and other support functions.
34678. allocated from valstack (fixed buffer)
34679. Unlike most built-ins, the internal [[PrimitiveValue]] of a Date * is mutable.
34680. DUK_TVAL_H_INCLUDED
34681. * Retry with several GC attempts. Initial attempts are made without * emergency mode; later attempts use emergency mode which minimizes * memory allocations forcibly.
34682. last component

34683. lexing (tokenization) state (contains two valstack slot indices)
34684. * Update an existing property of the base object.
34685. update length (curr points to length, and we assume it's still valid)
34686. in-place setup
34687. Currently all built-in native functions are strict. * duk_push_c_function() now sets strict flag, so * assert for it.
34688. The lo/hi indices may be crossed and hi < 0 is possible at entry.
34689. * Init compilation context
34690. add E
34691. '\xffThread'
34692. Must use memmove() because copy area may overlap (source and target * buffer may be the same, or from different slices).
34693. * Special name handling
34694. If function creation fails due to out-of-memory, the data buffer * pointer may be NULL in some cases. That's actually possible for * GC code, but shouldn't be possible here because the incomplete * function will be unwound from the value stack and never instantiated.
34695. A -> register of target object * B -> first register of key/value pair list * C -> number of key/value pairs
34696. **comment:** XXX: this is a major target for size optimization
 label: code-design
34697. 'constructor' property for all built-in objects (which have it) has attributes: * [[Writable]] = true, * [[Enumerable]] = false, * [[Configurable]] = true
34698. XXX: check num_args
34699. pop 'name'
34700. Here 'p' always points to the start of a term. * * We can also unconditionally reset q_last here: if this is * the last (non-empty) term q_last will have the right value * on loop exit.
34701. * All done, switch properties ('p') allocation to new one.
34702. * Duktape.Buffer, Node.js Buffer, and Khronos/ES6 TypedArray built-ins
34703. decode '%xx' to '%xx' if decoded char in reserved set
34704. **comment:** XXX: Finalizer lookup should traverse the prototype chain (to allow * inherited finalizers) but should not invoke accessors or proxy object * behavior.
 At the moment this lookup will invoke proxy behavior, so * caller must ensure that this function is not called if the target is * a Proxy.
 label: code-design
34705. Maximum number of breakpoints. Only breakpoints that are set are * consulted so increasing this has no performance impact.
34706. any other printable -> as is
34707. Called by duk_hstring.h macros
34708. step type: none, step into, step over, step out
34709. **comment:** XXX: faster implementation
 label: code-design
34710. cleared before entering catch part
34711. Clamp an input byte length (already assumed to be within the nominal * duk_hbufferobject 'length') to the current dynamic buffer limits to * yield a byte length limit that's safe for memory accesses. This value * can be invalidated by any side effect because it may trigger a user * callback that resizes the underlying buffer.
34712. **comment:** XXX: ignored now
 label: code-design
34713. **comment:** * Tagged type definition (duk_tval) and accessor macros. * * Access all fields through the accessor macros, as the representation * is quite tricky. * * There are two packed type alternatives: an 8-byte representation * based on an IEEE double (preferred for compactness), and a 12-byte * representation (portability). The latter is needed also in e.g. * 64-bit environments (it usually pads to 16 bytes per value). * * Selecting the tagged type format involves many trade-offs (memory * use, size and performance of generated code, portability, etc), * see doc/types.rst for a detailed discussion (especially of how the * IEEE double format is used to pack tagged values). * * NB: because macro arguments are often expressions, macros should * avoid evaluating their argument more than once.
 label: code-design
34714. * Automatically generated by extract_caseconv.py, do not edit!
34715. * Support functions for duk_heap.
34716. DUK_HCOMPILEDFUNCTION_H_INCLUDED
34717. write_to_array_part:
34718. **comment:** * Parse variant 1 or 2. The first part expression (which differs * in the variants) has already been parsed and its code emitted. * * reg_temps + 0: unused * reg_temps + 1: unused
 label: code-design
34719. E5 Section 15.5.5.1
34720. own pointer
34721. Note: although there is no 'undefined' literal, undefined * values can occur during compilation as a result of e.g. * the 'void' operator.
34722. Note: this may cause a corner case situation where a finalizer * may see a currently reachable activation whose 'func' is NULL.
34723. # total registers target function wants on entry (< 0 => never for ecma calls)
34724. bytecode has a stable pointer
34725. * Thread builtins
34726. overwrite str1
34727. **comment:** XXX: exposed duk_debug_read_buffer
 label: code-design
34728. DUK_JS_BYTECODE_H_INCLUDED
34729. Convert a duk_tval number (caller checks) to a 32-bit index. Returns * DUK__NO_ARRAY_INDEX if the number is not whole or not a valid array * index.
34730. SCANBUILD: scan-build produces a NULL pointer dereference warning * below; it never actually triggers because holder is actually never * NULL.
34731. Set up pointers to the new property area: this is hidden behind a macro * because it is memory layout specific.
34732. no need for refcount update
34733. **comment:** Zero 'count' is also allowed to make call sites easier. * Arithmetic in bytes generates better code in GCC.
 label: code-design
34734. compiler ensures this
34735. [... old_result result] -> [... result]
34736. **comment:** This shouldn't be necessary, but check just in case * to avoid any chance of overruns.
 label: code-design
34737. At the moment Buffer<str> will just use the string bytes as * is (ignoring encoding), so we return the string length here * unconditionally.
34738. * concat()
34739. different memory layout, alloc size, and init
34740. t2 = (* (+ r m+) B)
34741. exponent non-negative (and thus not minimum exponent)
34742. default: NULL, length 0
34743. env is closed, should be missing _Callee, _Thread, _Regbase
34744. **comment:** The E5.1 algorithm checks whether or not a decoded codepoint * is below 0x80 and perhaps may be in the "reserved" set. * This seems pointless because the single byte UTF-8 case is * handled separately, and non-shortest encodings are rejected. * So, 'cp' cannot be below 0x80 here, and thus cannot be in * the reserved set.
 label: code-design
34745. force_flag
34746. Used for minimal 'const': initializer required.
34747. * Table for base-64 encoding
34748. Note: no allocation pressure, no need to check refcounts etc

34749. args incorrect
34750. write on next loop
34751. * Helper to format a time value into caller buffer, used by logging. * 'out_buf' must be at least DUK_BL_DATE_ISO8601_BUFSIZE long.
34752. **comment:** XXX: the returned value is exotic in ES6, but we use a * simple object here with no prototype. Without a prototype, * [[DefaultValue]] coercion fails which is abit confusing. * No callable check/handling in the current Proxy subset.
label: code-design
34753. 2nd related value (type specific)
34754. **comment:** Extraop arithmetic opcodes must have destination same as * first source. If second source matches destination we need * a temporary register to avoid clobbering the second source. * * XXX: change calling code to avoid this situation in most cases.
label: code-design
34755. cannot grow
34756. 0=indexOf, 1=lastIndexOf
34757. bump up "allocated" reg count, just in case
34758. Strict standard behavior, ignore trailing elements for * result 'length'.
34759. For errors a string coerced result is most informative * right now, as the debug client doesn't have the capability * to traverse the error object.
34760. DUK_TOK_IF
34761. **comment:** NOTE: This is a bit fragile. It's important to ensure that * duk_debug_process_messages() never throws an error or * act->curr_pc will never be reset.
label: code-design
34762. Working with the pointer and current size.
34763. object has any exotic behavior(s)
34764. DUK_USE_STRTAB_PROBE
34765. written by a previous RESUME handling
34766. **comment:** * Macro support functions (use only macros in calling code)
label: code-design
34767. string 1 of token (borrowed, stored to ctx->slot1_idx)
34768. **comment:** XXX: more accurate?
label: code-design
34769. Note: assume array part is comprehensive, so that either * the write goes to the array part, or we've abandoned the * array above (and will not come here).
34770. level string
34771. sufficient for keeping temp reg numbers in check
34772. All other object types.
34773. **comment:** Flip highest bit of each byte which changes * the bit pattern 10xxxxxx into 00xxxxxx which * allows an easy bit mask test.
label: test
34774. mixed endian
34775. min_new_size
34776. length in bytes (not counting NUL term)
34777. No finalizers for ROM objects
34778. last element that was left in the heap
34779. compact; no need to seal because object is internal
34780. 0x90...0x9f
34781. const flag for C
34782. [... obj] -> [...]
34783. number of pairs in current MPUTOBJ set
34784. escaped NonEscapeCharacter
34785. Module id requested
34786. Empty searchstring always matches; cpos must be clamped here. * (If q_blen were < 0 due to clamped coercion, it would also be * caught here.)
34787. Never executed if new size is smaller.
34788. 'protected'
34789. because arg count is 1
34790. 'if'
34791. finally free the struct itself
34792. !DUK_USE_PARANOI_ERRORS
34793. Array or Array-like
34794. coerce towards zero
34795. must not be extensible
34796. **comment:** We've ensured space for one escaped input; then * bail out and recheck (this makes escape handling * quite slow but it's uncommon).
label: code-design
34797. mark-and-sweep: finalizable (on current pass)
34798. [... replacer match [captures] match_char_offset input]
34799. DUK_OP_TRYCATCH flags in A
34800. single string token
34801. Use byte copy.
34802. Most common cases first.
34803. no point in supporting encodings of 5 or more bytes
34804. '|'
34805. use_prev_pc
34806. We need ' nbytes' even for a failed offset; return value must be * (offset + nbytes) even when write fails due to invalid offset.
34807. Return value handling.
34808. Enable DUKFUNC exotic behavior once properties are set up.
34809. duk.bi_duk_object_yield() and duk.bi_duk_object_resume() ensure all of these are met
34810. Assumes that caller has normalized NaNs, otherwise trouble ahead.
34811. 0x00: finish (not part of number) * 0x01: continue
34812. **comment:** this is relatively expensive
label: code-design
34813. num_values and temp_start reset at top of outer loop
34814. **comment:** * Panic error * * Panic errors are not relative to either a heap or a thread, and cause * DUK_PANIC() macro to be invoked. Unless a user provides DUK_USE_PANIC_HANDLER, * DUK_PANIC() calls a helper which prints out the error and causes a process * exit. * * The user can override the macro to provide custom handling. A macro is * used to allow the user to have inline panic handling if desired (without * causing a potentially risky function call). * * Panics are only used in debug code such as assertions, and by the default * fatal error handler.
label: code-design
34815. specific assert for wrapping
34816. Could also rely on native sprintf(), but it will handle * values like NaN, Infinity, -0, exponent notation etc in * a JSON-incompatible way.
34817. [arg1 ... argN this loggerLevel loggerName buffer 'raw' buffer]
34818. **comment:** Function declaration for global/eval code is emitted even * for duplicates, because of E5 Section 10.5, step 5.e of * E5.1 (special behavior for variable bound to global object). * * DECLVAR will not re-declare a variable as such, but will * update the binding value.
label: code-design
34819. bits 2...5: type
34820. As a first approximation, buffer values are coerced to strings * for addition. This means that adding two buffers currently * results in a string.

34821. must be ecmascript
34822. 'toUTCString'
34823. Note: no need to re-lookup tv, conversion is side effect free
34824. require.id of current module
34825. low 32 bits is complete
34826. time when status/peek was last done (Date-based rate limit)
34827. [obj key value desc value]
34828. 'caller'
34829. rbp_flags
34830. 'data'
34831. 'type'
34832. **comment:** XXX: shared strings
 label: code-design
34833. -> [... source]
34834. default value
34835. output 3 bytes from 't'
34836. Special flags checks. Since these strings are always * reachable and a string cannot appear twice in the string * table, there's no need to check/set these flags elsewhere. * The 'internal' flag is set by string intern code.
34837. loop iterator init and limit changed from standard algorithm
34838. enum_flags
34839. '++' or '--' in a post-increment/decrement position, * and a LineTerminator occurs between the operator and * the preceding expression. Force the previous expr * to terminate, in effect treating e.g. "a,b\n++" as * "a,b;++" (= SyntaxError).
34840. by default, use caller's environment
34841. * Note: we currently assume that the setjmp() catchpoint is * not re-entrant (longjmp() cannot be called more than once * for a single setjmp()). * * See doc/code-issues.rst for notes on variable assignment * before and after setjmp().
34842. value resides in 'valstack_idx'
34843. Verbose errors with key/value summaries (non-paranoid) or without key/value * summaries (paranoid, for some security sensitive environments), the paranoid * vs. non-paranoid distinction affects only a few specific errors.
34844. 'typeof' must handle unresolvable references without throwing * a ReferenceError (E5 Section 11.4.3). Register mapped values * will never be unresolvable so special handling is only required * when an identifier is a "slow path" one.
34845. free inner references (these exist e.g. when external * strings are enabled)
34846. The 'this' after 'sep' will get ToString() coerced by * duk_join() automatically. We don't want to do that * coercion when providing .fileName or .lineNumber (GH-254).
34847. surrogate pairs get encoded here
34848. 11: getUTCMonth
34849. **comment:** XXX: any chance of unifying this with the 'length' key handling?
 label: code-design
34850. assert just a few critical flags
34851. correction
34852. default to false
34853. * Stack slice primitives
34854. Length in elements: take into account shift, but * intentionally don't check the underlying buffer here.
34855. [key val] -> []
34856. DUK_USE_STRHASH_DENSE
34857. * Get old and new length
34858. clipped codepoint
34859. h_new_proto may be NULL
34860. -> [... errhandler errval]
34861. this is the case for normalized numbers
34862. flags: "gim"
34863. Starting from this round, use emergency mode * for mark-and-sweep.
34864. The eval code is executed within the lexical environment of a specified * activation. For now, use global object eval() function, with the eval * considered a 'direct call to eval'. * * Callstack level for debug commands only affects scope -- the callstack * as seen by, e.g. Duktape.act() will be the same regardless.
34865. * Found existing own or inherited plain property, but original * base is a primitive value.
34866. optional callstack level
34867. e.g. duk_push_string_file_raw() pushed undefined
34868. Assume that either all memory funcs are NULL or non-NUL, mixed * cases will now be unsafe.
34869. 'error'
34870. don't throw for prototype loop
34871. (< (* r 2) s)
34872. [thisArg arg1 ... argN func] (thisArg+args == nargs total)
34873. Note: not set in template (has no "prototype")
34874. unreferenced w/o asserts
34875. if (is_repl_func)
34876. precision:shortest
34877. don't resize stringtable (but may sweep it); needed during stringtable resize
34878. Write curr_pc back for the debugger.
34879. Note: duk_dec_req_stridx() backtracks one char
34880. 'length'
34881. -> [... val retval]
34882. Object property access
34883. * The specification uses RegExp [[Match]] to attempt match at specific * offsets. We don't have such a primitive, so we use an actual RegExp * and tweak lastIndex. Since the RegExp may be non-global, we use a * special variant which forces global-like behavior for matching.
34884. 3
34885. Apply timezone offset to get the main parts in UTC
34886. * Zero sign, see misc/tcc_zerosign2.c.
34887. * Boolean built-ins
34888. LeftHandSideExpression does not allow empty expression
34889. **comment:** XXX: add proper spare handling to dynamic buffer, to minimize * reallocs; currently there is no spare at all.
 label: code-design
34890. Note: escaped characters differentiate directives
34891. defaults, E5 Section 8.6.1, Table 7
34892. * Intern the temporary byte buffer into a valstack slot * (in practice, slot1 or slot2).
34893. special RegExp literal handling after IdentifierName
34894. XXX: byte offset?
34895. 'switch'
34896. * The 'duktape.h' header provides the public API, but also handles all * compiler and platform specific feature detection, Duktape feature * resolution, inclusion of system headers, etc. These have been merged * because the public API is also dependent on e.g. detecting appropriate * C types which is quite platform/compiler

specific especially for a non-C99 * build. The public API is also dependent on the resolved feature set. * * Some actions taken by the merged header (such as including system headers) * are not appropriate for building a user application. The define * DUK_COMPILING_DUKTAPE allows the merged header to skip/include some * sections depending on what is being built.

34897. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * Lightfunc detection happens here too. Note that lightweight functions * can be wrapped by (non-lightweight) bound functions so we must resolve * the bound function chain first.

34898. avoid side effects

34899. result in csreg

34900. * Other heap related defines

34901. * Start doing property attributes updates. Steps 12-13. * * Start by computing new attribute flags without writing yet. * Property type conversion is done above if necessary.

34902. strict: non-deletable, non-writable

34903. * Assuming a register binds to a variable declared within this * function (a declarative binding), the 'this' for the call * setup is always 'undefined'. E5 Section 10.2.1.1.6.

34904. This may happen when forward and backward scanning disagree * (possible for non-extended-UTF-8 strings).

34905. use fallback as 'this' value

34906. Caller has already eaten the first character '(' which we don't need.

34907. r <- (* r B) * s <- s * m+ <- (* m+ B) * m- <- (* m- B) * k <- (- k 1)

34908. A -> object reg * B -> key reg/const * C -> value reg/const * * Note: intentional difference to register arrangement * of e.g. GETPROP; 'A' must contain a register-only value.

34909. C recursion check.

34910. b

34911. Misc

34912. catches EOF (0x00)

34913. **comment:** XXX: duk_get_length?

label: code-design

34914. XXX: if assertions enabled, walk through all valid PCs * and check line mapping.

34915. 'defineProperty'

34916. **comment:** XXX: awkward and bloated asm -- use faster internal accesses

label: code-design

34917. approximation, close enough

34918. marker for detecting internal "double faults", see duk_error_throw.c

34919. three flags

34920. No difference between raw/ensure because the buffer shrinks.

34921. * Determining which datetime components to overwrite based on * stack arguments is a bit complicated, but important to factor * out from setters themselves for compactness. * * If DUK_DATE_FLAG_TIMESETTER, maxnargs indicates setter type: * * 1 -> millisecond * 2 -> second, [millisecond] * 3 -> minute, [second], [millisecond] * 4 -> hour, [minute], [second], [millisecond] * * Else: * * 1 -> date * 2 -> month, [date] * 3 -> year, [month], [date] * * By comparing nargs and maxnargs (and flags) we know which * components to override. We rely on part index ordering.

34922. False if the object is NULL or the prototype 'p' is NULL. * In particular, false if both are NULL (don't compare equal).

34923. Start filling in the activation

34924. SCANBUILD: complains about use of uninitialized values. * The complaint is correct, but operating in undefined * values here is intentional in some cases and the caller * ignores the results.

34925. prevent duk_expr_led() by using a binding power less than anything valid

34926. borrowed reference; although 'tv1' comes from a register, * its value was loaded using LDCONST so the constant will * also exist and be reachable.

34927. [... proplist enum_obj key val]

34928. * Unix-like Date providers * * Generally useful Unix / POSIX / ANSI Date providers.

34929. Push the current 'this' binding; throw TypeError if binding is not object * coercible (CheckObjectCoercible).

34930. All strings beginning with 0xff are treated as "internal", * even strings interned by the user. This allows user code to * create internal properties too, and makes behavior consistent * in case user code happens to use a string also used by Duktape * (such as string has already been interned and has the 'internal' * flag set).

34931. 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [31 bits]

34932. shrink failure is not fatal

34933. !(l < p)

34934. valstack index of loop detection object

34935. limit is quite low: one array entry is 8 bytes, one normal entry is 4+1+8+4 = 17 bytes (with hash entry)

34936. yes -> move back to heap allocated

34937. strict flag for putvar comes from our caller (currently: fixed)

34938. catchstack limits

34939. * Manually optimized number-to-double conversion

34940. [ToObject(this) ToUint32(length) lowerValue upperValue]

34941. if a tail call: * - an Ecmascript activation must be on top of the callstack * - there cannot be any active catchstack entries

34942. [... enum_target res trap_result val true]

34943. Helper for component getter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), push a specified component as a return value to the * value stack and return 1 (caller can then tail call us).

34944. important for callers

34945. fail

34946. [sep ToObject(this) len]

34947. DUK_TOK_PERIOD

34948. Enumeration keys are checked against the enumeration target (to see * that they still exist). In the proxy enumeration case _Target will * be the proxy, and checking key existence against the proxy is not * required (or sensible, as the keys may be fully virtual).

34949. Easiest way to implement the search required by the specification * is to do a RegExp test() with lastIndex forced to zero. To avoid * side effects on the argument, "clone" the RegExp if a RegExp was * given as input. * * The global flag of the RegExp should be ignored; setting lastIndex * to zero (which happens when "cloning" the RegExp) should have an * equivalent effect.

34950. idx_reviver

34951. **comment:** Require a lot of stack to force a value stack grow/shrink. * Recursive mark-and-sweep is prevented by allocation macros * so this won't trigger another mark-and-sweep.

label: code-design

34952. signed shift

34953. Impose a string maximum length, need to handle overflow * correctly.

34954. **comment:** XXX: casts could be improved, especially for GET/SET DATA

label: code-design

34955. bytes left

34956. **comment:** XXX: these last tricks are unnecessary if the function is made * a genuine native function.

label: code-design

34957. No locale specific formatter; this is OK, we fall back * to ISO 8601.

34958. failed

34959. -> [... key]

34960. **comment:** Tested: not faster on x64
 label: code-design
34961. clamp to 10 chars
34962. * "LineTerminator" production check.
34963. A lightfunc might also inherit a .toJSON() so just bail out.
34964. **comment:** Note: this is very tricky; we must never 'overshoot' the * correction downwards.
 label: code-design
34965. **comment:** XXX: inefficient; block insert primitive
 label: code-design
34966. * Exposed helper for setting up heap longjmp state.
34967. in non-packed representation we don't care about which NaN is used
34968. **comment:** Period followed by a digit can only start DecimalLiteral * (handled in slow path). We could jump straight into the * DecimalLiteral handling but should avoid goto to inside * a block.
 label: code-design
34969. zero
34970. Handle change in value stack top. Respect value stack * initialization policy: 'undefined' above top. Note that * DECREF may cause a side effect that reallocates valstack, * so must relookup after DECREF.
34971. **comment:** XXX: tostring?
 label: code-design
34972. 'func' wants stack "as is"
34973. * Debug message processing
34974. Bottom of valstack for this activation, used to reset * valstack_bottom on return; index is absolute. Note: * idx_top not needed because top is set to 'nregs' always * when returning to an EcmaScript activation.
34975. * Sweep heap
34976. Object.keys
34977. map is reachable through obj
34978. **comment:** XXX: this needs to be reworked so that we never shrink the value * stack on function entry so that we never need to grow it here. * Needing to grow here is a sandboxing issue because we need to * allocate which may cause an error in the error handling path * and thus propagate an error out of a protected call.
 label: code-design
34979. **comment:** * Note: since this is an exposed API call, there should be * no way a mark-and-sweep could have a side effect on the * memory allocation behind 'ptr'; the pointer should never * be something that Duktape wants to change. * * Thus, no need to use DUK_REALLOC_INDIRECT (and we don't * have the storage location here anyway).
 label: code-design
34980. required for rehash to succeed, equality not that useful
34981. DUK_TOK_CLASS
34982. **comment:** unused for label
 label: code-design
34983. 'this' binding shouldn't matter here
34984. **comment:** remove old value
 label: code-design
34985. **comment:** get rid of the strings early to minimize memory use before intern
 label: code-design
34986. always in entry part, no need to look up parents etc
34987. * Raw intern and lookup
34988. Copy the property table verbatim; this handles attributes etc. * For ROM objects it's not necessary (or possible) to update * refcounts so leave them as is.
34989. CONDITIONAL EXPRESSION
34990. swap elements; deal with non-existent elements correctly
34991. [... closure/error]
34992. * Create a new property in the original object. * * Exotic properties need to be reconsidered here from a write * perspective (not just property attributes perspective). * However, the property does not exist in the object already, * so this limits the kind of exotic properties that apply.
34993. Initial value for highest_idx is -1 coerced to unsigned. This * is a bit odd, but (highest_idx + 1) will then wrap to 0 below * for out_min_size as intended.
34994. ignore
34995. **comment:** XXX: combine all the integer conversions: they share everything * but the helper function for coercion.
 label: code-design
34996. vararg function, thisArg needs special handling
34997. Assert for value stack initialization policy.
34998. **comment:** * Shallow fast path checks for accessing array elements with numeric * indices. The goal is to try to avoid coercing an array index to an * (interned) string for the most common lookups, in particular, for * standard Array objects. * * Interning is avoided but only for a very narrow set of cases: * - Object has array part, index is within array allocation, and * value is not unused (= key exists) * - Object has no interfering exotic behavior (e.g. arguments or * string object exotic behaviors interfere, array exotic * behavior does not). * * Current shortcoming: if key does not exist (even if it is within * the array allocation range) a slow path lookup with interning is * always required. This can probably be fixed so that there is a * quick fast path for non-existent elements as well, at least for * standard Array objects.
 label: code-design
34999. curr_token slot1 (matches 'lex' slot1_idx)
35000. eat 'break' or 'continue'
35001. zero-based -> one-based
35002. roughly 512 bytes
35003. * indexOf() and lastIndexOf()
35004. DUK_USE_HEAPPTR16
35005. regexp literal must not follow this token
35006. qmin and qmax will be 0 or 1
35007. same as during entry
35008. Have a catch variable.
35009. Note: in strict mode the compiler should reject explicit * declaration of 'eval' or 'arguments'. However, internal * bytecode may declare 'arguments' in the function prologue. * We don't bother checking (or asserting) for these now.
35010. Object property allocation layout
35011. Node.js accepts only actual Arrays.
35012. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined]
35013. **comment:** XXX: Shrink the stacks to minimize memory usage? May not * be worth the effort because terminated threads are usually * garbage collected quite soon.
 label: code-design
35014. * Actual Object.defineProperty() default algorithm.
35015. t1 = milliseconds within day (fits 32 bit) * t2 = day number from epoch (fits 32 bit, may be negative)
35016. 'class'
35017. compare: similar to string comparison but for buffer data.
35018. Note: if 'x' is zero, x->n becomes 0 here
35019. Debugger protocol version is defined in the public API header.
35020. not undefined

35021. Endianness indicator
35022. number of activation records in callstack preventing a yield
35023. DUK_TOK_IMPLEMENT
35024. 4'294'967'295
35025. XXX: switch cast?
35026. 'trace'
35027. array 'length' is always a number, as we coerce it
35028. NULL is allowed, no output
35029. * Convert duk_compiler_func to a function template
35030. Update duk_tval in-place if pointer provided and the * property is writable. If the property is not writable * (immutable binding), use duk_hobject_putprop() which * will respect mutability.
35031. Better equivalent algorithm. If the compiler is compliant, C and * Ecmascript semantics are identical for this particular comparison. * In particular, NaNs must never compare equal and zeroes must compare * equal regardless of sign. Could also use a macro, but this inlines * already nicely (no difference on gcc, for instance).
35032. Is whole and within 32 bit range. If the value happens to be 0xFFFFFFFF, * it's not a valid array index but will then match DUK_NO_ARRAY_INDEX.
35033. * PLAIN: x1 * ARITH: x1 <op> x2 * PROP: x1.x2 * VAR: x1 (name)
35034. eat closing slash
35035. * Regexp range tables
35036. Value stack intentionally mixed size here.
35037. Computation must not wrap; this limit works for 32-bit size_t: * >>> srclen = 3221225469 * >>> "%x" % ((srclen + 2) / 3 * 4) * 'fffffc'
35038. **comment:** * Special post-tweaks, for cases not covered by the init data format. * * - Set Date.prototype.toGMTString to Date.prototype.toUTCString. * toGMTString is required to have the same Function object as * toUTCString in E5 Section B.2.6. Note that while Smjs respects * this, V8 does not (the Function objects are distinct). * * - Make DoubleError non-extensible. * * - Add info about most important effective compile options to Duktape. * * - Possibly remove some properties (values or methods) which are not * desirable with current feature options but are not currently * conditional in init data.
label: code-design
35039. hobject property layout
35040. **comment:** UNUSED, intentionally empty
label: code-design
35041. double quote or backslash
35042. track utf-8 non-continuation bytes
35043. [hobject props enum(props)]
35044. 22 to 31
35045. **comment:** XXX: very similar to DUK_IVAL_ARITH - merge?
label: code-design
35046. **comment:** XXX: optional check, match_caps is zero if no regexp, * so dollar will be interpreted literally anyway.
label: code-design
35047. * Internal helper to define a property with specific flags, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes unless caller requests * that value only be updated if it doesn't already exists. * * Does not support: * - virtual properties (error if write attempted) * - getter/setter properties (error if write attempted) * - non-default (!= WEC) attributes for array entries (error if attempted) * - array abandoning: if array part exists, it is always extended * - array 'length' updating * * Stack: [... in_val] -> [] * * Used for e.g. built-in initialization and environment record * operations.
35048. -> [... enum key enum_target val]
35049. constants arbitrary, chosen for small loads
35050. reset 'allow_in' for parenthesized expression
35051. **comment:** Internal class is Object: Object.prototype.toString.call(new Buffer(0)) * prints "[object Object]".
label: code-design
35052. * The entire allocated buffer area, regardless of actual used * size, is kept zeroed in resizes for simplicity. If the buffer * is grown, zero the new part.
35053. XXX: integrate support for this into led() instead? * Similar issue as post-increment/post-decrement.
35054. E5 Sections 11.8.2, 11.8.5; x > y --> y < x
35055. if present, forces 16-byte duk_tval
35056. * 'arguments' binding is special; if a shadowing argument or * function declaration exists, an arguments object will * definitely not be needed, regardless of whether the identifier * 'arguments' is referenced inside the function body.
35057. exports (this binding)
35058. disabled
35059. **comment:** XXX: other variants like uint, u32 etc
label: code-design
35060. rely on interning, must be this string
35061. Does not allow e.g. 2**31-1, but one more would allow overflows of u32.
35062. Note: cannot be a bound function either right now, * this would be easy to relax though.
35063. Match labels starting from latest; once label_id no longer matches, we can * safely exit without checking the rest of the labels (only the topmost labels * are ever updated).
35064. The 'else' ambiguity is resolved by 'else' binding to the innermost * construct, so greedy matching is correct here.
35065. Short circuit if is safe: if act->curr_pc != NULL, 'fun' is * guaranteed to be a non-NULL Ecmascript function.
35066. read-only values 'lifted' for ease of use
35067. return value
35068. 'source'
35069. 'clog'
35070. digits
35071. * Statement value handling. * * Global code and eval code has an implicit return value * which comes from the last statement with a value * (technically a non- "empty" continuation, which is * different from an empty statement). * * Since we don't know whether a later statement will * override the value of the current statement, we need * to coerce the statement value to a register allocated * for implicit return values. In other cases we need * to coerce the statement value to a plain value to get * any side effects out (consider e.g. "foo.bar;").
35072. is_frozen
35073. **comment:** Downgrade checks are not made everywhere, so 'length' is not always * a fastint (it is a number though). This can be removed once length * is always guaranteed to be a fastint.
label: code-design
35074. if duk_uint32_t is exactly 32 bits, this is a NOP
35075. * Not found as a concrete property, check whether a String object * virtual property matches.
35076. * Function formal arguments, always bound to registers * (there's no support for shuffling them now).
35077. off >= step, and step >= 1
35078. zero-width joiner
35079. Rethrow error to calling state.
35080. ""
35081. next temporary register to allocate
35082. * For bound objects, [[HasInstance]] just calls the target function * [[HasInstance]]. If that is again a bound object, repeat until * we find a non-bound Function object.
35083. Pointer and buffer primitive values are treated like other * primitives values which have a fully fledged object counterpart: * promote to an object value. Lightfuncs are coerced with * ToObject() even they could also be returned as is.
35084. **comment:** XXX: option to fix opcode length so it lines up nicely

label: code-design

35085. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT16

35086. relookup after side effects (no side effects currently however)

35087. Note: this order matters (final value before deleting map entry must be done)

35088. 'arguments'

35089. * Fast path for bufferobject getprop/putprop

35090. XXX: shrink array allocation on entries compaction here?

35091. inline arithmetic check for constant values

35092. pre-incremented, points to first jump slot

35093. * Helpers for duk_compiler_func.

35094. Note: need to re-lookup because ToNumber() may have side effects

35095. expr_flags

35096. **comment:** XXX: exposed duk_debug_read_hbuffer

label: code-design

35097. accept string

35098. unreferenced without assertions

35099. **comment:** XXX: macro checks for array index flag, which is unnecessary here

label: code-design

35100. **comment:** XXX: this algorithm could be optimized quite a lot by using e.g. * a logarithm based estimator for 'k' and performing B^n multiplication * using a lookup table or using some bit-representation based exp * algorithm. Currently we just loop, with significant performance * impact for very large and very small numbers.

label: code-design

35101. 'toString'

35102. first character has been matched

35103. 0xffff0 is -Infinity

35104. **comment:** XXX: don't want to shrink allocation here

label: code-design

35105. -> [... error]

35106. Lightfunc coerces to a Function instance with concrete * properties. Since 'length' is virtual for Duktape/C * functions, don't need to define that. * * The result is made extensible to mimic what happens to * strings: * > Object.isExtensible(Object('foo')) * true

35107. curr_token follows 'function'

35108. * Update preventcount

35109. **comment:** These are not needed to implement quantifier capture handling, * but might be needed at some point.

label: requirement

35110. 'toJSON'

35111. function executes as a constructor (called via "new")

35112. heapdr: * - is not reachable * - is an object * - is not a finalized object * - has a finalizer

35113. DUK_ERROR_H_INCLUDED

35114. Number/string-or-buffer -> coerce string to number (e.g. "'1.5' == 1.5" -> true).

35115. XXX: There are several limitations in the current implementation for * strings with $\geq 0x80000000UL$ characters. In some cases one would need * to be able to represent the range [-0xffffffff,0xffffffff] and so on. * Generally character and byte length are assumed to fit into signed 32 * bits ($< 0x80000000UL$). Places with issues are not marked explicitly * below in all cases, look for signed type usage (duk_int_t etc) for * offsets/lengths.

35116. * Reset function state and perform register allocation, which creates * 'varmap' for second pass. Function prologue for variable declarations, * binding value initializations etc is emitted as a by-product. * * Strict mode restrictions for duplicate and invalid argument * names are checked here now that we know whether the function * is actually strict. See: test-dev-strict-mode-boundary.js. * * Inner functions are compiled during pass 1 and are not reset.

35117. if 1, doing a fixed format output (not free format)

35118. We want to do a straight memory copy if possible: this is * an important operation because .set() is the TypedArray * way to copy chunks of memory. However, because set() * conceptually works in terms of elements, not all views are * compatible with direct byte copying. * * If we do manage a direct copy, the "overlap issue" handled * below can just be solved using memmove() because the source * and destination element sizes are necessarily equal.

35119. borrowed; NULL if no step state (NULLed in unwind)

35120. new buffer with string contents

35121. Allow very small lenience because some compilers won't parse * exact IEEE double constants (happened in matrix testing with * Linux gcc-4.8 -m32 at least).

35122. We could fit built-in index into magic but that'd make the magic * number dependent on built-in numbering (genbuiltins.py doesn't * handle that yet). So map both class and prototype from the * element type.

35123. * Allocate a new thread. * * Leaves the built-ins array uninitialized. The caller must either * initialize a new global context or share existing built-ins from * another thread.

35124. no regexp instance should exist without a non-configurable bytecode property

35125. original input stack before nargs/nregs handling must be * intact for 'arguments' object

35126. replaces top of stack with new value if necessary

35127. 33: setUTCDate

35128. Notes: * - only numbered indices are relevant, so arr_idx fast reject is good * (this is valid unless there are more than $4^{**}32-1$ arguments). * - since variable lookup has no side effects, this can be skipped if * DUK_GETDESC_FLAG_PUSH_VALUE is not set.

35129. Automatic semicolon insertion is allowed if a token is preceded * by line terminator(s), or terminates a statement list (right curly * or EOF).

35130. eat 'for'

35131. init 'curr_token'

35132. -> [... new_global new_globalenv new_global]

35133. Index clamping is a bit tricky, we must ensure that we'll only iterate * through elements that exist and that the specific requirements from E5.1 * Sections 15.4.4.14 and 15.4.4.15 are fulfilled; especially: * * - indexOf: clamp to [-len,len], negative handling -> [0,len], * if clamped result is len, for-loop bails out immediately * * - lastIndexOf: clamp to [-len-1, len-1], negative handling -> [-1, len-1], * if clamped result is -1, for-loop bails out immediately * * If fromIndex is not given, ToInteger(undefined) = 0, which is correct * for indexOf() but incorrect for lastIndexOf(). Hence special handling, * and why lastIndexOf() needs to be a vararg function.

35134. implicit attributes

35135. allow a constant to be returned

35136. prop index in 'entry part', < 0 if not there

35137. add_auto_proto

35138. [... ToObject(this) ToUint32(length) val]

35139. Load bytecode instructions.

35140. unsupported: would consume multiple args

35141. DUK_USE_DEBUGGER_THROW_NOTIFY

35142. * Same type? * * Note: since number values have no explicit tag in the 8-byte * representation, need the awkward if + switch.

35143. There's intentionally no check for * current underlying buffer length.

35144. follow parent chain

35145. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT64

35146. **comment:** XXX: this should be an assert

label: test

35147. without explicit non-BMP support, assume non-BMP characters * are always accepted as letters.

35148. No debugger support, just pop values.

35149. The lookup byte is intentionally sign extended to (at least) * 32 bits and then ORed. This ensures that is at least 1 byte * is negative, the highest bit of 't' will be set at the end * and we don't need to check every byte.

35150. [hobject props enum(props) key desc value? getter? setter?]

35151. start match from beginning

35152. DUK_HOBJECT_FLAG_ARRAY_PART: don't care

35153. Validity checks for various fraction formats ("0.1", ".1", "1.", ".").

35154. DUK_USE_SECTION_B

35155. !DUK_USE_NONSTD_FUNC_CALLER_PROPERTY

35156. **comment:** XXX: better to get base and walk forwards?
label: code-design

35157. 'ownKeys'

35158. **comment:** Call this.raw(msg); look up through the instance allows user to override * the raw() function in the instance or in the prototype for maximum * flexibility.
label: code-design

35159. restore stack top

35160. DUK_USE_BASE64_FASTPATH

35161. Valstack should suffice here, required on function valstack init

35162. duk_hnativefunction specific fields.

35163. r <- (* f b2) [if b==2 -> (* f 4)] * s <- (* (expt b (- 1 e)) 2) == b^(1-e) * 2 [if b==2 -> b^(2-e)] * m+ <- b == 2 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round

35164. Insert bytes in the middle of the buffer from an external buffer.

35165. identifier names (E5 Section 7.6)

35166. * Set up the resolution input which is the requested ID directly * (if absolute or no current module path) or with current module * ID prepended (if relative and current module path exists). * Suppose current module is 'foo/bar' and relative path is './quux'. * The 'bar' component must be replaced so the initial input here is * 'foo/bar/../quux'.

35167. A non-Array object should not have an array part in practice. * But since it is supported internally (and perhaps used at some * point), check and abandon if that's the case.

35168. keep heap->dbg_detached_cb

35169. For tv1 == tv2, both pointing to stack top, the end result * is same as duk_pop(ctx).

35170. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

35171. During parsing, month and day are one-based; set defaults here.

35172. **comment:** * Logger arguments are: * * magic: log level (0-5) * this: logger * stack: plain log args * * We want to minimize memory churn so a two-pass approach * is used: first pass formats arguments and computes final * string length, second pass copies strings either into a * pre-allocated and reused buffer (short messages) or into a * newly allocated fixed buffer. If the backend function plays * nice, it won't coerce the buffer to a string (and thus * intern it).
label: code-design

35173. Non-zero refcounts should not happen because we refcount * finalize all unreachable objects which should cancel out * refcounts (even for cycles).

35174. token allows automatic semicolon insertion (eof or preceded by newline)

35175. Use special opcodes to load short strings

35176. * Add .fileName and .lineNumber to an error on the stack top.

35177. * CommonJS require() and modules support

35178. * Lexing helpers

35179. * Create built-in objects by parsing an init bitstream generated * by genbuiltins.py.

35180. explicit PropertyList

35181. Special handling for call setup instructions. The target * is expressed indirectly, but there is no output shuffling.

35182. * Create mantissa

35183. NOTE: act may be NULL if an error is thrown outside of any activation, * which may happen in the case of, e.g. syntax errors.

35184. ToNumber() for a fastint is a no-op.

35185. [key] -> [undefined] (default value)

35186. traceback depth fits into 16 bits

35187. [thisArg arg1 ... argN func boundFunc argArray]

35188. * Parser duk__advance() token eating functions

35189. **comment:** cleanup varmap from any null entries, compact it, etc; returns number * of final entries after cleanup.
label: code-design

35190. **comment:** * Needs a save of multiple saved[] entries depending on what range * may be overwritten. Because the regexp parser does no such analysis, * we currently save the entire saved array here. Lookaheads are thus * a bit expensive. Note that the saved array is not needed for just * the lookahead sub-match, but for the matching of the entire sequel. * * The temporary save buffer is pushed on to the valstack to handle * errors correctly. Each lookahead causes a C recursion and pushes * more stuff on the value stack. If the C recursion limit is less * than the value stack spare, there is no need to check the stack. * We do so regardless, just in case.
label: code-design

35191. **comment:** XXX: split into separate functions for each field type?
label: code-design

35192. Same as above but for unsigned int range.

35193. comma after a value, expected

35194. duplicate key

35195. Given a day number, determine year and day-within-year.

35196. **comment:** Vararg function: must be careful to check/require arguments. * The JSON helpers accept invalid indices and treat them like * non-existent optional parameters.
label: code-design

35197. Success path handles

35198. **comment:** XXX: getter with class check, useful in built-ins
label: code-design

35199. already updated

35200. checking for DUK__DELETED_MARKER is not necessary here, but helper does it now

35201. match: keep saves

35202. DUK_USE_PARANOIAD_MATH

35203. 'boolean'

35204. * Debugging disabled

35205. fixed offset in valstack

35206. DUK_TOK_ENUM

35207. when key is NULL, value is garbage so no need to set

35208. **comment:** * Error during call. The error value is at heap->lj.value1. * * The very first thing we do is restore the previous setjmp catcher. * This means that any error in error handling will propagate outwards * instead of causing a setjmp() re-entry above.
label: code-design

35209. Standard behavior for map(): trailing non-existent * elements don't invoke the user callback and are not * counted towards result 'length'.

35210. * Append a Unicode codepoint to the temporary byte buffer. Performs * CESU-8 surrogate pair encoding for codepoints above the BMP. * Existing surrogate pairs are allowed and also encoded into CESU-8.

35211. * Proxy helpers

35212. * Found; post-processing (Function and arguments objects)
35213. maps to finalizer 2nd argument
35214. DUK_USE_AVOID_PLATFORM_FUNCPTRS
35215. 'roundpos' is relative to nc_ctx->k and increases to the right * (opposite of how 'k' changes).
35216. key encountered as a plain property
35217. e.g. DUK_OP_POSTINCV
35218. Allow empty string to be interpreted as 0
35219. Math.pow(-0,y) where y<0 should be: * - -Infinity if y<0 and an odd integer * - Infinity otherwise * NetBSD pow() returns -Infinity for all finite y<0. The * if-clause also catches y == -Infinity (which works even * without the fix).
35220. **comment:** XXX: add temporary duk_p pointer here too; sharing
label: code-design
35221. not minimum exponent
35222. * Avoid nested calls. Concretely this happens during debugging, e.g. * when we eval() an expression. ** Also don't interrupt if we're currently doing debug processing * (which can be initiated outside the bytecode executor) as this * may cause the debugger to be called recursively. Check required * for correct operation of throw intercept and other "exotic" halting * scenarios.
35223. Because new_size > duk_count_used_probe(heap), guaranteed to work
35224. bump refcount to prevent refzero during finalizer processing
35225. * Pointers to function data area for faster access. Function * data is a buffer shared between all closures of the same * "template" function. The data buffer is always fixed (non- * dynamic, hence stable), with a layout as follows: * * constants (duk_tval) * inner functions (duk_hobject *) * bytecode (duk_instr_t) * * Note: bytecode end address can be computed from 'data' buffer * size. It is not strictly necessary functionally, assuming * bytecode never jumps outside its allocated area. However, * it's a safety/robustness feature for avoiding the chance of * executing random data as bytecode due to a compiler error. ** Note: values in the data buffer must be incref'd (they will * be decref'd on release) for every compiledfunction referring * to the 'data' element.
35226. **comment:** * Advance lookup window by N characters, filling in new characters as * necessary. After returning caller is guaranteed a character window of * at least DUK_LEXER_WINDOW_SIZE characters. ** The main function duk_advance_bytes() is called at least once per every * token so it has a major lexer/compiler performance impact. There are two * variants for the main duk_advance_bytes() algorithm: a sliding window * approach which is slightly faster at the cost of larger code footprint, * and a simple copying one. ** Decoding directly from the source string would be another lexing option. * But the lookup window based approach has the advantage of hiding the * source string and its encoding effectively which gives more flexibility * going forward to e.g. support chunked streaming of source from flash. ** Decodes UTF-8/CESU-8 leniently with support for code points from U+0000 to * U+10FFFF, causing an error if the input is unparseable. Leniency means: * * * Unicode code point validation is intentionally not performed, * except to check that the codepoint does not exceed 0x10ffff. ** In particular, surrogate pairs are allowed and not combined, which * allows source files to represent all SourceCharacters with CESU-8. * Broken surrogate pairs are allowed, as Ecmascript does not mandate * their validation. *** Allows non-shortest UTF-8 encodings. ** Leniency here causes few security concerns because all character data is * decoded into Unicode codepoints before lexer processing, and is then * re-encoded into CESU-8. The source can be parsed as strict UTF-8 with * a compiler option. However, Ecmascript source characters include -all- * 16-bit unsigned integer codepoints, so leniency seems to be appropriate. * Note that codepoints above the BMP are not strictly SourceCharacters, * but the lexer still accepts them as such. Before ending up in a string * or an identifier name, codepoints above BMP are converted into surrogate * pairs and then CESU-8 encoded, resulting in 16-bit Unicode data as * expected by Ecmascript. * An alternative approach to dealing with invalid or partial sequences * would be to skip them and replace them with e.g. the Unicode replacement * character U+FFFD. This has limited utility because a replacement character * will most likely cause a parse error, unless it occurs inside a string. * Further, Ecmascript source is typically pure ASCII. ** See: * * <http://en.wikipedia.org/wiki=UTF-8> * <http://en.wikipedia.org/wiki/CESU-8> * <http://tools.ietf.org/html/rfc3629> * http://en.wikipedia.org/wiki/UTF-8#Invalid_byte_sequences ** Future work: * * * Reject other invalid Unicode sequences (see Wikipedia entry for examples) * in strict UTF-8 mode. ** * Size optimize. An attempt to use a 16-byte lookup table for the first * byte resulted in a code increase though. ** * Is checking against maximum 0x10ffff really useful? 4-byte encoding * imposes a certain limit anyway. *** Support chunked streaming of source code. Can be implemented either * by streaming chunks of bytes or chunks of codepoints.
label: code-design
35227. duk_push_this() + CheckObjectCoercible() + duk_to_string()
35228. pointers for scanning
35229. array part size (entirely gc reachable)
35230. Steps 12 and 13: reorganize elements to make room for itemCount elements
35231. number of tokens parsed
35232. Tear down state.
35233. Don't check for Infinity unless the context allows it. * 'Infinity' is a valid integer literal in e.g. base-36: * * parseInt('Infinity', 36) * 1461559270678
35234. zero handled by caller
35235. Don't accept relative indices now.
35236. [holder name val] -> [holder]
35237. res_obj
35238. * XUTF-8 and CESU-8 encoding/decoding
35239. 1110 xxxx 10xx xxxx 10xx xxxx [16 bits]
35240. function declaration
35241. **comment:** XXX: disable error handlers for duration of compaction?
label: code-design
35242. * PUTVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is PutValue'd] * 8.7.2 PutValue (V,W) [see especially step 3.b, undefined -> automatic global in non-strict mode] * 8.12.4 [[CanPut]] (P) * 8.12.5 [[Put]] (P) * * Note: may invalidate any valstack (or object) duk_tval pointers because * putting a value may reallocate any object or any valstack. Caller beware.
35243. ?
35244. state == 3
35245. Maximum traversal depth for "bound function" chains.
35246. **comment:** XXX: magic for getter/setter? use duk_def_prop()?
label: code-design
35247. xxx -> DUK_HBUFFEROBJECT_ELEM_INT32
35248. [thread value]
35249. * GETPROP: Ecmascript property read.
35250. **comment:** * The __FILE__ and __LINE__ information is intentionally not used in the * creation of the error object, as it isn't useful in the tracedata. The * tracedata still contains the function which returned the negative return * code, and having the file/line of this function isn't very useful.
label: code-design
35251. object is a thread (duk_hthread)
35252. 'info'
35253. add a new pending split to the beginning of the entire disjunction
35254. duk_push_this() + CheckObjectCoercible() + duk_to_object()
35255. [... arr jsonx(arr) res] -> [... res jsonx(arr)]
35256. this may have side effects, so re-lookup act
35257. Fall through to handle the rest.
35258. [... error lineNumber fileName]
35259. * Number built-ins
35260. **comment:** XXX: this compiles to over 500 bytes now, even without special handling * for an array part. Uses signed ints so does not handle full array range correctly.
label: code-design
35261. -> [... closure template env]
35262. Note: if the reg_temp load generated shuffling * instructions, we may need to rewind more than * one instruction, so use explicit PC computation.
35263. 6: toUTCString

35264. **comment:** note: ctx-wide temporary
label: code-design

35265. value popped by call

35266. Note: this may fail, caller should protect the call if necessary

35267. Leaves new timevalue on stack top and returns 1, which is correct * for part setters.

35268. stack[0] = compareFn * stack[1] = ToObject(this) * stack[2] = ToUint32(length)

35269. 4

35270. Boolean/any -> coerce boolean to number and try again. If boolean is * compared to a pointer, the final comparison after coercion now always * yields false (as pointer vs. number compares to false), but this is * not special cased.

35271. s <- 2 * b

35272. native functions: 149

35273. * Range parsing is done with a special lexer function which calls * us for every range parsed. This is different from how rest of * the parsing works, but avoids a heavy, arbitrary size intermediate * value type to hold the ranges. * * Another complication is the handling of character ranges when * case insensitive matching is used (see docs for discussion). * The range handler callback given to the lexer takes care of this * as well. * * Note that duplicate ranges are not eliminated when parsing character * classes, so that canonicalization of * * [0-9a-fA-Fx-{}] * * creates the result (note the duplicate ranges): * * [0-9A-FA-FX-Z{-{}}] * * where [x-{ }] is split as a result of canonicalization. The duplicate * ranges are not a semantics issue: they work correctly.

35274. Although we can allow non-BMP characters (they'll decode * back into surrogate pairs), we don't allow extended UTF-8 * characters; they would encode to URIs which won't decode * back because of strict UTF-8 checks in URI decoding. * (However, we could just as well allow them here.)

35275. exponent 1070

35276. no need to reset temps, as we're finished emitting code

35277. DUK_ERR_ERROR: no macros needed

35278. **comment:** XXX: assumed to fit for now
label: code-design

35279. [... fallback retval]

35280. **comment:** XXX: Migrate bufwriter and other read/write helpers to its own header?
label: code-design

35281. recursive call for a primitive value (guaranteed not to cause second * recursion).

35282. Getter might have arbitrary side effects, * so bail out.

35283. * The traceback format is pretty arcane in an attempt to keep it compact * and cheap to create. It may change arbitrarily from version to version. * It should be decoded/accessed through version specific accessors only. * * See doc/error-objects.rst.

35284. The internal _Target property is kept pointing to the original * enumeration target (the proxy object), so that the enumerator * 'next' operation can read property values if so requested. The * fact that the _Target is a proxy disables key existence check * during enumeration.

35285. Reject a proxy object as the handler because it would cause * potentially unbounded recursion. (ES6 has no such restriction)

35286. absolute req_digits; e.g. digits = 1 -> last digit is 0, * but add an extra digit for rounding.

35287. -> [... voidp voidp]

35288. convert duk_compiler_func into a function template, leaving the result * on top of stack.

35289. r <- r (updated above: r <- (remainder (* r B) s) * s <- s * m+ <- m+ (updated above: m+ <- (* m+ B) * m- <- m- (updated above: m- <- (* m- B) * B, low_ok, high_ok are fixed

35290. special helper for emitting u16 lists (used for character ranges for built-in char classes)

35291. [... re_obj input bc saved_buf]

35292. 'reg_varbind' is the operation result and can also * become the expression value for top level assignments * such as: "var x; x += y;".

35293. **comment:** * Three possible outcomes: * * A try or finally catcher is found => resume there. * (or) * * The error propagates to the bytecode executor entry * level (and we're in the entry thread) => rethrow * with a new longjmp(), after restoring the previous * catchpoint. * * The error is not caught in the current thread, so * the thread finishes with an error. This works like * a yielded error, except that the thread is finished * and can no longer be resumed. (There is always a * resumer in this case.) * * Note: until we hit the entry level, there can only be * Ecmascript activations.
label: code-design

35294. * Number-to-string and string-to-number conversions. * * Slow path number-to-string and string-to-number conversion is based on * a Dragon4 variant, with fast paths for small integers. Big integer * arithmetic is needed for guaranteeing that the conversion is correct * and uses a minimum number of digits. The big number arithmetic has a * fixed maximum size and does not require dynamic allocations. * * See: doc/number-conversion.rst.

35295. Parser separator indices.

35296. Regexp

35297. k > 0 -> k was too low, and cannot be too high

35298. 'func' in the algorithm

35299. **comment:** * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written. * * Note: must re-lookup because calls above (e.g. duk_alloc_entry_checked()) * may realloc and compact properties and hence change e_idx.
label: code-design

35300. Outside any activation -> look up from global.

35301. Decrease by 25% every round

35302. For manual testing only.

35303. **comment:** Execute the fast path in a protected call. If any error is thrown, * fall back to the slow path. This includes e.g. recursion limit * because the fast path has a smaller recursion limit (and simpler, * limited loop detection).
label: code-design

35304. * Main heap structure

35305. * Argument exotic [[Delete]] behavior (ES6 Section 10.6) is * a post-check, keeping arguments internal 'map' in sync with * any successful deletes (note that property does not need to * exist for delete to 'succeed'). * * Delete key from 'map'. Since 'map' only contains array index * keys, we can use arr_idx for a fast skip.

35306. ES6 Section 9.2

35307. Backtrack utf-8 input and return a (possibly canonicalized) input character.

35308. Cause an interrupt after executing one instruction.

35309. Compiler

35310. at least errobj must be on stack

35311. 16 bits

35312. * Default panic handler

35313. thr->heap->dbg_detaching may be != 0 if a debugger write outside * the message loop caused a transport error and detach1() to run.

35314. 'modSearch'

35315. **comment:** The condition could be more narrow and check for the * copy area only, but there's no need for fine grained * behavior when the underlying buffer is misconfigured.
label: code-design

35316. **comment:** XXX: for a memory-code tradeoff, remove 'func' and make it's access either a function * or a macro. This would make the activation 32 bytes long on 32-bit platforms again.
label: code-design

35317. Function name is an Identifier (not IdentifierName), but we get * the raw name (not recognizing keywords) here and perform the name * checks only after pass 1.

35318. does not fit into 16 bits

35319. ensures no overflow

35320. [... input buffer]

35321. execution resumes in bytecode executor

35322. Standard JSON omits functions

35323. 7 bits

35324. * Value stack resizing. ** This resizing happens above the current "top": the value stack can be * grown or shrunk, but the "top" is not affected. The value stack cannot * be resized to a size below the current "top". ** The low level reallocation primitive must carefully recompute all value * stack pointers, and must also work if ALL pointers are NULL. The resize * is quite tricky because the valstack realloc may cause a mark-and-sweep, * which may run finalizers. Running finalizers may resize the valstack * recursively (the same value stack we're working on). So, after realloc * returns, we know that the valstack "top" should still be the same (there * should not be live values above the "top"), but its underlying size and * pointer may have changed.

35325. * Objects have internal references. Must finalize through * the "refzero" work list.

35326. different thread

35327. number of values in current MPUTARR set

35328. steps 13-14

35329. Compute result length and validate argument buffers.

35330. DUK_BUILTINS_H_INCLUDED

35331. copy bytecode instructions one at a time

35332. **comment:** E5.1 Section 15.2.4.6, step 3.a, lookup proto once before compare. * Prototype loops should cause an error to be thrown.
label: code-design

35333. [... RegExp val] -> [... res]

35334. load factor max 75%

35335. [value offset noAssert], when ftype != DUK_FLD_VARINT

35336. * Resizing

35337. token closes expression, e.g. ')', ']'

35338. fileName

35339. * Get prototype object for an integer error code.

35340. '\xffNext'

35341. PC/line semantics here are: * - For callstack top we're conceptually between two * opcodes and current PC indicates next line to * execute, so report that (matches Status). * - For other activations we're conceptually still * executing the instruction at PC-1, so report that * (matches error stacktrace behavior). * - See: <https://github.com/svaara/duktape/issues/281>

35342. Don't allow negative zero as it will cause trouble with * LDINT+LDINTX. But positive zero is OK.

35343. borrow literal Infinity from builtin string

35344. **comment:** XXX: This now coerces an identifier into a GETVAR to a temp, which * causes an extra LDREG in call setup. It's sufficient to coerce to a * unary ivalue?
label: code-design

35345. prefer direct execution

35346. **comment:** Longjmp state is cleaned up by error handling
label: code-design

35347. **comment:** spaces[] must be static to allow initializer with old compilers like BCC
label: code-design

35348. * Object.prototype.hasOwnProperty() and Object.prototype.propertyIsEnumerable().

35349. Ordinary gap, undefined encodes to 'null' in * standard JSON (and no JX/JC support here now).

35350. fall-through jump to next code of next case (backpatched)

35351. already one-based

35352. **comment:** XXX: could use at least one fewer loop counters
label: code-design

35353. Wrap checked above.

35354. Create function 'data' buffer but don't attach it yet.

35355. * Function built-ins

35356. * No match, E5 Section 15.10.6.2, step 9.a.i - 9.a.ii apply, regardless * of 'global' flag of the RegExp. In particular, if lastIndex is invalid * initially, it is reset to zero.

35357. **comment:** hacky helper for String.prototype.split()
label: code-design

35358. [(builtin objects) name func]

35359. We request a tail call, but in some corner cases * call handling can decide that a tail call is * actually not possible. * See: test-bug-tailcall-preventyield-assert.c.

35360. toString() conditions

35361. Write bytecode executor's curr_pc back to topmost activation (if any).

35362. [] -> [val]

35363. **comment:** XXX: Extensibility check for target uses IsExtensible(). If we * implemented the isExtensible trap and didn't reject proxies as * proxy targets, it should be respected here.
label: code-design

35364. **comment:** Technically return value is not needed because INVLHS will * unconditionally throw a ReferenceError. Coercion is necessary * for proper semantics (consider ToNumber() called for an object). * Use DUK_EXTRAOP_UNP with a dummy register to get ToNumber().
label: code-design

35365. decl type

35366. -> [sep TObject(this) len sep str]

35367. Shared lenient buffer length clamping helper. Indices are treated as * element indices (though output values are byte offsets) which only * really matters for TypedArray views as other buffer object have a zero * shift. Negative indices are counted from end of input slice; crossed * indices are clamped to zero length; and final indices are clamped * against input slice. Used for e.g. ArrayBuffer slice().

35368. XXX: perhaps refactor this to allow caller to specify some parameters, or * at least a 'compact' flag which skips any spare or round-up .. useful for * emergency gc.

35369. [... enum_target res handler trap]

35370. XXX: Decrementing and restoring act->curr_pc works now, but if the * debugger message loop gains the ability to adjust the current PC * (e.g. a forced jump) restoring the PC here will break. Another * approach would be to use a state flag for the "decrement 1 from * topmost activation's PC" and take it into account whenever dealing * with PC values.

35371. Create a fresh object environment for the global scope. This is * needed so that the global scope points to the newly created RAM-based * global object.

35372. x <- (1<<y)

35373. * HASPROP: Ecmascript property existence check ("in" operator). ** Interestingly, the 'in' operator does not do any coercion of * the target object.

35374. Any 'res' will do.

35375. If neutered must return 0; length is zeroed during * neutering.

35376. XXX: assert idx_base

35377. **comment:** XXX: refactor into internal helper, duk_clone_hobject()
label: code-design

35378. * Get starting character offset for match, and initialize 'sp' based on it. ** Note: lastIndex is non-configurable so it must be present (we check the * internal class of the object above, so we know it is). User code can set * its value to an arbitrary (garbage) value though; E5 requires that lastIndex * be coerced to a number before using. The code below works even if the * property is missing: the value will then be coerced to zero. ** Note: lastIndex may be outside Uint32 range even after ToInteger() coercion. * For instance, ToInteger(+Infinity) = +Infinity. We track the match offset * as an integer, but pre-check it to be inside the 32-bit range before the loop. * If not, the check in E5 Section 15.10.6.2, step 9.a applies.

35379. avoid side effect issues

35380. * Setup call: target and 'this' binding. Three cases: * * 1. Identifier base (e.g. "foo()") * 2. Property base (e.g. "foo.bar()") * 3. Register base (e.g. "foo()"); i.e. when a return value is a function

35381. SHIFT EXPRESSION

35382. overflow, fall through

35383. restored if ecma-to-ecma setup fails

35384. Match labels starting from latest label because there can be duplicate empty * labels in the label set.
35385. If debugger is paused, garbage collection is disabled by default.
35386. DUK_ERR_REFERENCE_ERROR: no macros needed
35387. XXX: shared helper for duk_push_hobject_or_undefined()
35388. * Strings have no internal references but do have "weak" * references in the string cache. Also note that strings * are not on the heap_allocated list like other heap * elements.
35389. unconditionally
35390. **comment:** XXX: At the moment Duktape accesses internal keys like _Finalizer using a * normal property set/get which would allow a proxy handler to interfere with * such behavior and to get access to internal key strings. This is not a problem * as such because internal key strings can be created in other ways too (e.g. * through buffers). The best fix is to change Duktape internal lookups to * skip proxy behavior. Until that, internal property accesses bypass the * proxy and are applied to the target (as if the handler did not exist). * This has some side effects, see test-bi-proxy-internal-keys.js.
label: code-design
35391. Get local time offset (in seconds) for a certain (UTC) instant 'd'.
35392. **comment:** Can be called multiple times with no harm. Mark the transport * bad (dbg_read_cb == NULL) and clear state except for the detached * callback and the udata field. The detached callback is delayed * to the message loop so that it can be called between messages; * this avoids corner cases related to immediate debugger reattach * inside the detached callback.
label: code-design
35393. resume
35394. * Parse a function body or a function-like expression, depending * on flags.
35395. actually used, non-NULL entries
35396. Fast path check.
35397. length check
35398. Do decrefs only with safe pointers to avoid side effects * disturbing e_idx.
35399. '{"_ninf":true}'
35400. shared checks for all descriptor types
35401. There's no need to check for buffer validity status for the * target here: the property access code will do that for each * element. Moreover, if we did check the validity here, side * effects from reading the source argument might invalidate * the results anyway.
35402. * Process space (3rd argument to JSON.stringify)
35403. Difference to non-strict/strict comparison is that NaNs compare * equal and signed zero signs matter.
35404. DUK_USE_PANIC_HANDLER
35405. avoid array abandoning which interns strings
35406. traits are separate; in particular, arguments not an array
35407. duk_get_min_grow_a() is always >= 1
35408. 9: getUTCFullYear
35409. [key getter this key] -> [key retval]
35410. Source end clamped silently to available length.
35411. First character has already been eaten and checked by the caller. * We can scan until a NUL in stridx string because no built-in strings * have internal NULs.
35412. * Proxy built-in (ES6)
35413. Native function, no relevant lineNumber.
35414. * Limited functionality bigint implementation. * * Restricted to non-negative numbers with less than 32 * DUK__BI_MAX_PARTS bits, * with the caller responsible for ensuring this is never exceeded. No memory * allocation (except stack) is needed for bigint computation. Operations * have been tailored for number conversion needs. * * Argument order is "assignment order", i.e. target first, then arguments: * x <- y * z --> duk__bi_mul(x, y, z);
35415. * InitJS code - Ecmascript code evaluated from a built-in source * which provides e.g. backward compatibility. User can also provide * JS code to be evaluated at startup.
35416. 'res' must be a plain ivalue, and not register-bound variable.
35417. function helpers
35418. 'Null'
35419. internal properties
35420. custom
35421. * XXX: shrink array allocation or entries compaction here?
35422. ToInteger(lastIndex)
35423. * Encoding/decoding helpers
35424. default: NaN
35425. covers -Infinity
35426. Error instance, use augmented error data directly
35427. t1 = (+ r m+)
35428. [... func arg1 ... argN]
35429. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small; some overlap with string * handling.
label: code-design
35430. Some contexts don't allow fractions at all; this can't be a * post-check because the state ('f' and expt) would be incorrect.
35431. variable value (function, we hope, not checked here)
35432. return 'res_obj'
35433. E5 Section 10.4.2
35434. DUK_FORWDECL_H_INCLUDED
35435. avoid unary minus on unsigned
35436. Just flip the single bit.
35437. unicode code points, window[0] is always next, points to 'buffer'
35438. probe
35439. may be set to 0 by duk_debugger_attach() inside callback
35440. Easy to get wrong, so assert for it.
35441. mask out flags not actually stored
35442. 'with' target must be created first, in case we run out of memory
35443. **comment:** Slow writing of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. There's * no special sign handling when writing varints.
label: code-design
35444. Important main primitive.
35445. no throw
35446. [... lval rval(func)]
35447. **comment:** XXX: 'this' will be ToObject() coerced twice, which is incorrect * but should have no visible side effects.
label: code-design
35448. **comment:** XXX: The duk_to_number() cast followed by integer coercion * is platform specific so NaN, +/- Infinity, and out-of-bounds * values result in platform specific output now. * See: test-bi-nodejs-buffer-proto-varint-special.js
label: code-design
35449. 'lastIndex'
35450. 0xd0-0xdf
35451. * Format tag parsing. Since we don't understand all the * possible format tags allowed, we just scan for a terminating * specifier and keep track of relevant modifiers that we do * understand. See man 3 printf.
35452. pop varname

35453. **comment:** XXX: this should probably just operate on the stack top, because it * needs to push stuff on the stack anyway...

label: code-design

35454. 'advtok' indicates how much to advance and which token id to assign * at the end. This shared functionality minimizes code size. All * code paths are required to set 'advtok' to some value, so no default * init value is used. Code paths calling DUK_ERROR() never return so * they don't need to set advtok.

35455. debugger

35456. * Automatically generated by extract_chars.py, do not edit!

35457. * Copy keys and values in the entry part (compacting them at the same time).

35458. expensive init, just want to fill window

35459. 'toLogString'

35460. split2: prefer jump execution (not direct)

35461. Zero special case: fake requested number of zero digits; ensure * no sign bit is printed. Relative and absolute fixed format * require separate handling.

35462. skip line info

35463. heap destruction ongoing, finalizer rescue no longer possible

35464. E5 Section 9.1

35465. value stack indices for tracking objects

35466. two encoding attempts suffices

35467. DUK_ISPEC_XXX

35468. **comment:** XXX: append primitive

label: code-design

35469. current lexical environment (may be NULL if delayed)

35470. explicit NULL inits

35471. [... closure template env]

35472. use a temp: decref only when valstack reachable values are correct

35473. Helper for string conversion calls: check 'this' binding, get the * internal time value, and format date and/or time in a few formats. * Return value allows tail calls.

35474. 0 or -1

35475. **comment:** * Allocate heap struct * * Use a raw call, all macros expect the heap to be initialized

label: code-design

35476. Object.defineProperty() equivalent C binding.

35477. skip jump slots

35478. We can directly access value stack here.

35479. DUK_TOK_TYPEOF

35480. mark as complex

35481. XXX: handling of timestamps outside Windows supported range. * How does Windows deal with dates before 1600? Does windows * support all Ecmascript years (like -200000 and +200000)? * Should equivalent year mapping be used here too? If so, use * a shared helper (currently integrated into timeval-to-parts).

35482. **comment:** m+ != m- (very rarely)

label: code-design

35483. * Duktape debugger

35484. Ecmascript activation + Duktape.Thread.resume() activation

35485. \"

35486. [... str]

35487. -> [... value]

35488. Use temporaries and update lex_ctx only when finished.

35489. mark-and-sweep not running -> must be empty

35490. **comment:** * Normal error * * Normal error is thrown with a longjmp() through the current setjmp() * catchpoint record in the duk_heap. The 'curr_thread' of the duk_heap * identifies the throwing thread. * * Error formatting is usually unnecessary. The error macros provide a * zero argument version (no formatting) and separate macros for small * argument counts. Variadic macros are not used to avoid portability * issues and avoid the need for stash-based workarounds when they're not * available. Vararg calls are avoided for non-formatted error calls * because vararg call sites are larger than normal, and there are a lot * of call sites with no formatting. * * Note that special formatting provided by debug macros is NOT available. * * The _RAW variants allow the caller to specify file and line. This makes * it easier to write checked calls which want to use the call site of the * checked function, not the error macro call inside the checked function.

label: code-design

35491. **comment:** Fast canonicalization lookup at the cost of 128kB footprint.

label: code-design

35492. DUK_TOK_EXPORT

35493. array of active label names

35494. * Helper for C function call negative return values.

35495. entry_top + 0

35496. * Thread switching * * To switch heap->curr_thread, use the macro below so that interrupt counters * get updated correctly. The macro allows a NULL target thread because that * happens e.g. in call handling.

35497. entry allocation updates hash part and increases the key * refcount; may need a props allocation resize but doesn't * 'recheck' the valstack.

35498. Not necessary to unwind catchstack: break/continue * handling will do it. The finally flag of 'cat' is * no longer set. The catch flag may be set, but it's * not checked by break/continue handling.

35499. Shared helper.

35500. x1 must be a string

35501. [... value? getter? setter?]

35502. **comment:** XXX: the insert here is a bit expensive if there are a lot of items. * It could also be special cased in the outermost for loop quite easily * (as the element is dup()'d anyway).

label: code-design

35503. Gather in big endian

35504. **comment:** SCANBUILD: warning about 'thr' potentially being NULL here, * warning is incorrect because thr != NULL always here.

label: code-design

35505. format is bit packed

35506. See test-bug-netbsd-math-pow.js: NetBSD 6.0 on x86 (at least) does not * correctly handle some cases where x=+-0. Specific fixes to these * here.

35507. parse args starting from "next temp", reg_target + 1

35508. Note: multiple threads may be simultaneously in the RUNNING * state, but not in the same "resume chain".

35509. * Named function expression, name needs to be bound * in an intermediate environment record. The "outer" * lexical/variable environment will thus be: * * a) { funcname: <func>, __prototype: outer_lex_env } * b) { funcname: <func>, __prototype: <globalenv> } (if outer_lex_env missing)

35510. * x is finite and non-zero * * -1.6 -> floor(-1.1) -> -2 * -1.5 -> floor(-1.0) -> -1 (towards +Inf) * -1.4 -> floor(-0.9) -> -1 * -0.5 -> -0.0 (special case) * -0.1 -> -0.0 (special case) * +0.1 -> +0.0 (special case) * +0.5 -> floor(+1.0) -> 1 (towards +Inf) * +1.4 -> floor(+1.9) -> 1 * +1.5 -> floor(+2.0) -> 2 (towards +Inf) * +1.6 -> floor(+2.1) -> 2

35511. * Context init

35512. **comment:** XXX: rework

label: code-design

35513. Call stack. [0,callstack_top[is GC reachable.

35514. DUK_NUMCONV_H_INCLUDED

35515. is a function declaration (as opposed to function expression)

35516. Precomputed pointers when using 16-bit heap pointer packing.

35517. The E5.1 Section 15.4.4 algorithm doesn't set the length explicitly * in the end, but because we're operating with an internal value which * is known to be an array, this should be equivalent.

35518. XXX: Allow customizing the pause and notify behavior at runtime * using debugger runtime flags. For now the behavior is fixed using * config options.
35519. = IsAccessorDescriptor(Desc)
35520. 36: setFullYear
35521. Object is currently being finalized.
35522. **comment:** not covered, return all zeroes
label: test

35523. * Unicode range matcher * * Matches a codepoint against a packed bitstream of character ranges. * Used for slow path Unicode matching.
35524. DUK_HTHREAD_H_INCLUDED
35525. DUK_USE_ES6_PROXY
35526. Located in duk_heap.hdr h_extra16. Subclasses of duk_hobject (like * duk_hcompiledfunction) are not free to use h_extra16 for this reason.
35527. When looking for .fileName/.lineNumber, blame compilation * or C call site unless flagged not to do so.
35528. Pretend like we got EOM
35529. [regexp source bytecode]
35530. catch depth at point of definition
35531. -> [... cons target]
35532. Grow entry part allocation for one additional entry.
35533. **comment:** * Calls. * * Protected variants should avoid ever throwing an error.
label: code-design
35534. **comment:** XXX: better multiline
label: code-design
35535. object literal key tracking flags
35536. curr is accessor -> cannot be in array part
35537. stmt has explicit/implicit semicolon terminator
35538. lexing
35539. [... result/error]
35540. catches EOF (NUL) and initial comma
35541. **comment:** If no explicit message given, put error code into message field * (as a number). This is not fully in keeping with the EcmaScript * error model because messages are supposed to be strings (Error * constructors use ToString() on their argument). However, it's * probably more useful than having a separate 'code' property.
label: code-design
35542. The #ifdef clutter here needs to handle the three cases: * (1) JX+JC, (2) JX only, (3) JC only.
35543. currently running thread
35544. temproots
35545. Tail call check: if last opcode emitted was CALL(I), and * the context allows it, change the CALL(I) to a tail call. * This doesn't guarantee that a tail call will be allowed at * runtime, so the RETURN must still be emitted. (Duktape * 0.10.0 avoided this and simulated a RETURN if a tail call * couldn't be used at runtime; but this didn't work * correctly with a thread yield/resume, see * test-bug-tailcall-thread-yield-resume.js for discussion.) * * In addition to the last opcode being CALL, we also need to * be sure that 'rc_val' is the result register of the CALL(I). * For instance, for the expression 'return 0, (function () { return 1; })', 2' the last opcode emitted is CALL (no * bytecode is emitted for '2') but 'rc_val' indicates * constant '2'. Similarly if '2' is replaced by a register * bound variable, no opcodes are emitted but tail call would * be incorrect. * * This is tricky and easy to get wrong. It would be best to * track enough expression metadata to check that 'rc_val' came * from that last CALL instruction. We don't have that metadata * now, so we check that 'rc_val' is a temporary register result * (not a constant or a register bound variable). There should * be no way currently for 'rc_val' to be a temporary for an * expression following the CALL instruction without emitting * some opcodes following the CALL. This proxy check is used * below. * * See: test-bug-comma-expr-gh131.js. * * The non-standard 'caller' property disables tail calls * because they pose some special cases which haven't been * fixed yet.
35546. * Helper: handle Array object 'length' write which automatically * deletes properties, see E5 Section 15.4.5.1, step 3. This is * quite tricky to get right. * * Used by duk_hobject_putprop().
35547. vararg
35548. Use 'res' as the expression value (it's side effect * free and may be a plain value, a register, or a * constant) and write it to the LHS binding too.
35549. Allow NULL 'msg'
35550. borrowed reference
35551. [... s1 s2] -> [... s1+s2]
35552. DUK_TOK_DIV
35553. **comment:** XXX: lithuanian not implemented
label: requirement
35554. Explicit zero size check to avoid NULL 'tv1'.
35555. * Get holder object
35556. XXX: can shift() / unshift() use the same helper? * shift() is (close to?) <-> splice(0, 1) * unshift is (close to?) <-> splice(0, 0, [items])?
35557. lead zero + 'digits' fractions + 1 for rounding
35558. stack discipline consistency check
35559. * Update the interrupt counter
35560. these non-standard properties are copied for convenience
35561. still reachable
35562. * Forward declarations for all Duktape structures.
35563. [... obj key]
35564. **comment:** * Object.preventExtensions() and Object.isExtensible() (E5 Sections 15.2.3.10, 15.2.3.13) * * Not needed, implemented by macros DUK_HOBJECT_{HAS,CLEAR,SET}_EXTENSIBLE * and the Object built-in bindings.
label: code-design
35565. * Macros for property handling
35566. val2 = min count, val1 = max count
35567. find and remove string from stringtable; caller must free the string itself
35568. Inner functions recursively.
35569. == DUK__MAX_TEMPS is OK
35570. Don't free h->resumer because it exists in the heap. * Callstack entries also contain function pointers which * are not freed for the same reason.
35571. use bigint area as a temp
35572. 'n'
35573. * Allocate initial stacks for a thread. Note that 'thr' must be reachable * as a garbage collection may be triggered by the allocation attempts. * Returns zero (without leaking memory) if init fails.
35574. step 4.b
35575. (Number.MAX_VALUE).toString(2).length == 1024, + spare
35576. * Check whether already declared. * * We need to check whether the binding exists in the environment * without walking its parents. However, we still need to check * register-bound identifiers and the prototype chain of an object * environment target object.
35577. Catch stack. [0,catchstack_top[is GC reachable.
35578. DUK_USE_DATE_TZO_WINDOWS
35579. -> [... res_obj]
35580. * E5 Section 15.10.2.6. The previous and current character * should -not- be canonicalized as they are now. However, * canonicalization does not affect the result of IsWordChar() * (which depends on Unicode characters never canonicalizing * into ASCII characters) so this does not matter.
35581. flags is unsigned
35582. Already declared but no initializer value * (e.g. 'var xyz;'), no-op.
35583. 0

35584. **comment:** XXX: better place for this
label: code-design

35585. Zero encoded pointer is required to match NULL

35586. Given a (year, month, day-within-month) triple, compute day number. * The input triple is un-normalized and may contain non-finite values.

35587. upper limit, assuming no whitespace etc

35588. Copy slice, respecting underlying buffer limits; remainder * is left as zero.

35589. **comment:** not necessary to init, disabled for faster parsing
label: code-design

35590. We already ate 'x', so backup one byte.

35591. maximum token count before error (sanity backstop)

35592. * For ASCII strings, the answer is simple.

35593. 'super'

35594. **comment:** XXX: This won't be shown in practice now * because this code is not run when builtins * are in ROM.
label: code-design

35595. 'JSON'

35596. combine left->x1 and res->x1 (right->x1, really) -> (left->x1 OP res->x1)

35597. We can't reliably pop anything here because the stack input * shape is incorrect. So we throw an error; if the caller has * no catch point for this, a fatal error will occur. Another * alternative would be to just return an error. But then the * stack would be in an unknown state which might cause some * very hard to diagnose problems later on. Also note that even * if we did not throw an error here, the underlying call handler * might STILL throw an out-of-memory error or some other internal * fatal error.

35598. marker: copy t if not changed

35599. thread has terminated

35600. * Context management

35601. Return codes for protected calls (duk_safe_call(), duk_pcall())

35602. RangeError

35603. set/clear writable

35604. DUK_USE_64BIT_OPS

35605. Ecmascript date range is 100 million days from Epoch: * > 100e6 * 24 * 60 * 60 * 1000 // 100M days in millisecs * 8640000000000000 * (= 8.64e15)

35606. * Coercion operations: in-place coercion, return coerced value where * applicable. If index is invalid, throw error. Some coercions may * throw an expected error (e.g. from a toString() or valueOf() call) * or an internal error (e.g. from out of memory).

35607. packed tvl

35608. (internal) catch compilation errors

35609. * END PUBLIC API

35610. E == 0x7ff, topmost four bits of F != 0 => assume NaN

35611. APIError

35612. * BEGIN PUBLIC API

35613. timeval breakdown: internal time value NaN -> RangeError (toISOString)

35614. internal flag: external buffer

35615. enumerate internal properties (regardless of enumerability)

35616. Ecmascript E5 specification error codes

35617. * Debugger (debug protocol)

35618. lstring

35619. timeval breakdown: internal time value NaN -> zero

35620. include time part in string conversion result

35621. prefer number

35622. Note: parentheses are required so that the comma expression works in assignments.

35623. no error if file does not exist

35624. internal flag value: throw if mask doesn't match

35625. set getter (given on value stack)

35626. * Memory management * * Raw functions have no side effects (cannot trigger GC).

35627. Coercion hints

35628. is_copy

35629. * Variable access

35630. weekday: 0 to 6, 0=sunday, 1=monday, etc

35631. DUK_DBUNION_H_INCLUDED

35632. lightweight function pointer

35633. * String manipulation

35634. getter: subtract 1900 from year when getting year part

35635. Value mask types, used by e.g. duk_get_type_mask()

35636. Ecmascript undefined

35637. file

35638. Log levels

35639. Ecmascript boolean: 0 or 1

35640. * Public API specific typedefs * * Many types are wrapped by Duktape for portability to rare platforms * where e.g. 'int' is a 16-bit type. See practical typing discussion * in Duktape web documentation.

35641. flags

35642. Assertion Error

35643. only enumerate array indices

35644. * Stack management

35645. Flags for duk_def_prop() and its variants

35646. NOTE: when writing a Date provider you only need a few specific * flags from here, the rest are internal. Avoid using anything you * don't need.

35647. * Date provider related constants * * NOTE: These are "semi public" - you should only use these if you write * your own platform specific Date provider, see doc/datetime.rst.

35648. internal: request fixed buffer result

35649. Number of value stack entries (in addition to actual call arguments) * guaranteed to be allocated on entry to a Duktape/C function.

35650. * Indexes of various types with respect to big endian (logical) layout

35651. no error (e.g. from duk_get_error_code())

35652. LICENSE.txt

35653. SyntaxError

35654. * Duktape/C function magic value

35655. internal: request dynamic buffer result

35656. * Avoid C++ name mangling

35657. **comment:** XXX: replace with TypeError?
label: code-design

35658. **comment:** XXX: to be removed?
label: code-design

35659. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2017 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the

"Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

35660. Git commit, describe, and branch for Duktape build. Useful for * non-official snapshot builds so that application code can easily log * which Duktape snapshot was used. Not available in the Ecmascript * environment.

35661. DUK_API_PUBLIC_H_INCLUDED

35662. Compilation flags for duk_compile() and duk_eval()

35663. * Duktape public API for Duktape 1.8.0. * * See the API reference for documentation on call semantics. * The exposed API is inside the DUK_API_PUBLIC_H_INCLUDED * include guard. Other parts of the header are Duktape * internal and related to platform/compiler/feature detection. * * Git commit 0a70d7e4c5227c84e3fed5209828973117d02849 (v1.8.0). * Git branch v1.8-maintenance. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.

35664. timeval breakdown: convert month and day-of-month parts to one-based (default is zero-based)

35665. Duktape version, (major * 10000) + (minor * 100) + patch. Allows C code * to #ifdef against Duktape API version. The same value is also available * to Ecmascript code in Duktape.version. Unofficial development snapshots * have 99 for patch level (e.g. 0.10.99 would be a development version * after 0.10.0 but before the next official release).

35666. * Buffer

35667. set enumerable (effective if DUK_DEFPROP_HAVE_ENUMERABLE set)

35668. set configurable (effective if DUK_DEFPROP_HAVE_CONFIGURABLE set)

35669. don't walk prototype chain, only check own properties

35670. UnimplementedError

35671. (internal) omit eval result

35672. **comment:** * If no variadic macros, __FILE__ and __LINE__ are passed through globals * which is ugly and not thread safe.
label: requirement

35673. last value is func pointer, arguments follow in parens

35674. set/clear configurable

35675. Return codes for C functions (shortcut for throwing an error)

35676. * Pop operations

35677. * Require operations: no coercion, throw error if index or type * is incorrect. No defaulting.

35678. * Bytecode load/dump

35679. compile function code (instead of global code)

35680. enumerate a proxy object itself without invoking proxy behavior

35681. **comment:** XXX: native 64-bit byteswaps when available
label: code-design

35682. **comment:** XXX: These calls are incomplete and not usable now. They are not (yet) * part of the public API.
label: code-design

35683. Error

35684. DUK_USE_FILE_IO

35685. ReferenceError

35686. timeval breakdown: replace year with equivalent year in the [1971,2037] range for DST calculations

35687. * Ecmascript operators

35688. args

35689. * Some defines forwarded from feature detection

35690. (internal) take strlen() of src_buffer (avoids double evaluation in macro)

35691. set/clear enumerable

35692. internal use

35693. * Error handling

35694. EvalError

35695. * Object prototype

35696. setter: perform 2-digit year fixup (00...99 -> 1900...1999)

35697. nop: no need to normalize

35698. * Compilation and evaluation

35699. or is a normalized NaN

35700. AllocError

35701. month: 0 to 11

35702. **comment:** XXX: Enough space to hold internal suspend/resume structure. * This is rather awkward and to be fixed when the internal * structure is visible for the public API header.
label: code-design

35703. internal flag: create backing ArrayBuffer; keep in one byte

35704. additional values begin at bit 12

35705. * Constants

35706. **comment:** XXX: replace with plain Error?
label: code-design

35707. * Object finalizer

35708. Reverse operation is the same.

35709. duk_context is now defined in duk_config.h because it may also be * referenced there by prototypes.

35710. * C++ name mangling

35711. UncaughtError

35712. Ecmascript year range: * > new Date(100e6 * 24 * 3600e3).toISOString() * '+275760-09-13T00:00:00.000Z' * > new Date(-100e6 * 24 * 3600e3).toISOString() * '-271821-04-20T00:00:00.000Z'

35713. create a new global environment

35714. * Global object

35715. DUKTAPE_H_INCLUDED

35716. E == 0x7ff, F != 0 => NaN

35717. DUK_USE_PACKED_TVAL

35718. * Logging

35719. safe variants of a few coercion operations

35720. Indicates that a native function does not have a fixed number of args, * and the argument stack should not be capped/extended at all.

35721. * Get operations: no coercion, returns default value for invalid * indices and invalid value types. * * duk_get_undefined() and duk_get_null() would be pointless and * are not included.

35722. **comment:** use locale specific formatting if available
label: code-design

35723. External duk_config.h provides platform/compiler/OS dependent * typedefs and macros, and DUK_USE_xxx config options so that * the rest of Duktape doesn't need to do any feature detection.

35724. internal: don't care about fixed/dynamic nature

35725. * Property access ** The basic function assumes key is on stack. The _string variant takes * a C string as a property name, while the _index variant takes an array * index as a property name (e.g. 123 is equivalent to the key "123").

35726. * Debugging

35727. 64-bit byteswap, same operation independent of target endianness.

35728. * Type checks ** duk_is_none(), which would indicate whether index is outside of stack, * is not needed; duk_is_valid_index() gives the same information.

35729. sort array indices, use with DUK_ENUM_ARRAY_INDICES_ONLY

35730. force change if possible, may still fail for e.g. virtual properties

35731. Millisecond count constants.

35732. used by packed duk_tval, assumes sizeof(void *) == 4

35733. set value (given on value stack)

35734. Used to represent invalid index; if caller uses this without checking, * this index will map to a non-existent stack entry. Also used in some * API calls as a marker to denote "no value".

35735. * Module helpers: put multiple function or constant properties

35736. raw void pointer

35737. Flags for duk_push_thread_raw()

35738. * ===== * Duktape authors * ===== * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt''. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt'' (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang \u00f3z \u2022 <lango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6ttfert <<https://github.com/jaseg>> * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6man * * Doug Sanden * * Josh Engebreton (<https://github.com/JoshEngebreton>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstruchtrup> * * Michael Drake (<https://github.com/tlsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn\u00e5s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`)) and I'll fix the omission.

35739. InternalError

35740. Value types, used by e.g. duk_get_type()

35741. **comment:** Duktape specific error codes (must be 8 bits at most, see duk_error.h)

label: code-design

35742. no value, e.g. invalid index

35743. * Stack manipulation (other than push/pop)

35744. Byteswap an IEEE double in the duk_double_union from host to network * order. For a big endian target this is a no-op.

35745. prefer string

35746. external use

35747. Flags for duk_push_string_file_raw()

35748. Duktape debug protocol version used by this build.

35749. **comment:** DUK_COMPILE_xxx bits 0-2 are reserved for an internal 'nargs' argument * (the nargs value passed is direct stack arguments + 1 to account for an * internal extra argument).

label: code-design

35750. fixed or dynamic, garbage collected byte buffer

35751. * Thread management

35752. * Double NaN manipulation macros related to NaN normalization needed when * using the packed duk_tval representation. NaN normalization is necessary * to keep double values compatible with the duk_tval format. * * When packed duk_tval is used, the NaN space is used to store pointers * and other tagged values in addition to NaNs. Actual NaNs are normalized * to a specific quiet NaN. The macros below are used by the implementation * to check and normalize NaN values when they might be created. The macros * are essentially NOPs when the non-packed duk_tval representation is used. * * A FULL check is exact and checks all bits. A NOTFULL check is used by * the packed duk_tval and works correctly for all NaNs except those that * begin with 0x7ff0. Since the 'normalized NaN' values used with packed * duk_tval begin with 0x7ff8, the partial check is reliable when packed * duk_tval is used. The 0x7ff8 prefix means the normalized NaN will be a * quiet NaN regardless of its remaining lower bits. * * The ME variant below is specifically for ARM byte order, which has the * feature that while doubles have a mixed byte order (32107654), unsigned * long long values has a little endian byte order (76543210). When writing * a logical double value through a ULL pointer, the 32-bit words need to be * swapped; hence the #ifdefs below for ULL writes with DUK_USE_DOUBLE_ME. * This is not full ARM support but suffices for some environments.

35753. either not a NaN

35754. convert time value to local time

35755. not directly applicable, byte order differs from a double

35756. end 'extern "C"' wrapper

35757. * Object operations

35758. set writable (effective if DUK_DEFPROP_HAVE_WRITABLE set)

35759. UnsupportedError

35760. **comment:** Internal API call flags, used for various functions in this file. * Certain flags are used by only certain functions, but since the flags * don't overlap, a single flags value can be passed around to multiple * functions. * * The unused top bits of the flags field are also used to pass values * to helpers (duk_get_part_helper() and duk_set_part_helper()). * * (Must be in-sync with genbuiltins.py)

label: code-design

35761. There are currently no native functions to yield/resume, due to the internal * limitations on coroutine handling. These will be added later.

35762. Concrete macros for NaN handling used by the implementation internals. * Chosen so that they match the duk_tval representation: with a packed * duk_tval, ensure NaNs are properly normalized; with a non-packed duk_tval * these are essentially NOPs.

35763. (internal) no source string on stack

35764. * Function (method) calls

35765. plain

35766. ECMAScript string: CESU-8 / extended UTF-8 encoded

35767. internal flag: don't zero allocated buffer

35768. use strict (outer) context for global, eval, or function code

35769. in non-packed representation we don't care about which NaN is used

35770. * Union for accessing double parts, also serves as packed duk_tval

35771. include date part in string conversion result

35772. internal flag: dynamic buffer

35773. ECMAScript null

35774. enumerate non-enumerable properties in addition to enumerable

35775. **comment:** XXX: replace with RangeError?

label: code-design

35776. * Push operations * * Push functions return the absolute (relative to bottom of frame) * position of the pushed value for convenience. * * Note: duk_dup() is technically a push.

35777. string
35778. * ROM pointer compression
35779. * Union to access IEEE double memory representation, indexes for double * memory representation, and some macros for double manipulation. * * Also used by packed duk_tval. Use a union for bit manipulation to * minimize aliasing issues in practice. The C99 standard does not * guarantee that this should work, but it's a very widely supported * practice for low level manipulation. * * IEEE double format summary: * * seeeeeeee eeeeffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * See http://en.wikipedia.org/wiki/Double_precision_floating-point_format. * * NaNs are represented as exponent 0x7ff and mantissa != 0. The NaN is a * signaling NaN when the highest bit of the mantissa is zero, and a quiet * NaN when the highest bit is set. * * At least three memory layouts are relevant here: * * A B C D E F G H Big endian (e.g. 68k) DUK_USE_DOUBLE_BE * H G F E D C B A Little endian (e.g. x86) DUK_USE_DOUBLE_LE * D C B A H G F E Mixed/cross endian (e.g. ARM) DUK_USE_DOUBLE_ME * * ARM is a special case: ARM double values are in mixed/cross endian * format while ARM duk_uint64_t values are in standard little endian * format (H G F E D C B A). When a double is read as a duk_uint64_t * from memory, the register will contain the (logical) value * E F G H A B C D. This requires some special handling below. * * Indexes of various types (8-bit, 16-bit, 32-bit) in memory relative to * the logical (big endian) order: * * byte order duk_uint8_t duk_uint16_t duk_uint32_t * BE 01234567 0123 01 * LE 76543210 3210 10 * ME (ARM) 32107654 1032 01 * * Some processors may alter NaN values in a floating point load+store. * For instance, on X86 a FLD + FSTP may convert a signaling NaN to a * quiet one. This is catastrophic when NaN space is used in packed * duk_tval values. See: misc/clang_aliasing.c.

35780. * Helper macros for reading/writing memory representation parts, used * by duk_numconv.c and duk_tval.h.
35781. string conversion: use 'T' instead of '' as a separator
35782. Ecmascript number: double
35783. TypeError
35784. Support array for ROM pointer compression. Only declared when ROM * pointer compression is active.
35785. **comment:** Although extra/top could be an unsigned type here, using a signed type * makes the API more robust to calling code calculation errors or corner * cases (where caller might occasionally come up with negative values). * Negative values are treated as zero, which is better than casting them * to a large unsigned number. (This principle is used elsewhere in the * API too.)
label: code-design
35786. * Misc conversion
35787. Ecmascript object: includes objects, arrays, functions, threads
35788. URIError
35789. all doubles are considered normalized
35790. AUTHORS.rst
35791. day within month: 0 to 30
35792. setter: call is a time setter (affects hour, min, sec, ms); otherwise date setter (affects year, month, day-in-month)
35793. (internal) no filename on stack
35794. compile eval code (instead of global code)
35795. One problem with this macro is that expressions like the following fail * to compile: "(void) duk_error(...)". But because duk_error() is noreturn, * they make little sense anyway.
35796. set setter (given on value stack)
35797. prefer number, unless input is a Date, in which * case prefer string (E5 Section 8.12.8)
35798. Part indices for internal breakdowns. Part order from DUK_DATE_IDX_YEAR * to DUK_DATE_IDX_MILLISECOND matches argument ordering of Ecmascript API * calls (like Date constructor call). Some functions in duk_bi_date.c * depend on the specific ordering, so change with care. 16 bits are not * enough for all parts (year, specifically). * * (Must be in-sync with genbuiltins.py.)
35799. E == 0x7ff, F == 8 => normalized NaN
35800. Enumeration flags for duk_enum()
35801. year
35802. * Other state related functions
35803. DUK_USE_PROVIDE_DEFAULT_ALLOC_FUNCTIONS
35804. * Default allocation functions. * * Assumes behavior such as malloc allowing zero size, yielding * a NULL or a unique pointer which is a no-op for free.
35805. * Buffer
35806. maximum size check is handled by callee
35807. **comment:** Forget the previous allocation, setting size to 0 and alloc to * NULL. Caller is responsible for freeing the previous allocation. * Getting the allocation and clearing it is done in the same API * call to avoid any chance of a realloc.
label: code-design
35808. Load constants onto value stack but don't yet copy to buffer.
35809. -> [func funcname env]
35810. _Formals
35811. Setup function properties.
35812. Push function object, init flags etc. This must match * duk_js_push_closure() quite carefully.
35813. **comment:** * Bytecode dump/load * * The bytecode load primitive is more important performance-wise than the * dump primitive. * * Unlike most Duktape API calls, bytecode dump/load is not guaranteed to be * memory safe for invalid arguments - caller beware! There's little point * in trying to achieve memory safety unless bytecode instructions are also * validated which is not easy to do with indirect register references etc.
label: code-design
35814. _Varmap is dense
35815. We know _Formals is dense and all entries will be in the * array part. GC and finalizers shouldn't affect _Formals * so side effects should be fine.
35816. skip line info
35817. Bound functions don't have all properties so we'd either need to * lookup the non-bound target function or reject bound functions. * For now, bound functions are rejected.
35818. end of _Formals
35819. **comment:** XXX: There's some overlap with duk_js_closure() here, but * seems difficult to share code. Ensure that the final function * looks the same as created by duk_js_closure().
label: code-design
35820. The caller is responsible for being sure that bytecode being loaded * is valid and trusted. Invalid bytecode can cause memory unsafe * behavior directly during loading or later during bytecode execution * (instruction validation would be quite complex to implement). * * This signature check is the only sanity check for detecting * accidental invalid inputs. The initial 0xFF byte ensures no * ordinary string will be accepted by accident.
35821. buffer limits
35822. Inner functions recursively.
35823. known to be number; in fact an integer
35824. Object extra properties. * * There are some difference between function templates and functions. * For example, function templates don't have .length and nargs is * normally used to instantiate the functions.
35825. Bytecode instructions: endian conversion needed unless * platform is big endian.
35826. **comment:** Load a function from bytecode. The function object returned here must * match what is created by duk_js_push_closure() with respect to its flags, * properties, etc. * * NOTE: there are intentionally no input buffer length / bound checks. * Adding them would be easy but wouldn't ensure memory safety as untrusted * or broken bytecode is unsafe during execution unless the opcodes themselves * are validated (which is quite complex, especially for indirect opcodes).
label: code-design
35827. duk_hcompiledfunction flags; quite version specific
35828. standard prototype
35829. Load bytecode instructions.
35830. string limits
35831. no side effects

35832. We know _Varmap only has own properties so walk property * table directly. We also know _Varmap is dense and all * values are numbers; assert for these. GC and finalizers * shouldn't affect _Varmap so side effects should be fine.

35833. **comment:** known to be 32-bit
label: code-design

35834. Fixed header info.

35835. constants are strings or numbers now

35836. Create function 'data' buffer but don't attach it yet.

35837. With constants and inner functions on value stack, we can now * atomically finish the function 'data' buffer, bump refcounts, * etc. * * Here we take advantage of the value stack being just a duk_tval * array: we can just memcpy() the constants as long as we incref * them afterwards.

35838. Estimating the result size beforehand would be costly, so * start with a reasonable size and extend as needed.

35839. DUK_USE_BYTECODE_DUMP_SUPPORT

35840. ensures no overflow

35841. The function object is now reachable and refcounts are fine, * so we can pop off all the temporaries.

35842. _Varmap

35843. -> [func funcname env funcname func]

35844. XXX: This causes recursion up to inner function depth * which is normally not an issue, e.g. mark-and-sweep uses * a recursion limiter to avoid C stack issues. Avoiding * this would mean some sort of a work list or just refusing * to serialize deep functions.

35845. Constants: variable size encoding.

35846. Return with final function pushed on stack top.

35847. [...] -> [...] buf]

35848. [...] buf func] -> [...] func]

35849. Original function instance/template had NAMEBINDING. * Must create a lexical environment on loading to allow * recursive functions like 'function foo() { foo(); }'.

35850. **comment:** * Dump/load helpers, xxx_raw() helpers do no buffer checks
label: code-design

35851. may be NULL if no constants or inner funcs

35852. assert just a few critical flags

35853. Load inner functions to value stack, but don't yet copy to buffer.

35854. **comment:** XXX: awkward
label: code-design

35855. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway

35856. **comment:** Array is dense and contains only strings, but ASIZE may * be larger than used part and there are UNUSED entries.
label: code-design

35857. Important to do a fastint check so that constants are * properly read back as fastints.

35858. Value stack is used to ensure reachability of constants and * inner functions being loaded. Require enough space to handle * large functions correctly.

35859. Explicit zero size check to avoid NULL 'tv1'.

35860. func.prototype.constructor = func

35861. end of _Varmap

35862. -> [func funcname env funcname]

35863. -> [...] cons target]

35864. noblame_fileline

35865. [...] func this arg1 ... argN]

35866. thread

35867. * Figure out the final, non-bound constructor, to get "prototype" * property.

35868. **comment:** * Calls. * * Protected variants should avoid ever throwing an error.
label: code-design

35869. num_stack_res

35870. class Object, extensible

35871. note that we can't reliably pop anything here

35872. [...] retval]

35873. Strict by default.

35874. **comment:** XXX: merge this with duk_js_call.c, as this function implements * core semantics (or perhaps merge the two files altogether).
label: code-design

35875. num_stack_args

35876. nrets

35877. Anything else is not constructable.

35878. Prepare value stack for a method call through an object property. * May currently throw an error e.g. when getting the property.

35879. not protected, respect reclimit, is a constructor call

35880. respect reclimit, not constructor

35881. * Augment created errors upon creation (not when they are thrown or * rethrown). __FILE__ and __LINE__ are not desirable here; the call * stack reflects the caller which is correct.

35882. **comment:** XXX: code duplication
label: code-design

35883. not protected, respect reclimit, not constructor

35884. pop final_cons

35885. [...] fallback retval]

35886. -> [...] target]

35887. For user code this could just return 1 (strict) always * because all Duktape/C functions are considered strict, * and strict is also the default when nothing is running. * However, Duktape may call this function internally when * the current activation is an Ecmascript function, so * this cannot be replaced by a 'return 1' without fixing * the internal call sites.

35888. make absolute

35889. use fallback as 'this' value

35890. [...] fallback constructor fallback(this) arg1 ... argN]; * Note: idx_cons points to first 'fallback', not 'constructor'.

35891. **comment:** * XXX: if duk_handle_call() took values through indices, this could be * made much more sensible. However, duk_handle_call() needs to fudge * the 'this' and 'func' values to handle bound function chains, which * is now done "in-place", so this is not a trivial change.
label: code-design

35892. * Determine whether to use the constructor return value as the created * object instance or not.

35893. Lightfuncs cannot be bound.

35894. * Call the constructor function (called in "constructor mode").

35895. [...] key arg1 ... argN]

35896. Inputs: explicit arguments (nargs), +1 for key, +2 for obj_index/nargs passing. * If the value stack does not contain enough args, an error is thrown; this matches * behavior of the other protected call API functions.

35897. **comment:** * There are two [[Construct]] operations in the specification: * * - E5 Section 13.2.2: for Function objects * - E5 Section 15.3.4.5.2: for "bound" Function objects * * The chain of bound functions is resolved in Section 15.3.4.5.2, * with arguments "piling up" until the [[Construct]] internal * method is called on the final, actual Function object. Note * that the "prototype" property is looked up *only* from the * final object, *before* calling the constructor. * * Currently we follow the bound function chain here to get the * "prototype" property value from the final, non-bound function. * However, we let duk_handle_call() handle the argument "piling" * when the constructor is called. The bound function chain is * thus now processed twice. * * When constructing new Array instances, an unnecessary object is * created and discarded now: the standard [[Construct]] creates an * object, and calls the Array constructor. The Array constructor * returns

an Array instance, which is used as the result value for * the "new" operation; the object created before the Array constructor * call is discarded. ** This would be easy to fix, e.g. by knowing that the Array constructor * will always create a replacement object and skip creating the fallback * object in that case. ** Note: functions called via "new" need to know they are called as a * constructor. For instance, built-in constructors behave differently * depending on how they are called.

label: code-design

35898. [... constructor arg1 ... argN final_cons]

35899. See comments in duk_pcall().

35900. Checking callability of the immediate target * is important, same for constructability. * Checking it for functions down the bound * function chain is not strictly necessary * because .bind() should normally reject them. * But it's good to check anyway because it's * technically possible to edit the bound function * chain via internal keys.

35901. * Create "fallback" object to be used as the object instance, * unless the constructor returns a replacement value. * Its internal prototype needs to be set based on "prototype" * property of the constructor.

35902. call_flags

35903. Get the original arguments. Note that obj_index may be a relative * index so the stack must have the same top when we use it.

35904. must work for nargs <= 0

35905. * Duktape/C function magic

35906. fall through

35907. unreachable

35908. awkward; we assume there is space for this

35909. [... key arg1 ... argN func]

35910. func

35911. Note: -nargs alone would fail for nargs == 0, this is OK

35912. [... constructor arg1 ... argN]

35913. also stash it before constructor, * in case we need it (as the fallback value)

35914. nargs

35915. **comment:** For now, just use duk_safe_call() to wrap duk_new(). We can't * simply use a protected duk_handle_call() because there's post * processing which might throw. It should be possible to ensure * the post processing never throws (except in internal errors and * out of memory etc which are always allowed) and then remove this * wrapper.

label: code-design

35916. [... func arg1 ... argN]

35917. [... constructor arg1 ... argN final_cons fallback]

35918. duplicate key

35919. We can't reliably pop anything here because the stack input * shape is incorrect. So we throw an error; if the caller has * no catch point for this, a fatal error will occur. Another * alternative would be to just return an error. But then the * stack would be in an unknown state which might cause some * very hard to diagnose problems later on. Also note that even * if we did not throw an error here, the underlying call handler * might STILL throw an out-of-memory error or some other internal * fatal error.

35920. * Manipulate callstack for the call.

35921. **comment:** XXX: awkward; we assume there is space for this, overwrite * directly instead?

label: code-design

35922. * Must be careful to catch errors related to value stack manipulation * and property lookup, not just the call itself.

35923. -----XXXX

35924. output 3 bytes from 't'

35925. XX==

35926. Innermost fast path processes 4 valid base-64 characters at a time * but bails out on whitespace, padding chars ('=') and invalid chars. * Once the slow path segment has been processed, we return to the * inner fast path again. This handles e.g. base64 with newlines * reasonably well because the majority of a line is in the fast path.

35927. Fixed buffer, no zeroing because we'll fill all the data.

35928. allowed ascii whitespace

35929. full 3-byte -> 4-char conversions

35930. allow basic ASCII whitespace

35931. Here we'd have the option of decoding unpadded base64 * (e.g. "xxxxyy" instead of "xxxxyy==". Currently not * accepted.

35932. Computation must not wrap, only srclen + 3 is at risk of * wrapping because after that the number gets smaller. * This limit works for 32-bit size_t: * 0x100000000
- 3 - 1 = 4294967292

35933. XXX=

35934. read 3 bytes into 't', padded by zero

35935. back to fast loop

35936. never here

35937. **comment:** * Missing bytes snip base64 example * 0 4 XXXX * 1 3 XXX= * 2 2 XX==

label: code-design

35938. Backtrack.

35939. idx_value

35940. idx_space

35941. For invalid characters the value -1 gets extended to * at least 16 bits. If either nybble is invalid, the * resulting 't' will be < 0.

35942. -----XX XXXX-----

35943. **comment:** XXX: optimize for buffer inputs: no need to coerce to a string * which causes an unnecessary interning.

label: code-design

35944. DUK_USE_HEX_FASTPATH

35945. pointer is aligned, guaranteed for fixed buffer

35946. Don't allow actual chars after equal sign.

35947. Continue parsing after padding, allows concatenated, * padded base64.

35948. case 0: nop

35949. XXXXXX--

35950. **comment:** Shared handling for encode/decode argument. Fast path handling for * buffer and string values because they're the most common. In particular, * avoid creating a temporary string or buffer when possible.

label: code-design

35951. Handle one slow path unit (or finish if we're done).

35952. idx_reviver

35953. Check if any lookup above had a negative result.

35954. * Encoding and decoding basic formats: hex, base64. ** These are in-place operations which may allow an optimized implementation. ** Base-64:
<https://tools.ietf.org/html/rfc4648#section-4>

35955. **comment:** if 'src < src_end_safe', safe to read 4 bytes

label: requirement

35956. two level break

35957. **comment:** XXX: optimize for string inputs: no need to coerce to a buffer * which makes a copy of the input.

label: code-design

35958. Don't allow mixed padding and actual chars.

35959. -----XX

35960. The lookup byte is intentionally sign extended to (at least) * 32 bits and then ORed. This ensures that is at least 1 byte * is negative, the highest bit of 't' will be set at the end * and we don't need to check every byte.

35961. invalid padding

35962. We don't check the zero padding bytes here right now * (that they're actually zero). This seems to be common * behavior for base-64 decoders.

35963. aaaaabb bbbbcccc cccccccc

35964. **comment:** Here we can choose either to end parsing and ignore * whatever follows, or to continue parsing in case * multiple (possibly padded) base64 strings have been * concatenated. Currently, keep on parsing.
label: code-design

35965. **comment:** XXX: Using a string return value forces a string intern which is * not always necessary. As a rough performance measure, hex encode * time for tests/perf/test-hex-encode.js dropped from ~35s to ~15s * without string coercion. Change to returning a buffer and let the * caller coerce to string if necessary?
label: code-design

35966. upper limit, assuming no whitespace etc

35967. checked by caller

35968. shift in zeroes

35969. idx_replacer

35970. Fast path, handle units with just actual encoding characters.

35971. **comment:** XXX: convert to fixed buffer?
label: code-design

35972. Computation must not wrap; this limit works for 32-bit size_t: * >>> srclen = 3221225469 * >>> '%x' % ((srclen + 2) / 3 * 4) * 'fffffc'

35973. **comment:** There may be whitespace between the equal signs.
label: code-design

35974. XXXXX-- -----

35975. **comment:** Tested: not faster on x64
label: code-design

35976. **comment:** A straightforward 64-byte lookup would be faster * and cleaner, but this is shorter.
label: code-design

35977. Emit 3 bytes and backtrack if there was padding. There's * always space for the whole 3 bytes so no check needed.

35978. DUK_USE_BASE64_FASTPATH

35979. flags

35980. Note: for dstlen=0, dst may be NULL

35981. Note: for srclen=0, src may be NULL

35982. * Compilation and evaluation

35983. Arguments can be: [source? filename? &comp_args] so that * nargs is 1 to 3. Call site encodes the correct nargs count * directly into flags.

35984. Helper which can be called both directly and with duk_safe_call().

35985. [... source? func_template]

35986. [... source? filename?]

35987. add_auto_proto

35988. [... closure/error]

35989. **comment:** String length is computed here to avoid multiple evaluation * of a macro argument in the calling side.
label: code-design

35990. [... result/error]

35991. **comment:** Note: strictness is *not* inherited from the current Duktape/C. * This would be confusing because the current strictness state * depends on whether we're running inside a Duktape/C activation * (= strict mode) or outside of any activation (= non-strict mode). * See tests/api/test-eval-strictness.c for more discussion.
label: code-design

35992. [... source? filename?] (depends on flags)

35993. [... func_template]

35994. **comment:** XXX: when this error is caused by a nonexistent * file given to duk_peval_file() or similar, the * error message is not the best possible.
label: code-design

35995. e.g. duk_push_string_file_raw() pushed undefined

35996. should be first on 64-bit platforms

35997. Automatic filename: 'eval' or 'input'.

35998. [... source? filename? &comp_args] (depends on flags)

35999. explicit 'this' binding, see GH-164

36000. [... closure]

36001. **comment:** XXX: unnecessary translation of flags
label: code-design

36002. Note: strictness is not inherited from the current Duktape/C * context. Otherwise it would not be possible to compile * non-strict code inside a Duktape/C activation (which is * always strict now). See tests/api/test-eval-strictness.c * for discussion.

36003. Eval is just a wrapper now.

36004. args incorrect

36005. may be safe, or non-safe depending on flags

36006. -> [... closure]

36007. [... source? filename]

36008. Treat like debugger statement: nop

36009. [... arr jsonx(arr) res] -> [... res jsonx(arr)]

36010. **comment:** XXX: conversion errors should not propagate outwards. * Perhaps values need to be coerced individually?
label: code-design

36011. idx_value

36012. idx_space

36013. **comment:** XXX: should there be an error or an automatic detach if * already attached?
label: code-design

36014. Other callbacks are optional.

36015. nop

36016. Pause on the next opcode executed. This is always safe to do even * inside the debugger message loop: the interrupt counter will be reset * to its proper value when the message loop exits.

36017. DUK_USE_DEBUGGER_SUPPORT

36018. Start in paused state.

36019. no_block

36020. No debugger support, just pop values.

36021. idx_replacer

36022. unreachable

36023. Return non-zero (true) if we have a good reason to believe * the notify was delivered; if we're still attached at least * a transport error was not indicated by the transport write * callback. This is not a 100% guarantee of course.

36024. Calling duk_debugger_cooperate() while Duktape is being * called into is not supported. This is not a 100% check * but prevents any damage in most cases.

36025. Treat like a debugger statement: ignore when not attached.

36026. We don't duk_require_stack() here now, but rely on the caller having * enough space.

36027. * Debugging related API calls

36028. flags
36029. Send version identification and flush right afterwards. Note that * we must write raw, unframed bytes here.
36030. Can be called multiple times with no harm.
36031. [... new_glob new_env]
36032. side effects, in theory (referenced by global env)
36033. Assume that either all memory funcs are NULL or non-NUL, mixed * cases will now be unsafe.
36034. * Heap creation and destruction
36035. [... new_glob new_env new_glob new_glob]
36036. XXX: just assert non-NUL values here and make caller arguments * do the defaulting to the default implementations (smaller code)?
36037. unvalidated
36038. side effects
36039. * Replace lexical environment for global scope ** Create a new object environment for the global lexical scope. * We can't just reset the _Target property of the current one, * because the lexical scope is shared by other threads with the * same (initial) built-ins.
36040. [...]
36041. **comment:** XXX: better place for this
 label: code-design
36042. * Replace global object.
36043. no prototype, updated below
36044. without refcounts
36045. **comment:** These are macros for now, but could be separate functions to reduce code * footprint (check call site count before refactoring).
 label: code-design
36046. sometimes stack and array indices need to go on the stack
36047. [key val] -> []
36048. Current convention is to use duk_size_t for value stack sizes and global indices, * and duk_idx_t for local frame indices.
36049. [val] -> []
36050. **comment:** XXX: assumed to fit for now
 label: code-design
36051. duk_push_this() + CheckObjectCoercible() + duk_to_object()
36052. only needed by debugger for now
36053. * Internal API calls which have (stack and other) semantics similar * to the public API.
36054. Raw internal valstack access macros: access is unsafe so call site * must have a guarantee that the index is valid. When that is the case, * using these macro results in faster and smaller code than duk_get_tval(). * Both 'ctx' and 'idx' are evaluated multiple times, but only for asserts.
36055. Flag ORed to err_code to indicate __FILE__ / __LINE__ is not * blamed as source of error for error fileName / lineNumber.
36056. [] -> []
36057. duk_push_sprintf constants
36058. **comment:** XXX: add fastint support?
 label: requirement
36059. This would be pointless: unexpected type and lightfunc would both return NULL
36060. duk_push_(u)int() is guaranteed to support at least (un)signed 32-bit range
36061. Get a borrowed duk_tval pointer to the current 'this' binding. Caller must * make sure there's an active callstack entry. Note that the returned pointer * is unstable with regards to side effects.
36062. DUK_API_INTERNAL_H_INCLUDED
36063. Set object 'length'.
36064. duk_push_this() + CheckObjectCoercible() + duk_to_string()
36065. Push the current 'this' binding; throw TypeError if binding is not object * coercible (CheckObjectCoercible).
36066. [] -> [val]
36067. Valstack resize flags
36068. out_clamped=NULL, RangeError if outside range
36069. **comment:** unused
 label: code-design
36070. stridx_logfunc[] must be static to allow initializer with old compilers like BCC
36071. [... Logger clog logfunc clog(=this) msg]
36072. nargs
36073. [... Logger clog logfunc clog]
36074. **comment:** * Logging ** Current logging primitive is a sprintf-style log which is convenient * for most C code. Another useful primitive would be to log N arguments * from value stack (like the Ecmascript binding does).
 label: code-design
36075. [... Logger clog res]
36076. * Memory calls.
36077. NULL accepted
36078. **comment:** * Note: since this is an exposed API call, there should be * no way a mark-and-sweep could have a side effect on the * memory allocation behind 'ptr'; the pointer should never * be something that Duktape wants to change. ** Thus, no need to use DUK_REALLOC_INDIRECT (and we don't * have the storage location here anyway).
 label: code-design
36079. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property get right now.
36080. * Helpers for writing multiple properties
36081. Key and value indices are either (-2, -1) or (-1, -2). Given idx_key, * idx_val is always (idx_key ^ 0x01).
36082. remove key and value
36083. [... obj ...]
36084. This is a rare property helper; it sets the global thrower (E5 Section 13.2.3) * setter/getter into an object property. This is needed by the 'arguments' * object creation code, function instance creation code, and Function.prototype.bind().
36085. * Object finalizer
36086. proto can also be NULL here (allowed explicitly)
36087. a value is left on stack regardless of rc
36088. pop key
36089. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property put right now (putprop protects * against it internally).
36090. "Have" flags must not be conflicting so that they would * apply to both a plain property and an accessor at the same * time.
36091. Object.defineProperty() equivalent C binding.
36092. Define own property without inheritance looks and such. This differs from * [[DefineOwnProperty]] because special behaviors (like Array 'length') are * not invoked by this method. The caller must be careful to invoke any such * behaviors if necessary.
36093. **comment:** XXX: the duk_hobject_enum.c stack APIs should be reworked
 label: code-design
36094. defprop_flags
36095. * Property handling ** The API exposes only the most common property handling functions. * The caller can invoke Ecmascript built-ins for full control (e.g. * defineProperty, getOwnPropertyDescriptor).
36096. * Object related * * Note: seal() and freeze() are accessible through Ecmascript bindings, * and are not exposed through the API.
36097. XXX: shared helper for duk_push_hobject_or_undefined()
36098. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property existence check right now.

36099. * Object handling: property access and other support functions.
36100. remove key
36101. **comment:** Careful here and with other duk_put_prop_xxx() helpers: the * target object and the property value may be in the same value * stack slot (unusual, but still conceptually clear).
 label: code-design
36102. value popped by call
36103. 1 if property found, 0 otherwise
36104. **comment:** XXX: "read only object"?
 label: code-design
36105. [...] -> [...] global]
36106. Note: this may fail, caller should protect the call if necessary
36107. **comment:** XXX: direct implementation
 label: requirement
36108. **comment:** Clean up stack
 label: code-design
36109. [target] -> [enum]
36110. * Object prototype
36111. **comment:** XXX: these could be implemented as macros calling an internal function * directly. * XXX: same issue as with Duktape.fin: there's no way to delete the property * now (just set it to undefined).
 label: code-design
36112. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property delete right now.
36113. * Shortcut for accessing global object properties
36114. **comment:** XXX: shared api error strings, and perhaps even throw code for rare cases?
 label: code-design
36115. Maximum value check ensures 'nbytes' won't wrap below.
36116. DUK_BUFOBJ_UINT32ARRAY
36117. throw_flag
36118. only flag now
36119. DUK_USE_VARIADIC_MACROS
36120. E5 Section 9.1
36121. The underlying types for offset/length in duk_hbufferobject is * duk_uint_t; make sure argument values fit and that offset + length * does not wrap.
36122. * When resizing the valstack, a mark-and-sweep may be triggered for * the allocation of the new valstack. If the mark-and-sweep needs * to use our thread for something, it may cause *the same valstack* * to be resized recursively. This happens e.g. when mark-and-sweep * finalizers are called. This is taken into account carefully in * duk_resize_valstack(). * * 'new_size' is known to be <= valstack_max, which ensures that * size_t and pointer arithmetic won't wrap in duk_resize_valstack().
36123. **comment:** Function pointers do not always cast correctly to void * * (depends on memory and segmentation model for instance), * so they coerce to NULL.
 label: code-design
36124. This would be pointless: we'd return NULL for both lightfuncs and * unexpected types.
36125. Lightfunc coerces to a Function instance with concrete * properties. Since 'length' is virtual for Duktape/C * functions, don't need to define that. * * The result is made extensible to mimic what happens to * strings: * > Object.isExtensible(Object('foo')) * true
36126. consequence of above
36127. Never executed if new size is smaller.
36128. * An array must have a 'length' property (E5 Section 15.4.5.2). * The special array behavior flag must only be enabled once the * length property has been added. * * The internal property must be a number (and preferably a * fastint if fastint support is enabled).
36129. **comment:** * Global state for working around missing variadic macros
 label: code-design
36130. use a temp: decref only when valstack reachable values are correct
36131. Note: recursive call
36132. * Stack index validation/normalization and getting a stack duk_tval ptr. * * These are called by many API entrypoints so the implementations must be * fast and "inlined". * * There's some repetition because of this; keep the functions in sync.
36133. **comment:** XXX: There's quite a bit of overlap with buffer creation handling in * duk.bi_buffer.c. Look for overlap and refactor.
 label: code-design
36134. Should match Function.prototype.toString()
36135. **comment:** XXX: combine all the integer conversions: they share everything * but the helper function for coercion.
 label: code-design
36136. fatal_func should be noreturn, but noreturn declarations on function * pointers has a very spotty support apparently so it's not currently * done.
36137. Allowed.
36138. **comment:** XXX: .code = err_code disabled, not sure if useful
 label: code-design
36139. * Special cases like NaN and +/- Infinity are handled explicitly * because a plain C coercion from double to int handles these cases * in undesirable ways. For instance, NaN may coerce to INT_MIN * (not zero), and INT_MAX + 1 may coerce to INT_MIN (not INT_MAX). * * This double-to-int coercion differs from ToInteger() because it * has a finite range (ToInteger() allows e.g. +/- Infinity). It * also differs fromToInt32() because the INT_MIN/INT_MAX clamping * depends on the size of the int type on the platform. In particular, * on platforms with a 64-bit int type, the full range is allowed.
36140. Assert for value stack initialization policy.
36141. val is unsigned so >= 0
36142. important to do this *after* pushing, to make the thread reachable for gc
36143. !DUK_USE_PARANOIA_ERRORS
36144. initial estimate based on format string
36145. no need to incref
36146. coerce towards zero
36147. Wrap checked above.
36148. * API calls related to general value stack manipulation: resizing the value * stack, pushing and popping values, type checking and reading values, * coercing values, etc. * * Also contains internal functions (such as duk_get_tval()), defined * in duk_api_internal.h, with semantics similar to the public API.
36149. Non-critical.
36150. default: false
36151. success, fixup pointers
36152. Coerce top into Object.prototype.toString() output.
36153. * Misc helpers
36154. nbytes zero size case * <-----> * [...] p | x | x | q [...] p==q] * => [...] x | x | q [...]
36155. **comment:** XXX: several pointer comparison issues here
 label: code-design
36156. must be computed after realloc
36157. side effects
36158. number
36159. radix
36160. Popping one element is called so often that when footprint is not an issue, * compile a specialized function for it.
36161. Here we rely on duk_hstring instances always being zero * terminated even if the actual string is not.
36162. get pointer offsets for tweaking below

36163. zero size not an issue: pointers are valid
36164. DUK_BUFOBJ_INT8ARRAY
36165. TypedArray views need an automatic ArrayBuffer which must be * provided as .buffer property of the view. Just create a new * ArrayBuffer sharing the same underlying buffer. * * The ArrayBuffer offset is always set to zero, so that if one * accesses the ArrayBuffer at the view's .byteOffset, the value * matches the view at index 0.
36166. advance manually
36167. out_clamped==NULL -> RangeError if outside range
36168. not reached
36169. because of valstack init policy
36170. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Returns NULL for a lightfunc.
36171. ensure value is 1 or 0 (not other non-zero)
36172. DUK_BUFOBJ_INT16ARRAY
36173. No net refcount changes.
36174. **comment:** SCANBUILD: with suitable dmin/dmax limits 'd' is unused
 label: code-design
36175. object is now reachable
36176. 4 low bits
36177. * Pushers
36178. quotes
36179. String table assert check omitted from 1.x branch * backport.
36180. XXX: perhaps refactor this to allow caller to specify some parameters, or * at least a 'compact' flag which skips any spare or round-up .. useful for * emergency gc.
36181. **comment:** XXX: string not shared because it is conditional
 label: code-design
36182. E5 Section 8.12.8
36183. 'd' and 'res' agree here
36184. XXX: direct valstack write
36185. Creation time error augmentation
36186. filename may be NULL in which case file/line is not recorded
36187. Check for maximum buffer length.
36188. Inref copies, keep originals.
36189. DUK_HOBJECT_FLAG_EXOTIC_DUKFUNC: omitted here intentionally
36190. 'undefined' already on stack top
36191. unsigned
36192. covers +Infinity
36193. **comment:** A plain buffer coerces to a Duktape.Buffer because it's the * object counterpart of the plain buffer value. But it might * still make more sense to produce an ArrayBuffer here?
 label: code-design
36194. avoid warning (unsigned)
36195. default prototype (Note: 'obj' must be reachable)
36196. everything except object stay as is
36197. flags
36198. Enable DUKFUNC exotic behavior once properties are set up.
36199. DUK_BUFOBJ_UINT8ARRAY
36200. may be NULL
36201. **comment:** No coercions or other side effects, so safe
 label: requirement
36202. Allocate a new valstack. * * Note: cannot use a plain DUK_REALLOC() because a mark-and-sweep may * invalidate the original thr->valstack base pointer inside the realloc * process. See doc/memory-management.rst.
36203. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is never NULL.
36204. DUK_BUFOBJ_DATAVIEW
36205. If tv1==tv2 this is a NOP, no check is needed
36206. DUK_USE_ASSERTIONS
36207. * Basic stack manipulation: swap, dup, insert, replace, etc
36208. No net refcount change.
36209. **comment:** XXX: shorter version for 12-byte representation?
 label: code-design
36210. special handling of fmt==NULL
36211. specific assert for wrapping
36212. Clamping only necessary for 32-bit ints.
36213. **comment:** * Function pointers * * Printing function pointers is non-portable, so we do that by hex printing * bytes from memory.
 label: code-design
36214. DUK_BUFOBJ_UINT8CLAMPEDARRAY
36215. caller required to know
36216. ANSI C typing
36217. as is
36218. Only allow Duktape.Buffer when support disabled.
36219. **comment:** XXX: Simplify this algorithm, should be possible to come up with * a shorter and faster algorithm by inspecting IEEE representation * directly.
 label: code-design
36220. String sanitizer which escapes ASCII control characters and a few other * ASCII characters, passes Unicode as is, and replaces invalid UTF-8 with * question marks. No errors are thrown for any input string, except in out * of memory situations.
36221. Explicit length is only needed if it differs from 'nargs'.
36222. DUK_BUFOBJ_FLOAT64ARRAY
36223. Reviving an object using a heap pointer is a dangerous API * operation: if the application doesn't guarantee that the * pointer target is always reachable, difficult-to-diagnose * problems may ensue. Try to validate the 'ptr' argument to * the extent possible.
36224. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is NULL if value is neither an object nor a * lightfunc.
36225. * Must be very careful here, every DECREF may cause reallocation * of our valstack.
36226. inline initialized for coercers[] is not allowed by old compilers like BCC
36227. **comment:** XXX: duk_ssize_t would be useful here
 label: code-design
36228. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
36229. **comment:** XXX: better macro for DUK_TVAL_IS_UNDEFINED_OR_NULL(tv)
 label: code-design
36230. no shrink
36231. nop: insert top to top
36232. **comment:** Lightfunc name, includes Duktape/C native function pointer, which * can often be used to locate the function from a symbol table. * The name also includes the 16-bit duk_tval flags field because it * includes the magic value. Because a single native function often * provides different functionality depending on

the magic value, it * seems reasonably to include it in the name. ** On the other hand, a complicated name increases string table * pressure in low memory environments (but only when function name * is accessed).

label: code-design

36233. periods

36234. Note: here we must be wary of the fact that a pointer may be * valid and be a NULL.

36235. Nothing to initialize, str[] is in ROM.

36236. [... parent stash stash] -> [... parent stash]

36237. if slice not fully valid, treat as error

36238. Note: no need to re-lookup tv, conversion is side effect free

36239. Template functions are not strictly constructable (they don't * have a "prototype" property for instance), so leave the * DUK_HOBJECT_FLAG_CONSTRUCTABLE flag cleared here.

36240. DUK_BUFOBJ_ARRAYBUFFER

36241. check stack first

36242. Object is currently being finalized.

36243. [... retval]; popped below

36244. tv points to element just below prev top

36245. **comment:** XXX: fastint?

label: code-design

36246. Would indicate corrupted lists.

36247. dynamic

36248. DUK_USE_FILE_IO

36249. Return invalid index; if caller uses this without checking * in another API call, the index won't map to a valid stack * entry.

36250. ignore fclose() error

36251. DUK_BUFOBJ_FLOAT32ARRAY

36252. not reachable

36253. coerced value is updated to value stack even when RangeError thrown

36254. error gets its 'name' from the prototype

36255. no side effects

36256. shrink case; leave some spare

36257. 'this' binding exists

36258. may be NULL (but only if size is 0)

36259. failed, resize and try again

36260. * Poppers

36261. -> [... func this]

36262. **comment:** not very useful, used for debugging

label: code-design

36263. * Lightfunc

36264. Because new_size != 0, if condition doesn't need to be * (new_valstack != NULL || new_size == 0).

36265. Not halfway, round to nearest.

36266. * Number is special because it doesn't have a specific * tag in the 8-byte representation.

36267. **comment:** XXX: fast primitive to set a bunch of values to UNDEFINED

label: code-design

36268. make the new thread reachable

36269. **comment:** copied so that 'ap' can be reused

label: code-design

36270. estimate is valid

36271. * Comparison

36272. DUK_USE_REFERENCE_COUNTING

36273. **comment:** XXX: repetition of stack pre-checks -> helper or macro or inline

label: code-design

36274. Value coercion (in stack): ToInteger(), E5 Section 9.4 * API return value coercion: custom

36275. [... func]

36276. nrets

36277. Error code also packs a tracedata related flag.

36278. For tv1 == tv2, both pointing to stack top, the end result * is same as duk_pop(ctx).

36279. allow fmt==NULL

36280. thr->heap->jmpbuf_ptr is checked by duk_err_longjmp() so we don't * need to check that here. If the value is NULL, a panic occurs because * we can't return.

36281. DUK_BUFOBJ_NODEJS_BUFFER

36282. Stack size increases or stays the same.

36283. Exact halfway, round to even.

36284. **comment:** If no explicit message given, put error code into message field * (as a number). This is not fully in keeping with the EcmaScript * error model because messages are supposed to be strings (Error * constructors use ToString() on their argument). However, it's * probably more useful than having a separate 'code' property.

label: code-design

36285. Note: may be triggered even if minimal new_size would not reach the limit, * plan limit accordingly (taking DUK_VALSTACK_GROW_STEP into account).

36286. We intentionally ignore the duk_safe_call() return value and only * check the output type. This way we don't also need to check that * the returned value is indeed a string in the success case.

36287. Cannot use duk_to_string() on the buffer because it is usually * larger than 'len'. Also, 'buf' is usually a stack buffer.

36288. Clamping needed if duk_int_t is 64 bits.

36289. Note: src_data may be NULL if input is a zero-size * dynamic buffer.

36290. !DUK_USE_PREFER_SIZE

36291. Error: try coercing error to string once.

36292. Note: the realloc may have triggered a mark-and-sweep which may * have resized our valstack internally. However, the mark-and-sweep * MUST NOT leave the stack bottom/top in a different state. Particular * assumptions and facts: * * - The thr->valstack pointer may be different after realloc, * and the offset between thr->valstack_end <-> thr->valstack * may have changed. * - The offset between thr->valstack_bottom <-> thr->valstack * and thr->valstack_top <-> thr->valstack MUST NOT have changed, * because mark-and-sweep must adhere to a strict stack policy. * In other words, logical bottom and top MUST NOT have changed. * - All values above the top are unreachable but are initialized * to UNDEFINED, up to the post-realloc valstack_end. * - 'old_end_offset' must be computed after realloc to be correct.

36293. **comment:** XXX: inlined DECREF macro would be nice here: no NULL check, * refzero queueing but no refzero algorithm run (= no pointer * instability), inline code.

label: code-design

36294. **comment:** XXX: other variants like uint, u32 etc

label: code-design

36295. rely on interning, must be this string

36296. DUK_USE_FASTINT

36297. Because value stack init policy is 'undefined above top', * we don't need to write, just assert.

36298. sanity limit for function pointer size

36299. Initial stack size satisfies the stack spare constraints so there * is no need to require stack here.

36300. * Error throwing
36301. can't resize below 'top'
36302. -> [... retval]
36303. Non-buffer value is first ToString() coerced, then converted * to a buffer (fixed buffer is used unless a dynamic buffer is * explicitly requested).
36304. shared flags for a subset of types
36305. round up roughly to next 'grow step'
36306. Custom coercion for API
36307. NULL with zero length represents an empty string; NULL with higher * length is also now treated like an empty string although it is * a bit dubious. This is unlike duk_push_string() which pushes a * 'null' if the input string is a NULL.
36308. tv -> value just before prev top value; must relookup
36309. Init newly allocated slots (only).
36310. default: NULL, length 0
36311. * instanceof
36312. Assume value stack sizes (in elements) fits into duk_idx_t.
36313. useful for debugging
36314. **comment:** Relookup in case duk_js_tointeger() ends up e.g. coercing an object.
 label: code-design
36315. valstack limit caller has check, prevents wrapping
36316. default: NaN
36317. Coercion may be needed, the helper handles that by pushing the * tagged values to the stack.
36318. Note: this check relies on the fact that even a zero-size string * has a non-NULL pointer.
36319. Negative indices are always within allocated stack but * must not go below zero index.
36320. covers -Infinity
36321. * Get/require
36322. **comment:** * Number should already be in NaN-normalized form, * but let's normalize anyway.
 label: code-design
36323. Note: number has no explicit tag (in 8-byte representation)
36324. Positive index can be higher than valstack top but must * not go above allocated stack (equality is OK).
36325. Same as above but for unsigned int range.
36326. Errors are augmented when they are created, not when they are * thrown or re-thrown. The current error handler, however, runs * just before an error is thrown.
36327. **comment:** XXX: Hex encoded, length limited buffer summary here?
 label: code-design
36328. When src_size == 0, src_data may be NULL (if source * buffer is dynamic), and dst_data may be NULL (if * target buffer is dynamic). Avoid zero-size memcpy() * with an invalid pointer.
36329. check stack before interning (avoid hanging temp)
36330. **comment:** Try to make do with a stack buffer to avoid allocating a temporary buffer. * This works 99% of the time which is quite nice.
 label: code-design
36331. * Stack slice primitives
36332. **comment:** unused
 label: code-design
36333. normalize NaN which may not match our canonical internal NaN
36334. * Value stack top handling
36335. check_object_coercible
36336. If not present in finalize_list or refzero_list, the pointer * must be either in heap_allocated or the string table.
36337. One particular problem case is where an object has been * queued for finalization but the finalizer hasn't yet been * executed. * * Corner case: we're running in a finalizer for object X, and * user code calls duk_push_heapptr() for X itself. In this * case X will be in finalize_list, and we can detect the case * by seeing that X's FINALIZED flag is set (which is done before * the finalizer starts executing).
36338. **comment:** Note: Boolean prototype's internal value property is not writable, * but duk_xdef_prop_stridx() disregards the write protection. Boolean * instances are immutable. * * String and buffer special behaviors are already enabled which is not * ideal, but a write to the internal value is not affected by them.
 label: code-design
36339. Note: this allows creation of internal strings.
36340. Check for maximum string length
36341. E5 Section 9.2
36342. force_exponential
36343. DUK_INVALID_INDEX won't be accepted as a valid index.
36344. only needed by debugger for now
36345. Raw helper for getting a value from the stack, checking its tag. * The tag cannot be a number because numbers don't have an internal * tag in the packed representation.
36346. DUK_USE_BUFFEROBJECT_SUPPORT
36347. Heap allocated: return heap pointer which is NOT useful * for the caller, except for debugging.
36348. * Forward declarations
36349. **comment:** This is a bit clunky because it is ANSI C portable. Should perhaps * relocate to another file because this is potentially platform * dependent.
 label: code-design
36350. For tv1 == tv2, this is a no-op (no explicit check needed).
36351. require
36352. **comment:** XXX: typing
 label: code-design
36353. Use unsigned arithmetic to optimize comparison.
36354. **comment:** * Number should already be in NaN-normalized form, but let's * normalize anyway.
 label: code-design
36355. **comment:** XXX: take advantage of val being unsigned, no need to mask
 label: code-design
36356. Return value of 'sz' or more indicates output was (potentially) * truncated.
36357. DUK_BUFOBJ_INT32ARRAY
36358. Also check for the refzero_list; must not be there unless it is * being finalized when duk_push_heapptr() is called. * * Corner case: similar to finalize_list.
36359. nop
36360. Faster but value stack overruns are memory unsafe.
36361. E5 Section 9.4, ToInteger()
36362. Note: need to re-lookup because ToNumber() may have side effects
36363. Relookup in case coerce_func() has side effects, e.g. ends up coercing an object
36364. Buffer is kept as is, with the fixed/dynamic nature of the * buffer only changed if requested. An external buffer * is converted into a non-external dynamic buffer in a * duk_to_dynamic_buffer() call.
36365. * Type checking
36366. Double error
36367. no compact
36368. w/o refcounting
36369. [... func/retval] -> [...]
36370. Special coercion for Uint8ClampedArray.

36371. flags is unsigned
36372. Example: d=3.5, t=0.5 -> ret = (3 + 1) & 0xfe = 4 & 0xfe = 4 * Example: d=4.5, t=0.5 -> ret = (4 + 1) & 0xfe = 5 & 0xfe = 4
36373. **comment:** XXX: optimize loops
 label: code-design
36374. Handle change in value stack top. Respect value stack * initialization policy: 'undefined' above top. Note that * DECREF may cause a side effect that reallocates valstack, * so must relookup after DECREF.
36375. nbytes * <-----> * [... | p | x | x | q] * => [... | q | p | x | x]
36376. tagged null pointers should never occur
36377. no throw
36378. Care must be taken to avoid pointer wrapping in the index * validation. For instance, on a 32-bit platform with 8-byte * duk_tval the index 0x20000000UL would wrap the memory space * once.
36379. * Conversions and coercions * * The conversion/coercions are in-place operations on the value stack. * Some operations are implemented here directly, while others call a * helper in duk_js_ops.c after validating arguments.
36380. initialize built-ins - either by copying or creating new ones
36381. may throw an error
36382. **comment:** Set stack top within currently allocated range, but don't reallocate. * This is performance critical especially for call handling, so whenever * changing, profile and look at generated code.
 label: code-design
36383. **comment:** XXX: size optimize
 label: code-design
36384. **comment:** NUL terminator handling doesn't matter here
 label: code-design
36385. Index validation is strict, which differs from duk_equals(). * The strict behavior mimics how instanceof itself works, e.g. * it is a TypeError if rval is not a - callable- object. It would * be somewhat inconsistent if rval would be allowed to be * non-existent without a TypeError.
36386. **comment:** Clamping to zero makes the API more robust to calling code * calculation errors.
 label: code-design
36387. Surround with parentheses like in JX, ensures NULL pointer * is distinguishable from null value ("null" vs "null").
36388. * Value stack resizing. * * This resizing happens above the current "top": the value stack can be * grown or shrunk, but the "top" is not affected. The value stack cannot * be resized to a size below the current "top". * * The low level reallocation primitive must carefully recompute all value * stack pointers, and must also work if ALL pointers are NULL. The resize * is quite tricky because the valstack realloc may cause a mark-and-sweep, * which may run finalizers. Running finalizers may resize the valstack * recursively (the same value stack we're working on). So, after realloc * returns, we know that the valstack "top" should still be the same (there * should not be live values above the "top"), but its underlying size and * pointer may have changed.
36389. Stack size decreases.
36390. format plus something to avoid just missing
36391. copy values (no overlap even if to_ctx == from_ctx; that's not * allowed now anyway)
36392. unreachable
36393. **comment:** XXX: this is probably a useful shared helper: for a * duk_hbufferobject, get a validated buffer pointer/length.
 label: code-design
36394. Represent a null pointer as 'null' to be consistent with * the JX format variant. Native '%p' format for a NULL * pointer may be e.g. '(nil)'.
36395. **comment:** XXX: duk_ssize_t
 label: code-design
36396. ... and its 'message' from an instance property
36397. DUK_BUFOBJ_DUKTAPE_BUFFER
36398. DUK_BUFOBJ_UINT16ARRAY
36399. nargs
36400. **comment:** * Push readable string summarizing duk_tval. The operation is side effect * free and will only throw from internal errors (e.g. out of memory). * This is used by e.g. property access code to summarize a key/base safely, * and is not intended to be fast (but small and safe).
 label: code-design
36401. [buf? res]
36402. Avoid NaN-to-integer coercion as it is compiler specific.
36403. no prototype
36404. Check that there's room to push one value.
36405. Note: 'q' is top-1
36406. **comment:** XXX: This will now return false for non-numbers, even though they would * coerce to NaN (as a general rule). In particular, duk_get_number() * returns a NaN for non-numbers, so should this function also return * true for non-numbers?
 label: code-design
36407. min_new_size
36408. since index non-negative
36409. not popped by side effect
36410. precision:shortest
36411. 'this' binding is just before current activation's bottom
36412. [... (sep) str1 str2 ... strN buf]
36413. no size check is necessary
36414. use stack allocated buffer to ensure reachability in errors (e.g. intern error)
36415. overwrite str1
36416. extra -1 for buffer
36417. **comment:** XXX: could write output in chunks with fewer ensure calls, * but relative benefit would be small here.
 label: code-design
36418. Combined size of separators already overflows
36419. [... buf]
36420. always true, arg is unsigned
36421. start (incl) and end (excl) of trimmed part
36422. [... res]
36423. Impose a string maximum length, need to handle overflow * correctly.
36424. overwrite sep
36425. guaranteed by string limits
36426. XXX: this is quite clunky. Add Unicode helpers to scan backwards and * forwards with a callback to process codepoints?
36427. **comment:** XXX: could map/decode be unified with duk_unicode_support.c code? * Case conversion needs also the character surroundings though.
 label: code-design
36428. entire string is whitespace
36429. A bit tricky overflow test, see doc/code-issues.rst.
36430. * String manipulation
36431. pointers for scanning
36432. wrapped
36433. This may happen when forward and backward scanning disagree * (possible for non-extended-UTF-8 strings).
36434. is_join
36435. **comment:** get rid of the strings early to minimize memory use before intern
 label: code-design

36436. reasonable output estimate
36437. * Variable access
36438. [... val]
36439. always throw ReferenceError for unresolvable
36440. Return value would be pointless: because throw_flag==1, we always * throw if the identifier doesn't resolve.
36441. [... varname val]
36442. **comment:** XXX: tostring?
 label: code-design
36443. [...]
36444. [... varname val this] (because throw_flag == 1, always resolved)
36445. -> [... varname val this]
36446. Outside any activation -> put to global.
36447. Outside any activation -> look up from global.
36448. -> [... val retval]
36449. Real world behavior for map(): trailing non-existent * elements don't invoke the user callback, but are still * counted towards result 'length'.
36450. stack top contains 'false'
36451. Steps 12 and 13: reorganize elements to make room for itemCount elements
36452. **comment:** Shared entry code for many Array built-ins. Note that length is left * on stack (it could be popped, but that's not necessary).
 label: code-design
36453. **comment:** XXX: expensive check (also shared elsewhere - so add a shared internal API call?)
 label: code-design
36454. swap elements; deal with non-existent elements correctly
36455. move pivot to its final place
36456. return ToObject(this)
36457. **comment:** Note: 'this' is not necessarily an Array object. The push() * algorithm is supposed to work for other kinds of objects too, * so the algorithm has e.g. an explicit update for the 'length' * property which is normally "magical" in arrays.
 label: code-design
36458. -> [sep ToObject(this) len str]
36459. end of loop (careful with len==0)
36460. stack[0] = start * stack[1] = deleteCount * stack[2...nargs-1] = items * stack[nargs] = ToObject(this) -3 * stack[nargs+1] = ToUint32(length) -2 * stack[nargs+2] = result array -1
36461. for lastIndexOf, result may be -1 (mark immediate termination)
36462. According to E5.1 Section 15.4.4.4 nonexistent trailing * elements do not affect 'length' of the result. Test262 * and other engines disagree, so update idx_last here too.
36463. For now, restrict result array into 32-bit length range.
36464. avoid degenerate cases, so that (len - 1) won't underflow
36465. * shift()
36466. fixed offsets in valstack
36467. k+argCount-1; note that may be above 32-bit range
36468. intermediate join to avoid valstack overflow
36469. orig value
36470. fromPresent = true
36471. XXX: len >= 0x80000000 won't work below because we need to be * able to represent -len.
36472. right exists, [[Put]] regardless whether or not left exists
36473. [ToObject(this) item1 ... itemN arr item(i) item(i)[j]]
36474. topmost element is the result array already
36475. -> [sep ToObject(this) len sep str]
36476. [ToObject(this) item1 ... itemN arr]
36477. The lo/hi indices may be crossed and hi < 0 is possible at entry.
36478. **comment:** XXX: This helper is a bit awkward because the handling for the different iteration * callers is quite different. This now compiles to a bit less than 500 bytes, so with * 5 callers the net result is about 100 bytes / caller.
 label: code-design
36479. [sep ToObject(this) len sep result]
36480. **comment:** XXX: there's no explicit recursion bound here now. For the average * qsort recursion depth O(log n) that's not really necessary: e.g. for * 2**32 recursion depth would be about 32 which is OK. However, qsort * worst case recursion depth is O(n) which may be a problem.
 label: code-design
36481. Index clamping is a bit tricky, we must ensure that we'll only iterate * through elements that exist and that the specific requirements from E5.1 * Sections 15.4.4.14 and 15.4.4.15 are fulfilled; especially: * * - indexOf: clamp to [-len,len], negative handling -> [0,len], * if clamped result is len, for-loop bails out immediately * * - lastIndexOf: clamp to [-len-1, len-1], negative handling -> [-1, len-1], * if clamped result is -1, for-loop bails out immediately * * If fromIndex is not given, ToInteger(undefined) = 0, which is correct * for indexOf() but incorrect for lastIndexOf(). Hence special handling, * and why lastIndexOf() needs to be a vararg function.
36482. **comment:** XXX: best behavior for real world compatibility?
 label: code-design
36483. Range limited to [0, 0x7fffffff] range, i.e. range that can be * represented with duk_int32_t. Use this when the method doesn't * handle the full 32-bit unsigned range correctly.
36484. **comment:** Fall back to the initial (original) Object.toString(). We don't * currently have pointers to the built-in functions, only the top * level global objects (like "Array") so this is now done in a bit * of a hacky manner. It would be cleaner to push the (original) * function and use duk_call_method().
 label: code-design
36485. -> [... toLocaleString ToObject(val)]
36486. Note that 'l' and 'r' may cross, i.e. r < l
36487. **comment:** XXX: the insert here is a bit expensive if there are a lot of items. * It could also be special cased in the outermost for loop quite easily * (as the element is dup()'d anyway).
 label: code-design
36488. In some cases it may be that lo > hi, or hi < 0; these * degenerate cases happen e.g. for empty arrays, and in * recursion leaves.
36489. -> [... ToObject(this) ToUint32(length)]
36490. trivial cases
36491. find elements to swap
36492. args start at index 2
36493. indexOf: clamp fromIndex to [-len, len] * (if fromIndex == len, for-loop terminates directly) * * lastIndexOf: clamp fromIndex to [-len - 1, len - 1] * (if clamped to -len-1 -> fromIndex becomes -1, terminates for-loop directly)
36494. [... ToObject(this) ToUint32(length) val]
36495. **comment:** XXX: an array can have length higher than 32 bits; this is not handled * correctly now.
 label: code-design
36496. **comment:** XXX: this compiles to over 500 bytes now, even without special handling * for an array part. Uses signed ints so does not handle full array range correctly.
 label: code-design
36497. For len == 0, i is initialized to len - 1 which underflows. * The condition (i < len) will then exit the for-loop on the * first round which is correct. Similarly, loop termination * happens by i underflowing.

36498. Step 16: update length; note that the final length may be above 32 bit range * (but we checked above that this isn't the case here)
36499. stack top contains 'true'
36500. stack[0] = start * stack[1] = end * stack[2] = ToObject(this) * stack[3] = ToUint32(length) * stack[4] = result array
36501. Step 9: copy elements-to-be-deleted into the result array
36502. retval is directly usable
36503. result array is already at the top of stack
36504. has access to 'this' binding
36505. stack[0...nargs-1] = unshift args (vararg) * stack[nargs] = ToObject(this) * stack[nargs+1] = ToUint32(length)
36506. The original value needs to be preserved for filter(), hence * this funny order. We can't re-get the value because of side * effects.
36507. * splice()
36508. * isArray()
36509. The E5.1 Section 15.4.4.4 algorithm doesn't set the length explicitly * in the end, but because we're operating with an internal value which * is known to be an array, this should be equivalent.
36510. Debug print which visualizes the qsort partitioning process.
36511. for indexOf, ToInteger(undefined) would be 0, i.e. correct, but * handle both indexOf and lastIndexOf specially here.
36512. -> [sep ToObject(this) len str sep]
36513. stack[0] = compareFn * stack[1] = ToObject(this) * stack[2] = ToUint32(length)
36514. * indexOf(), lastIndexOf()
36515. move pivot out of the way
36516. **comment:** XXX: must be able to represent -len
 label: code-design
36517. * reverse()
36518. !(l < p)
36519. XXX: len >= 0x80000000 won't work below because a signed type * is needed by qsort.
36520. result index for filter()
36521. loop iterator init and limit changed from standard algorithm
36522. [... val callback thisArg val i obj]
36523. !(p < r)
36524. [ToObject(this) ToUint32(length) lowerValue upperValue]
36525. **comment:** XXX: could duk_is_undefined() provide defaulting undefined to 'len' * (the upper limit)?
 label: code-design
36526. Perform an intermediate join when this many elements have been pushed * on the value stack.
36527. **comment:** stack[0] = callback * stack[1] = thisArg * stack[2] = object * stack[3] = ToUint32(length) (unused, but avoid unnecessary pop) * stack[4] = result array
 (or undefined)
 label: code-design
36528. XXX: if 'len' is low, may want to ensure array part is kept: * the caller is likely to want a dense array.
36529. string compare is the default (a bit oddly)
36530. * pop(), push()
36531. [arg1 ... argN obj length new_length]
36532. lastIndexOf() needs to be a vararg function because we must distinguish * between an undefined fromIndex and a "not given" fromIndex; indexOf() is * made vararg for symmetry although it doesn't strictly need to be.
36533. E5.1 standard behavior when deleteCount is not given would be * to treat it just like if 'undefined' was given, which coerces * ultimately to 0. Real world behavior is to splice to the end * of array, see test-bi-array-proto-splice-no-delcount.js.
36534. no need to check callable; duk_call() will do that
36535. if thisArg not supplied, behave as if undefined was supplied
36536. * reduce(), reduceRight()
36537. [sep ToObject(this) len]
36538. NOTE: The Array special behaviors are NOT invoked by duk_xdef_prop_index() * (which differs from the official algorithm). If no error is thrown, this * doesn't matter as the length is updated at the end. However, if an error * is thrown, the length will be unset. That shouldn't matter because the * caller won't get a reference to the intermediate value.
36539. -> [... fn x y]
36540. -> [... res]
36541. stack[0] = callback fn * stack[1] = initialValue * stack[2] = object (coerced this) * stack[3] = length (not needed, but not popped above) * stack[4] = accumulator
36542. idx_step is +1 for indexOf, -1 for lastIndexOf
36543. [A B C D E F G H] rel_index = 2, del_count 3, item count 3 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)
36544. * concat()
36545. [sep ToObject(this) len sep str0 ... str(count-1)]
36546. * sort() * * Currently qsort with random pivot. This is now really, really slow, * because there is no fast path for array parts. * * Signed indices are used because qsort() leaves and degenerate cases * may use a negative offset.
36547. [... func this]
36548. stack[0] = searchElement * stack[1] = fromIndex * stack[2] = object * stack[3] = length (not needed, but not popped above)
36549. [ToUint32(len) array ToUint32(len)] -> [ToUint32(len) array]
36550. idx_step is +1 for reduce, -1 for reduceRight
36551. The extra (+4) is tight.
36552. -> [ToObject(this)]
36553. [sep ToObject(this) len sep]
36554. XXX: can shift() / unshift() use the same helper? * shift() is (close to?) <-> splice(0, 1) * unshift is (close to?) <-> splice(0, 0, [items])?
36555. stack[0] = object (this) * stack[1] = ToUint32(length) * stack[2] = elem at index 0 (retval)
36556. randomized pivot selection
36557. **comment:** XXX: optimize by creating array into correct size directly, and * operating on the array part directly; values can be memcpy()'d from * value stack directly as long as reffcounts are increased.
 label: code-design
36558. * slice()
36559. * unshift()
36560. retval to result[i]
36561. * Constructor
36562. if true, the stack already contains the final result
36563. * every(), some(), forEach(), map(), filter()
36564. * toString()
36565. Strict standard behavior, ignore trailing elements for * result 'length'.
36566. [A B C D E F G H] rel_index = 2, del_count 3, item count 4 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)
36567. fixed offset in valstack
36568. rnd in [lo,hi]
36569. Standard behavior for map(): trailing non-existent * elements don't invoke the user callback and are not * counted towards result 'length'.
36570. -> [... ToObject(this) ToUint32(length) final_len final_len]
36571. nop

36572. Technically Array.prototype.push() can create an Array with length * longer than 2^32-1, i.e. outside the 32-bit range. The final length * is *not* wrapped to 32 bits in the specification. ** This implementation tracks length with a uint32 because it's much * more practical. ** See: test-bi-array-push-maxlen.js.

36573. The specification is a bit vague what to do if the return * value is not a number. Other implementations seem to * tolerate non-numbers but e.g. V8 won't apparently do a * ToNumber().

36574. We're a varargs function because we need to detect whether * initialValue was given or not.

36575. -> [... ToObject(this) ToUInt32(length) arg[i]]

36576. [arg1 ... argN obj length]

36577. throw flag irrelevant (false in std alg)

36578. Step 15: insert itemCount elements into the hole made above

36579. For join(), nargs is 1. For toLocaleString(), nargs is 0 and * setting the top essentially pushes an undefined to the stack, * thus defaulting to a comma separator.

36580. **comment:** XXX: 'this' will be ToObject() coerced twice, which is incorrect * but should have no visible side effects.

label: code-design

36581. [A B C D E F G H] rel_index = 2, del_count 3, item count 1 * -> [A B F G H] (conceptual intermediate step) * -> [A B . F G H] (placeholder marked) * [A B C F G H] (actual result at this point, C will be replaced)

36582. If len <= 1, middle will be 0 and for-loop bails out * immediately (0 < 0 -> false).

36583. Note: unshift() may operate on indices above unsigned 32-bit range * and the final length may be >= 2**32. However, we restrict the * final result to 32-bit range for practicality.

36584. i >= 0 would always be true

36585. **comment:** Fast exit if indices are identical. This is valid for a non-existent property, * for an undefined value, and almost always for ToString() coerced comparison of * arbitrary values (corner cases where this is not the case include e.g. a an * object with varying ToString() coercion). ** The specification does not prohibit "caching" of values read from the array, so * assuming equality for comparing an index with itself falls into the category of * "caching". ** Also, compareFn may be inconsistent, so skipping a call to compareFn here may * have an effect on the final result. The specification does not require any * specific behavior for inconsistent compare functions, so again, this fast path * is OK.

label: code-design

36586. fromPresent = false

36587. -> [... x y fn]

36588. [... this func]

36589. each call this helper serves has nargs==2

36590. -> [ToObject(this) item1 ... itemN arr]

36591. [ToObject(this) item1 ... itemN arr item(i)]

36592. * join(), toLocaleString() ** Note: checking valstack is necessary, but only in the per-element loop. ** Note: the trivial approach of pushing all the elements on the value stack * and then calling duk_join() fails when the array contains a large number * of elements. This problem can't be offloaded to duk_join() because the * elements to join must be handled here and have special handling. Current * approach is to do intermediate joins with very large number of elements. * There is no fancy handling; the prefix gets re-joined multiple times.

36593. **comment:** * Array built-ins ** Note that most Array built-ins are intentionally generic and work even * when the 'this' binding is not an Array instance. To ensure this, * Array algorithms do not assume "magical" Array behavior for the "length" * property, for instance. ** XXX: the "Throw" flag should be set for (almost?) all [[Put]] and * [[Delete]] operations, but it's currently false throughout. Go through * all put/delete cases and check throw flag use. Need a new API primitive * which allows throws flag to be specified. ** XXX: array lengths above 2G won't work reliably. There are many places * where one needs a full signed 32-bit range ([-0xffffffff, 0xffffffff], * i.e. -33- bits). Although array 'length' cannot be written to be outside * the unsigned 32-bit range (E5.1 Section 15.4.5.1 throws a RangeError if so) * some intermediate values may be above 0xffffffff and this may not be always * correctly handled now (duk_uint32_t is not enough for all algorithms). ** For instance, push() can legitimately write entries beyond length 0xffffffff * and cause a RangeError only at the end. To do this properly, the current * push() implementation tracks the array index using a 'double' instead of a * duk_uint32_t (which is somewhat awkward). See test-bi-array-push-maxlen.js. ** On using "put" vs. "def" prop * ===== * Code below must be careful to use the appropriate primitive as it matters * for compliance. When using "put" there may be inherited properties in * Array.prototype which cause side effects when values are written. When * using "define" there are no such side effects, and many test262 test cases * check for this (for real world code, such side effects are very rare). * Both "put" and "define" are used in the E5.1 specification; as a rule, * "put" is used when modifying an existing array (or a non-array 'this' * binding) and "define" for setting values into a fresh result array. ** Also note that Array instance 'length' should be writable, but not * enumerable and definitely not configurable: even Duktape code internally * assumes that an Array instance will always have a 'length' property. * Preventing deletion of the property is critical.

label: code-design

36594. XXX: proper flags?

36595. * Boolean built-ins

36596. unbalanced stack

36597. -> [val obj val]

36598. **comment:** XXX: there is room to use a shared helper here, many built-ins * check the 'this' type, and if it's an object, check its class, * then get its internal value, etc.

label: code-design

36599. Shared helper to provide toString() and valueOf(). Checks 'this', gets * the primitive value to stack top, and optionally coerces with ToString().

36600. **comment:** XXX: helper; rely on Boolean.prototype as being non-writable, non-configurable

label: code-design

36601. * Duktape.Buffer: toString(), valueOf()

36602. [value offset noAssert], when ftype != DUK_FLD_VARINT

36603. reachable so pop OK

36604. Khronos/ES6 requires zeroing even when DUK_USE_ZERO_BUFFER_DATA * is not set.

36605. throw_flag

36606. **comment:** XXX: encoding is ignored now.

label: code-design

36607. This behavior mostly mimics Node.js now.

36608. nargs

36609. Fast path: source is a TypedArray (or any bufferobject).

36610. NOTE! Caller must ensure that any side effects from the * coercions below are safe. If that cannot be guaranteed * (which is normally the case), caller must coerce the * argument using duk_to_number() before any pointer * validations; the result of duk_to_number() always coerces * without side effects here.

36611. Don't accept relative indices now.

36612. bits 0...1: shift

36613. default is explicit index read/write copy

36614. Clamp to target's end if too long. ** NOTE: there's no overflow possibility in the comparison; * both target_ustart and copy_size are >= 0 and based on * values in duk_int_t range. Adding them as duk_uint_t * values is then guaranteed not to overflow.

36615. [offset noAssert], when ftype != DUK_FLD_VARINT

36616. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT32

36617. Select copy mode. Must take into account element * compatibility and validity of the underlying source * buffer.

36618. For the case n==1 Node.js doesn't seem to type check * the sole member but we do it before returning it. * For this case only the original buffer object is * returned (not a copy).

36619. Push the resulting view object and attach the ArrayBuffer.

36620. DUK_FLD_32BIT

36621. >= 0

36622. arg count

36623. DUK_FLD_8BIT

36624. Neutered. We could go into the switch-case safely with * buf == NULL because check_length == 0. To avoid scanbuild * warnings, fail directly instead.

36625. * TypedArray.prototype.set() ** TypedArray set() is pretty interesting to implement because: ** - The source argument may be a plain array or a typedarray. If the * source is a TypedArray, values are decoded and re-encoded into the * target (not as a plain byte copy). This may happen even when the * element byte size is the same, e.g. integer values may be re-encoded * into floats. ** - Source and target may refer to the same underlying buffer, so that * the set() operation may overlap. The specification requires that this * must work as if a copy was made before the operation. Note that this * is NOT a simple memmove() situation because the source and target * byte sizes may be different -- e.g. a 4-byte source (Int8Array) may * expand to a 16-byte target (Uint32Array) so that the target overlaps * the source both from beginning and the end (unlike in typical memmove). ** - Even if 'buf' pointers of the source and target differ, there's no * guarantee that their memory areas don't overlap. This may be the * case with external buffers. ** Even so, it is nice to optimize for the common case: ** - Source and target separate buffers or non-overlapping. ** - Source and target have a compatible type so that a plain byte copy * is possible. Note that while e.g. uint8 and int8 are compatible * (coercion one way or another doesn't change the byte representation), * e.g. int8 and uint8clamped are NOT compatible when writing int8 * values into uint8clamped typedarray (-1 would clamp to 0 for instance). ** See test-bi-typedarray-proto-set.js.

36626. Non-object argument is simply int coerced, matches * V8 behavior (except for "null", which we coerce to * 0 but V8 TypeErrors).

36627. Array or Array-like

36628. **comment:** Static call style.
label: code-design

36629. [offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT

36630. sign doesn't matter when writing

36631. Format of magic, bits: * 0...1: elem size shift (0-3) * 2...5: elem type (DUK_HBUFFEROBJECT_ELEM_XXX)

36632. **comment:** Slow gathering of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. Handling * of negative numbers is a bit non-obvious in both cases.
label: code-design

36633. Resolve start/end offset as element indices first; arguments * at idx_start/idx_end are element offsets. Working with element * indices first also avoids potential for wrapping.

36634. Write in little endian

36635. [targetBuffer targetStart sourceStart sourceEnd]

36636. source starts after dest ends

36637. * Misc helpers

36638. For TypedArrays 'undefined' return value is specified * by ES6 (matches V8).

36639. Check that value is a duk_hbufferobject and return a pointer to it.

36640. **comment:** Fill offset handling is more lenient than in Node.js.
label: code-design

36641. byte sizes will match

36642. return h_bufres

36643. Must be element size multiple from * start offset to end of buffer.

36644. * Node.js Buffer: toString([encoding], [start], [end])

36645. We want to do a straight memory copy if possible: this is * an important operation because .set() is the TypedArray * way to copy chunks of memory. However, because set() * conceptually works in terms of elements, not all views are * compatible with direct byte copying. ** If we do manage a direct copy, the "overlap issue" handled * below can just be solved using memmove() because the source * and destination element sizes are necessarily equal.

36646. new buffer with string contents

36647. XXX: could add fast path for u8 compatible views

36648. A validated read() is always a number, so it's write coercion * is always side effect free and won't invalidate pointers etc.

36649. Slice offsets are element (not byte) offsets, which only matters * for TypedArray views, Node.js Buffer and ArrayBuffer have shift * zero so byte and element offsets are the same. Negative indices * are counted from end of slice, crossed indices are allowed (and * result in zero length result), and final values are clamped * against the current slice. There's intentionally no check * against the underlying buffer here.

36650. We could fit built-in index into magic but that'd make the magic * number dependent on built-in numbering (genbuiltins.py doesn't * handle that yet). So map both class and prototype from the * element type.

36651. bytes in dest

36652. Use byte copy.

36653. Argument variants. When the argument is an ArrayBuffer a view to * the same buffer is created; otherwise a new ArrayBuffer is always * created.

36654. Accept plain buffer values like array initializers * (new in Duktape 1.4.0).

36655. Shared lenient buffer length clamping helper. Indices are treated as * element indices (though output values are byte offsets) which only * really matters for TypedArray views as other buffer object have a zero * shift. Negative indices are counted from end of input slice; crossed * indices are clamped to zero length; and final indices are clamped * against input slice. Used for e.g. ArrayBuffer slice().

36656. 0xff => 255 - 256 = -1; 0x80 => 128 - 256 = -128

36657. DUK_FLD_16BIT

36658. Update 'buffer_length' to be the effective, safe limit which * takes into account the underlying buffer. This value will be * potentially invalidated by any side effect.

36659. xxx -> DUK_HBUFFEROBJECT_ELEM_INT32

36660. avoid side effects!

36661. Node.js accepts only actual Arrays.

36662. **comment:** XXX: regetting the pointer may be overkill - we're writing * to a side-effect free array here.
label: code-design

36663. crossed offsets or zero size

36664. Must use memmove() because copy area may overlap (source and target * buffer may be the same, or from different slices).

36665. Gather in little endian

36666. unsigned

36667. We need 'nbytes' even for a failed offset; return value must be * (offset + nbytes) even when write fails due to invalid offset.

36668. **comment:** XXX: function flag to make this automatic?
label: requirement

36669. XXX: happens e.g. when evaluating: String(Buffer.prototype).

36670. For all duk_hbufferobjects, get the plain buffer inside * without making a copy. This is compatible with Duktape 1.2 * but means that a slice/view information is ignored and the * full underlying buffer is returned. ** If called as a constructor, a new Duktape.Buffer object * pointing to the same plain buffer is created below.

36671. DUK_FLD_VARINT; not relevant here

36672. Shared helper.

36673. **comment:** ArrayBuffer argument is handled specially above; the rest of the * argument variants are handled by shared code below.
label: code-design

36674. Copy values by index reads and writes. Let virtual * property handling take care of coercion.

36675. Accept any duk_hbufferobject, though we're only normally * called for Duktape.Buffer values.

36676. idx_end

36677. may be NULL

36678. -> [buffer]

36679. [offset value littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)

36680. Node.js return value for noAssert out-of-bounds reads is * usually (but not always) NaN. Return NaN consistently.

36681. Copy values through direct validated reads and writes.

36682. Return the Buffer to allow chaining: b.fill(0x11).fill(0x22, 3, 5).toString()

36683. Gather in big endian

36684. compare: similar to string comparison but for buffer data.

36685. **comment:** * Node.js Buffer.prototype.equals() * Node.js Buffer.prototype.compare() * Node.js Buffer.compare()
label: requirement

36686. use p_src_base from now on

36687. ignore encoding for now
36688. * Node.js Buffer.isBuffer()
36689. * Node.js Buffer.prototype.fill()
36690. * Duktape.Buffer, Node.js Buffer, and Khronos/ES6 TypedArray built-ins
36691. * Node.js Buffer.isEncoding()
36692. TypedArray (or other non-ArrayBuffer duk_hbufferobject). * Conceptually same behavior as for an Array-like argument, * with a few fast paths.
36693. 1=little endian
36694. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT64
36695. * Node.js Buffer.prototype.write(string, [offset], [length], [encoding])
36696. Not enough data.
36697. Note: start_offset/end_offset can still be < 0 here.
36698. bits 2...5: type
36699. **comment:** XXX: V8 throws a TypeError for negative values. Would it * be more useful to interpret negative offsets here from the * end of the buffer too?
label: code-design
36700. Push a new ArrayBuffer (becomes view .buffer)
36701. Overflow not an issue because subtraction is used on the right * side and guaranteed to be ≥ 0 .
36702. idx_start
36703. should never happen but default here
36704. Now we can check offset validity.
36705. Slow path: quite slow, but we save space by using the property code * to write coerce target values. We don't need to worry about overlap * here because the source is not a TypedArray. * * We could use the bufferobject write coercion helper but since the * property read may have arbitrary side effects, full validity checks * would be needed for every element anyway.
36706. [value offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT
36707. inherit
36708. * Node.js Buffer.prototype.copy()
36709. **comment:** not covered, return all zeroes
label: test
36710. [value offset end]
36711. **comment:** * Node.js Buffer.concat()
label: code-design
36712. Value stack intentionally mixed size here.
36713. [array totalLength]
36714. * Shared readfield and writefield methods * * The readfield/writefield methods need support for endianness and field * types. All offsets are byte based so no offset shifting is needed.
36715. not reachable
36716. * Node.js Buffer.prototype.toJSON()
36717. Check actual underlying buffers for validity and that they * cover the copy. No side effects are allowed after the check * so that the validity status doesn't change.
36718. length check
36719. We want to avoid making a copy to process set() but that's * not always possible: the source and the target may overlap * and because element sizes are different, the overlap cannot * always be handled with a memmove() or choosing the copy * direction in a certain way. For example, if source type is * uint8 and target type is uint32, the target area may exceed * the source area from both ends! * * Note that because external buffers may point to the same * memory areas, we must ultimately make this check using * pointers. * * NOTE: careful with side effects: any side effect may cause * a buffer resize (or external buffer pointer/length update!).
36720. There's no need to check for buffer validity status for the * target here: the property access code will do that for each * element. Moreover, if we did check the validity here, side * effects from reading the source argument might invalidate * the results anyway.
36721. Must be ≥ 0 and multiple of element size.
36722. * Duktape.Buffer: constructor
36723. Ready to make the copy. We must proceed element by element * and must avoid any side effects that might cause the buffer * validity check above to become invalid. * * Although we work through the value stack here, only plain * numbers are handled which should be side effect safe.
36724. * Constructor arguments are currently somewhat compatible with * (keep it that way if possible): * * <http://nodejs.org/api/buffer.html> * * Note that the ToBuffer() coercion (duk_to_buffer()) does NOT match * the constructor behavior.
36725. Handle single character fills as memset() even when * the fill data comes from a one-char argument.
36726. Just skip, leaving zeroes in the result.
36727. Equality may be OK but >length not. Checking * this explicitly avoids some overflow cases * below.
36728. Byte length would overflow.
36729. Set .buffer
36730. equals
36731. We want to compare the slice/view areas of the arguments. * If either slice/view is invalid (underlying buffer is shorter) * ensure equals() is false, but otherwise the only thing that * matters is to be memory safe.
36732. **comment:** Format of magic, bits: * ...1: field type; 0:uint8, 1:uint16, 2:uint32, 3=float, 4=double, 5=unused, 6=unused, 7=unused * 3: endianness: 0=little, 1=big * 4: signed: 1=yes, 0=no * 5: typedarray: 1=yes, 0=no
label: code-design
36733. Shared offset/length coercion helper.
36734. bytes in source
36735. Source end clamped silently to available length.
36736. **comment:** Internal class is Object: Object.prototype.toString.call(new Buffer(0)) * prints "[object Object]".
label: code-design
36737. elems in source and dest
36738. XXX: fast path for array arguments?
36739. either nonzero value is ok
36740. Handle TypedArray vs. Node.js Buffer arg differences
36741. Negative offsets cause a RangeError.
36742. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8CLAMPED * Note: INT8 is -not- copy compatible, e.g. -1 would coerce to 0x00.
36743. Shared lenient buffer length clamping helper. No negative indices, no * element/byte shifting.
36744. * Node.js Buffer.prototype.slice([start], [end]) * ArrayBuffer.prototype.slice(begin, [end]) * * The API calls are almost identical; negative indices are counted from end * of buffer, and final indices are clamped (allowing crossed indices). Main * differences: * * - Copy/view behavior; Node.js .slice() and TypedArray .subarray() create * views, ArrayBuffer .slice() creates a copy * * - Resulting object has a different class and prototype depending on the * call (or 'this' argument) * * - TypedArray .subarray() arguments are element indices, not byte offsets
36745. offset/value order different from Node.js
36746. Node.js Buffer: return offset + #bytes written (i.e. next * write offset).
36747. Check that 'this' is a duk_hbufferobject and return a pointer to it * (NULL if not).
36748. ArrayBuffer: unlike any other argument variant, create * a view into the existing buffer.
36749. one i_step added at top of loop
36750. **comment:** unnecessary div for last byte
label: code-design
36751. **comment:** Serialize uncovered backing buffer as a null; doesn't * really matter as long we're memory safe.
label: code-design
36752. Ensure copy is covered by underlying buffers.

36753. At the moment Buffer(<str>) will just use the string bytes as * is (ignoring encoding), so we return the string length here * unconditionally.
36754. **comment:** Node.js Buffer variable width integer field. We don't really * care about speed here, so aim for shortest algorithm.
 label: code-design
36755. **comment:** XXX: accept any duk_hbufferobject type as an input also?
 label: code-design
36756. User totalLength overrides a computed length, but we'll check * every copy in the copy loop. Note that duk_to_uint() can * technically have arbitrary side effects so we need to recheck * the buffers in the copy loop.
36757. * ArrayBuffer, DataView, and TypedArray constructors
36758. guaranteed by duk_to_string()
36759. Map DUK_HBUFFEROBJECT_ELEM_xxx to prototype object built-in index. * Sync with duk_hbufferobject.h.
36760. -> [array totalLength buf]
36761. source ends before dest starts
36762. Bitfield for each DUK_HBUFFEROBJECT_ELEM_xxx indicating which element types * are compatible with a blind byte copy for the TypedArray set() method (also * used for TypedArray constructor). Array index is target buffer elem type, * bitfield indicates compatible source types. The types must have same byte * size and they must be coercion compatible.
36763. source out-of-bounds (but positive)
36764. Map DUK_FLX_xxx to byte size.
36765. **comment:** The DataView .buffer property is ordinarily set to the argument * which is an ArrayBuffer. We accept any duk_hbufferobject as * an argument and .buffer will be set to the argument regardless * of what it is. This may be a bit confusing if the argument * is e.g. a DataView or another TypedArray view. * * XXX: Copy .buffer property from a DataView/TypedArray argument? * Create a fresh ArrayBuffer for Duktape.Buffer and Node.js Buffer * arguments? See: test-bug-dataview-buffer-prop.js.
 label: code-design
36766. **comment:** XXX: split into separate functions for each field type?
 label: code-design
36767. [start end]
36768. [array totalLength bufres buf]
36769. Offset is coerced first to signed integer range and then to unsigned. * This ensures we can add a small byte length (1-8) to the offset in * bound checks and not wrap.
36770. one i_step over
36771. default to false
36772. Unlike for negative arguments, some call sites * want length to be clamped if it's positive.
36773. For n == 0, Node.js ignores totalLength argument and * returns a zero length buffer.
36774. Return value is like write(), number of bytes written. * The return value matters because of code like: * "off += buf.copy(...)".
36775. xxx -> DUK_HBUFFEROBJECT_ELEM_INT8
36776. xxx -> DUK_HBUFFEROBJECT_ELEM_INT16
36777. These are not needed when only Duktape.Buffer is supported.
36778. DUK_FLD_FLOAT
36779. undefined coerces to zero which is correct
36780. Copy values, the copy method depends on the arguments. * * Copy mode decision may depend on the validity of the underlying * buffer of the source argument; there must be no harmful side effects * from there to here for copy_mode to still be valid.
36781. DUK_FLD_DOUBLE
36782. Map DUK_HBUFFEROBJECT_ELEM_xxx to duk_hobject class number. * Sync with duk_hbufferobject.h and duk_hobject.h.
36783. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT32
36784. [offset littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)
36785. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8
36786. even for zero-length string
36787. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT16
36788. new buffer of specified size
36789. pop plain buffer, now reachable through h_bufres
36790. Node.js return value for failed writes is offset + #bytes * that would have been written.
36791. DUK_USE_BUFFEROBJECT_SUPPORT
36792. Custom behavior: plain buffer is used as internal buffer * without making a copy (matches Duktape.Buffer).
36793. ignore_loop
36794. Shift to sign extend.
36795. No copy, leave zero bytes in the buffer. There's no * ambiguity with Float32/Float64 because zero bytes also * represent 0.0.
36796. **comment:** XXX: for negative input offsets, 'offset' will be a large * positive value so the result here is confusing.
 label: code-design
36797. no overflow
36798. **comment:** Copy the .buffer property, needed for TypedArray.prototype.subarray(). * * XXX: limit copy only for TypedArray classes specifically?
 label: code-design
36799. stack is unbalanced, but: [<something> buf]
36800. **comment:** The condition could be more narrow and check for the * copy area only, but there's no need for fine grained * behavior when the underlying buffer is misconfigured.
 label: code-design
36801. Set .buffer to the argument ArrayBuffer.
36802. **comment:** unnecessary shift for last byte
 label: code-design
36803. **comment:** Slow writing of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. There's * no special sign handling when writing variants.
 label: code-design
36804. * ArrayBuffer.isView()
36805. **comment:** XXX: easier check with less code?
 label: code-design
36806. **comment:** The resulting buffer object gets the same class and prototype as * the buffer in 'this', e.g. if the input is a Node.js Buffer the * result is a Node.js Buffer; if the input is a Float32Array, the * result is a Float32Array. * * For the class number this seems correct. The internal prototype * is not so clear: if 'this' is a bufferobject with a non-standard * prototype object, that value gets copied over into the result * (instead of using the standard prototype for that object type).
 label: code-design
36807. return input buffer, converted to a Duktape.Buffer object * if called as a constructor (no change if called as a * function).
36808. * Indexed read/write helpers (also used from outside this file)
36809. * Node.js Buffer: constructor
36810. Write in big endian
36811. replaced
36812. * Node.js Buffer.byteLength()
36813. non-zero: make copy
36814. There's overlap: the desired end result is that * conceptually a copy is made to avoid "trampling" * of source data by destination writes. We make * an actual temporary copy to handle this case.
36815. Note: unbalanced stack on purpose
36816. Copy slice, respecting underlying buffer limits; remainder * is left as zero.

36817. Nominal size check.
36818. Coerce value to a number before computing check_length, so that * the field type specific coercion below can't have side effects * that would invalidate check_length.
36819. **comment:** Neutered checks not necessary here: neutered buffers have * zero 'length' so we'll effectively skip them.
label: code-design
36820. Check that 'this' is a duk_hbufferobject and return a pointer to it.
36821. **comment:** XXX: The duk_to_number() cast followed by integer coercion * is platform specific so NaN, +/- Infinity, and out-of-bounds * values result in platform specific output now. * See: test-bi-nodejs-buffer-proto-varint-special.js
label: code-design
36822. as elements
36823. Argument must be a string, e.g. a buffer is not allowed.
36824. target out-of-bounds (but positive)
36825. unbalanced stack on purpose
36826. Compute result length and validate argument buffers.
36827. only accept lowercase 'utf8' now.
36828. Cannot overlap.
36829. Convert indices to byte offsets.
36830. no need to decref
36831. [... this]
36832. stack top: new time value, return 1 to allow tail calls
36833. Convert day from one-based to zero-based (internal). This may * cause the day part to be negative, which is OK.
36834. 3
36835. Apply timezone offset to get the main parts in UTC
36836. **comment:** XXX: there is a small risk here: because the ISO 8601 parser is * very loose, it may end up parsing some datetime values which * would be better parsed with a platform specific parser.
label: code-design
36837. 8: getFullYear
36838. now expected to fit into a 32-bit integer
36839. All components default to 0 except day-of-month which defaults * to 1. However, because our internal day-of-month is zero-based, * it also defaults to zero here.
36840. MakeDate
36841. 37: setUTCFullYear
36842. * Return (t - LocalTime(t)) in minutes: * * t - LocalTime(t) = t - (t + LocalTZA + DaylightSavingTA(t)) * = -(LocalTZA + DaylightSavingTA(t)) * * where DaylightSavingTA() is checked for time 't'. * * Note that the sign of the result is opposite to common usage, * e.g. for EE(S)T which normally is +2h or +3h from UTC, this * function returns -120 or -180. *
36843. Integer division which floors also negative values correctly.
36844. -"-"
36845. invalid value which never matches
36846. 7
36847. estimate year upwards (towards positive infinity), then back down; * two iterations should be enough
36848. Helper for string conversion calls: check 'this' binding, get the * internal time value, and format date and/or time in a few formats. * Return value allows tail calls.
36849. Here we'd have the option to normalize -0 to +0.
36850. -> [O toISOString O]
36851. -> [timeval this timeval]
36852. Optional UTC conversion.
36853. Note: %06d for positive value, %07d for negative value to include * sign and 6 digits.
36854. Internal timevalue is already NaN, so don't touch it.
36855. 5: toLocaleTimeString
36856. Old solution: don't iterate, incorrect
36857. 38: getYear
36858. 6
36859. * Other file level defines
36860. zero-based -> one-based
36861. called as a normal function: return new Date().toString()
36862. * ISO 8601 subset parser.
36863. 39: setYear
36864. Rule control flags.
36865. 1: toDateString
36866. E5.1 Section 15.9.1.6
36867. tzoffset seconds are dropped; 16 bits suffice for * time offset in minutes
36868. if-digit-else-ctrl
36869. Causes a ToNumber() coercion, but doesn't break coercion order since * year is coerced first anyway.
36870. Unlike most built-ins, the internal [[PrimitiveValue]] of a Date * is mutable.
36871. Parts are in local time, convert when setting.
36872. **comment:** * String/JSON conversions * * Human readable conversions are now basically ISO 8601 with a space * (instead of 'T') as the date/time separator. This is a good baseline * and is platform independent. * * A shared native helper to provide many conversions. Magic value contains * a set of flags. The helper provides: * * toString() * toDateString() * toTimeString() * toLocaleString() * toLocaleDateString() * toLocaleTimeString() * toUTCString() * toISOString() * * Notes: * * - Date.prototype.toGMTString() and Date.prototype.toUTCString() are * required to be the same EcmaScript function object (!), so it is * omitted from here. * * - Date.prototype.toUTCString(): E5.1 specification does not require a * specific format, but result should be human readable. The * specification suggests using ISO 8601 format with a space (instead of 'T') separator if a more human readable format is not available. * * - Date.prototype.toISOString(): unlike other conversion functions, * toISOString() requires a RangeError for invalid date values.
label: code-design
36873. -> [... this]
36874. 16: getHours
36875. Note2
36876. char loop
36877. 33: setUTCDate
36878. 25: setUTCMilliseconds
36879. Use explicit steps in computation to try to ensure that * computation happens with intermediate results coerced to * double values (instead of using something more accurate). * E.g. E5.1 Section 15.9.1.11 requires use of IEEE 754 * rules (= EcmaScript '+' and '*' operators). * * Without 'volatile' even this approach fails on some platform * and compiler combinations. For instance, gcc 4.8.1 on Ubuntu * 64-bit, with -m32 and without -std=c99, test-bi-date-canceling.js * would fail because of some optimizations when computing tmp_time * (MakeTime below). Adding 'volatile' to tmp_time solved this * particular problem (annoyingly, also adding debug prints or * running the executable under valgrind hides it).
36880. match against rule part/sep bits
36881. Coerce all finite parts with ToInteger(). ToInteger() must not * be called for NaN/Infinity because it will convert e.g. NaN to * zero. If ToInteger() has already been called, this has no side * effects and is idempotent. * * Don't read dparts[DUK_DATE_IDX_WEEKDAY]; it will cause Valgrind * issues if the value is uninitialized.
36882. NaN timevalue: we need to coerce the arguments, but * the resulting internal timestamp needs to remain NaN. * This works but is not pretty: parts and dparts will * be partially uninitialized, but we only write to them.

36883. Rule table: first matching rule is used to determine what to do next.
36884. flags
36885. fits into duk_small_int_t
36886. 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.
36887. formatters always get one-based month/day-of-month
36888. Setter APIs detect special year numbers (0...99) and apply a +1900 * only in certain cases. The legacy getYear() getter applies -1900 * unconditionally.
36889. 2: toTimeString
36890. **comment:** TimeClip() should never be necessary
label: code-design
36891. continue matching, set neg_tzoffset flag
36892. -> [... this timeval_new]
36893. 30: setHours
36894. TimeClip(), which also handles Infinity -> NaN conversion
36895. unsigned
36896. zero-based day
36897. 34: setMonth
36898. Note: rely on index ordering
36899. Parser part masks.
36900. 20: getSeconds
36901. Expects 'this' at top of stack on entry.
36902. 28: setMinutes
36903. t1 = milliseconds within day (fits 32 bit) * t2 = day number from epoch (fits 32 bit, may be negative)
36904. * Indirect magic value lookup for Date methods. ** Date methods don't put their control flags into the function magic value * because they wouldn't fit into a LIGHTFUNC's magic field. Instead, the * magic value is set to an index pointing to the array of control flags * below. ** This must be kept in strict sync with genbuiltins.py!
36905. Day-of-month is one-based in the API, but zero-based * internally, so fix here. Note that month is zero-based * both in the API and internally.
36906. 1
36907. * Determining which datetime components to overwrite based on * stack arguments is a bit complicated, but important to factor * out from setters themselves for compactness. ** If DUK_DATE_FLAG_TIMESETTER, maxnargs indicates setter type: ** 1 -> millisecond * 2 -> second, [millisecond] * 3 -> minute, [second], [millisecond] * 4 -> hour, [minute], [second], [millisecond] ** Else: ** 1 -> date * 2 -> month, [date] * 3 -> year, [month], [date] ** By comparing nargs and maxnargs (and flags) we know which * components to override. We rely on part index ordering.
36908. zero-based month
36909. Parser part count.
36910. 6: toUTCString
36911. no argument given -> leave components untouched
36912. msec
36913. Compute time value from (double) parts. The parts can be either UTC * or local time; if local, they need to be (conceptually) converted into * UTC time. The parts may represent valid or invalid time, and may be * wildly out of range (but may cancel each other and still come out in * the valid Date range).
36914. Matching separator index is used in the control table
36915. don't care value, year is mandatory
36916. **comment:** * Date built-ins ** Unlike most built-ins, Date has some platform dependencies for getting * UTC time, converting between UTC and local time, and parsing and * formatting time values. These are all abstracted behind DUK_USE_xxx * config options. There are built-in platform specific providers for * POSIX and Windows, but external providers can also be used. ** See doc/datetime.rst.
label: code-design
36917. SCANBUILD: complains about use of uninitialized values. * The complaint is correct, but operating in undefined * values here is intentional in some cases and the caller * ignores the results.
36918. 27: setUTCSeconds
36919. unreferenced with some options
36920. 36: setFullYear
36921. -> [this]
36922. Leaves new timevalue on stack top and returns 1, which is correct * for part setters.
36923. 4
36924. 21: getUTCSeconds
36925. seconds, doesn't fit into 16 bits
36926. Equivalent year for DST calculations outside [1970,2038[range, see * E5 Section 15.9.1.8. Equivalent year has the same leap-year-ness and * starts with the same weekday on Jan 1. * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
36927. may be NaN
36928. 3: toLocaleString
36929. 17: getUTCHours
36930. 22: getMilliseconds
36931. Set datetime parts from stack arguments, defaulting any missing values. * Day-of-week is not set; it is not required when setting the time value.
36932. 12: getDate
36933. There are at most 7 args, but we use 8 here so that also * DUK_DATE_IDX_WEEKDAY gets initialized (to zero) to avoid the potential * for any Valgrind gripes later.
36934. 35: setUTCMonth
36935. Special handling for year sign.
36936. caller checks
36937. 24: setMilliseconds
36938. MakeTime
36939. 0x08583b00
36940. 5
36941. If 'day' is NaN, returns NaN.
36942. **comment:** DUK_USE_DATE_GET_LOCAL_TZOFFSET() needs to be called with a * time value computed from UTC parts. At this point we only * have 'd' which is a time value computed from local parts, so * it is off by the UTC-to-local time offset which we don't know * yet. The current solution for computing the UTC-to-local * time offset is to iterate a few times and detect a fixed * point or a two-cycle loop (or a sanity iteration limit), * see test-bi-date-local-parts.js and test-bi-date-tzoffset-basic-fi.js. ** E5.1 Section 15.9.1.9: * UTC(t) = t - LocalTZA - DaylightSavingTA(t - LocalTZA) * * For NaN/inf, DUK_USE_DATE_GET_LOCAL_TZOFFSET() returns 0.
label: code-design
36943. 9: getUTCFullYear
36944. During parsing, month and day are one-based; set defaults here.
36945. The algorithm in E5.1 Section 15.9.1.12 normalizes month, but * does not normalize the day-of-month (nor check whether or not * it is finite) because it's not necessary for finding the day * number which matches the (year,month) pair. ** We assume that duk_day_from_year() is exact here. ** Without an explicit infinity / NaN check in the beginning, * day_num would be a bogus integer here. ** It's possible for 'year' to be out of integer range here. * If so, we need to return NaN without integer overflow. * This fixes test-bug-setyear-overflow.js.
36946. MakeDay
36947. Push 'this' binding, check that it is a Date object; then push the * internal time value. At the end, stack is: [... this timeval]. * Returns the time value. Local time adjustment is done if requested.

36948. Helper for component getter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), push a specified component as a return value to the * value stack and return 1 (caller can then tail call us).

36949. Different calling convention than above used because the helper * is shared.

36950. non-leap year: sunday, monday, ...

36951. Note1: the specification doesn't require matching a time form with * just hours ("HH"), but we accept it here, e.g. "2012-01-02T12Z". ** Note2: the specification doesn't require matching a timezone offset * with just hours ("HH"), but accept it here, e.g. "2012-01-02T03:04:05+02"

36952. * API oriented helpers

36953. 11: getUTCMonth

36954. leap year: sunday, monday, ...

36955. **comment:** This shouldn't be necessary, but check just in case * to avoid any chance of overruns.
label: code-design

36956. Note: DST adjustment is determined using UTC time.

36957. handle negative values

36958. 32: setDate

36959. Note: DST adjustment is determined using UTC time. * If 'd' is NaN, tzoffset will be 0.

36960. **comment:** XXX: idx_val would fit into 16 bits, but using duk_small_uint_t * might not generate better code due to casting.
label: code-design

36961. 7: toISOString

36962. **comment:** * Setters. ** Setters are a bit more complicated than getters. Component setters * break down the current time value into its (normalized) component * parts, replace one or more components with -unnormalized- new values, * and the components are then converted back into a time value. As an * example of using unnormalized values: ** var d = new Date(1234567890); ** is equivalent to: ** var d = new Date(0); * d.setUTCSeconds(1234567890); ** A shared native helper to provide almost all setters. Magic value * contains a set of flags and also packs the "maxnargs" argument. The * helper provides: ** setMilliseconds() * setUTCSeconds() * setUTCMinutes() * setUTCHours() * setDate() * setUTCDate() * setMonth() * setUTCMonth() * setFullYear() * setUTCFullYear() * setYear() ** Notes: ** - Date.prototype.setYear() (Section B addition): special year check * is omitted. NaN / Infinity will just flow through and ultimately * result in a NaN internal time value. ** - Date.prototype.setYear() does not have optional arguments for * setting month and day-in-month (like setFullYear()), but we indicate * 'maxnargs' to be 3 to get the year written to the correct component * index in duk_set_part_helper(). The function has nargs == 1, so only * the year will be set regardless of actual argument count.
label: code-design

36963. 13: getUTCDate

36964. No locale specific formatter; this is OK, we fall back * to ISO 8601.

36965. 2

36966. Maximum iteration count for computing UTC-to-local time offset when * creating an Ecmascript time value from local parts.

36967. Assume that year, month, day are all coerced to whole numbers. * They may also be NaN or infinity, in which case this function * must return NaN or infinity to ensure time value becomes NaN. * If 'day' is NaN, the final return will end up returning a NaN, * so it doesn't need to be checked here.

36968. 23: getUTCSeconds

36969. -> [timeval this]

36970. We should never exit the loop above.

36971. e.g. a = -4, b = 5 --> -4 - 5 + 1 / 5 --> -8 / 5 --> -1 * a = -5, b = 5 --> -5 - 5 + 1 / 5 --> -9 / 5 --> -1 * a = -6, b = 5 --> -6 - 5 + 1 / 5 --> -10 / 5 --> -2

36972. Parser separator indices.

36973. -> [... this timeval_new timeval_new]

36974. "-1234560"

36975. rule match

36976. deal with negative values

36977. 10: getMonth

36978. * Calendar helpers ** Some helpers are used for getters and can operate on normalized values * which can be represented with 32-bit signed integers. Other helpers are * needed by setters and operate on un-normalized double values, must watch * out for non-finite numbers etc.

36979. **comment:** Debug macro to print all parts and dparts (used manually because of debug level).
label: code-design

36980. Unlike year, the other parts fit into 16 bits so %d format * is portable.

36981. * Date/time parsing helper. ** Parse a datetime string into a time value. We must first try to parse * the input according to the standard format in E5.1 Section 15.9.1.15. * If that fails, we can try to parse using custom parsing, which can * either be platform neutral (custom code) or platform specific (using * existing platform API calls). ** Note in particular that we must parse whatever toString(), toUTCString(), * and toISOString() can produce; see E5.1 Section 15.9.4.2. ** Returns 1 to allow tail calling. ** There is much room for improvement here with respect to supporting * alternative datetime formats. For instance, V8 parses '2012-01-01' as * UTC and '2012/01/01' as local time.

36982. unpack args

36983. Set timeval to 'this' from dparts, push the new time value onto the * value stack and return 1 (caller can then tail call us). Expects * the value stack to contain 'this' on the stack top.

36984. Given a day number, determine year and day-within-year.

36985. Parser separator masks.

36986. Equivalent year mapping, used to avoid DST trouble when platform * may fail to provide reasonable DST answers for dates outside the * ordinary range (e.g. 1970-2038). An equivalent year has the same * leap-year-ness as the original year and begins on the same weekday * (Jan 1). ** The year 2038 is avoided because there seem to be problems with it * on some platforms. The year 1970 is also avoided as there were * practical problems with it; an equivalent year is used for it too, * which breaks some DST computations for 1970 right now, see e.g. * test-bi-date-tzoffset-brute-fi.js.

36987. * Helper to format a time value into caller buffer, used by logging. * 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.

36988. * Constructor calls

36989. Helper for component setter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), modify one or more components as specified, recompute * the time value, set it as the internal value. Finally, push the * new time value as a return value to the value stack and return 1 * (caller can then tail call us).

36990. ignore millisecond fractions after 3

36991. try locale specific formatter; if it refuses to format the * string, fall back to an ISO 8601 formatted value in local * time.

36992. 15: getUTCDay

36993. This assert depends on the input parts representing time inside * the Ecmascript range.

36994. **comment:** XXX: getter with class check, useful in built-ins
label: code-design

36995. get as double to handle huge numbers correctly

36996. these computations are guaranteed to be exact for the valid * E5 time value range, assuming milliseconds without fractions.

36997. if no NaN handling flag, may still be NaN here, but not Inf

36998. 0: toString

36999. 26: setSeconds

37000. Split time value into parts. The time value is assumed to be an internal * one, i.e. finite, no fractions. Possible local time adjustment has already * been applied when reading the time value.

37001. Because 'day since epoch' can be negative and is used to compute weekday * using a modulo operation, add this multiple of 7 to avoid negative values * when year is below 1970 epoch. Ecmascript time values are restricted to * +/- 100 million days from epoch, so this adder fits nicely into 32 bits. * Round to a multiple of 7 (= floor(100000000 / 7) * 7) and add margin.

37002. Contract, either: * - Push string to value stack and return 1 * - Don't push anything and return 0

37003. 19: getUTCMonth

37004. 14: getDay

37005. **comment:** Note: this is very tricky; we must never 'overshoot' the * correction downwards.

label: code-design

37006. * Forward declarations

37007. conservative

37008. Note: in integer arithmetic, $(x / 4)$ is same as $\text{floor}(x / 4)$ for non-negative * values, but is incorrect for negative ones.

37009. Compute day number of the first day of a given year.

37010. 29: setUTCMinutes

37011. Behavior for nargs < 2 is implementation dependent: currently we'll * set a NaN time value (matching V8 behavior) in this case.

37012. "+11:22:0"

37013. Note: toJSON() is a generic function which works even if 'this' * is not a Date. The sole argument is ignored.

37014. **comment:** Two value cycle, see e.g. test-bi-date-tzoffset-basic-fi.js. * In these cases, favor a higher tzoffset to get a consistent * result which is independent of iteration count. Not sure if * this is a generically correct solution.**label:** code-design

37015. complete the millisecond field

37016. * Getters. ** Implementing getters is quite easy. The internal time value is either * NaN, or represents milliseconds (without fractions) from Jan 1, 1970. * The internal time value can be converted to integer parts, and each * part will be normalized and will fit into a 32-bit signed integer. ** A shared native helper to provide all getters. Magic value contains * a set of flags and also packs the date component index argument. The * helper provides: * * getFullYear() * getUTCFullYear() * getMonth() * getUTCMonth() * getDate() * getUTCDate() * getDay() * getUTCDay() * getHours() * getUTCHours() * getMinutes() * getUTCMinutes() * getSeconds() * getUTCSecs() * getMilliseconds() * getUTCMilliseconds() * getYear() * * Notes: * * - Date.prototype.getDate(): 'date' means day-of-month, and is * zero-based in internal calculations but public API expects it to * be one-based. * * - Date.prototype.getTime() and Date.prototype.valueOf() have identical * behavior. They have separate function objects, but share the same C * function (duk_bi_date_prototype_value_of).

37017. day number for Jan 1 since epoch

37018. -> [... this timeval]

37019. Parser part indices.

37020. no fractions in internal time

37021. no need to mask idx portion

37022. does not fit into 16 bits

37023. This is based on Rhino EquivalentYear() algorithm: *

<https://github.com/mozilla/rhino/blob/f99cc11d616f0cdda2c42bde72b3484df6182947/src/org.mozilla/javascript/NativeDate.java>

37024. 0

37025. Iteration solution

37026. accept string if next char is NUL (otherwise reject)

37027. SCANBUILD: scan-build complains here about assigned value * being garbage or undefined. This is correct but operating * on undefined values has no ill effect and is ignored by the * caller in the case where this happens.

37028. 31: setUTCHours

37029. This native function is also used for Date.prototype.getTime() * as their behavior is identical.

37030. Note1

37031. Given a (year, month, day-within-month) triple, compute day number. * The input triple is un-normalized and may contain non-finite values.

37032. accept string

37033. No platform-specific parsing, this is not an error.

37034. Apply ToNumber() to specified index; if ToInteger(val) in [0,99], add * 1900 and replace value at idx_val.

37035. 4: toLocaleDateString

37036. This is based on V8 EquivalentYear() algorithm (see src/genequivyear.py): * <http://code.google.com/p/v8/source/browse/trunk/src/date.h#146>

37037. Use double parts, they tolerate unnormalized time. ** Note: DUK_DATE_IDX_WEEKDAY is initialized with a bogus value (DUK_PI_TZHOUR) * on purpose. It won't be actually used by duk_bi_date_get_timeval_from_dparts(), * but will make the value initialized just in case, and avoid any * potential for Valgrind issues.

37038. See comments below on MakeTime why these are volatile.

37039. Contract, either: * - Push value on stack and return 1 * - Don't push anything on stack and return 0

37040. E5 Sections 15.9.3.1, B.2.4, B.2.5

37041. tzoffset

37042. The timevalue must be in valid Ecmascript range, but since a local * time offset can be applied, we need to allow a +/- 24h leeway to * the value. In other words, although the UTC time is within the * Ecmascript range, the local part values can be just outside of it.

37043. seconds

37044. 18: getMinutes

37045. Buffer sizes for some UNIX calls. Larger than strictly necessary * to avoid Valgrind errors.

37046. DUK_USE_DATE_TZO_GMTIME

37047. one-based -> zero-based

37048. DUK_USE_DATE_PRS_STRPTIME

37049. **comment:** XXX: return something more useful, so that caller can throw?**label:** code-design

37050. For NaN/inf, the return value doesn't matter.

37051. **comment:** If the platform doesn't support the entire Ecmascript range, we need * to return 0 so that the caller can fall back to the default formatter. ** For now, assume that if time_t is 8 bytes or more, the whole Ecmascript * range is supported. For smaller time_t values (4 bytes in practice), * assumes that the signed 32-bit range is supported. * * XXX: detect this more correctly per platform. The size of time_t is * probably not an accurate guarantee of strftime() supporting or not * supporting a large time range (the full Ecmascript range).**label:** code-design

37052. Compute final offset in seconds, positive if local time ahead of * UTC (returned value is UTC-to-local offset). ** difftime() returns a double, so coercion to int generates quite * a lot of code. Direct subtraction is not portable, however. * XXX: allow direct subtraction on known platforms.

37053. no fractions

37054. The necessary #includes are in place in duk_config.h.

37055. already one-based

37056. DUK_USE_DATE_PRS_GETDATE

37057. DUK_USE_DATE_NOW_GETTIMEOFDAY

37058. be paranoid for 32-bit time values (even avoiding negative ones)

37059. unsigned 31-bit range

37060. Get local time offset (in seconds) for a certain (UTC) instant 'd'.

37061. negative: dst info not available

37062. Not a very good provider: only full seconds are available.

37063. copy to buffer with spare to avoid Valgrind gripes from strftime

37064. valgrind whine without this

37065. This check used to be for $(t < 0)$ but on some platforms * time_t is unsigned and apparently the proper way to detect * an mktime() error return is the cast above. See e.g.: * <http://pubs.opengroup.org/onlinepubs/009695299/functions/mktime.html>

37066. * Unix-like Date providers ** Generally useful Unix / POSIX / ANSI Date providers.

37067. DUK_USE_DATE_NOW_TIME

37068. Get current Ecmascript time (= UNIX/Posix time, but in milliseconds).

37069. UTC

37070. * This is a bit tricky to implement portably. The result depends * on the timestamp (specifically, DST depends on the timestamp). * If e.g. UNIX APIs are used, they'll have portability issues with * very small and very large years. ** Current approach: * * - Stay within portable UNIX limits by using equivalent year mapping. * Avoid year 1970 and 2038 as some conversions start to fail, at * least on some platforms. Avoiding 1970 means that there are * currently DST

discrepancies for 1970. ** - Create a UTC and local time breakdowns from 't'. Then create * a time_t using gmtime() and localtime() and compute the time * difference between the two. ** Equivalent year mapping (E5 Section 15.9.1.8): ** If the host environment provides functionality for determining * daylight saving time, the implementation of ECMAScript is free * to map the year in question to an equivalent year (same * leap-year-ness and same starting week day for the year) for which * the host environment provides daylight saving time information. * The only restriction is that all equivalent years should produce * the same result. ** This approach is quite reasonable but not entirely correct, e.g. * the specification also states (E5 Section 15.9.1.8): ** The implementation of ECMAScript should not try to determine * whether the exact time was subject to daylight saving time, but * just whether daylight saving time would have been in effect if * the _current daylight saving time algorithm_ had been used at the * time. This avoids complications such as taking into account the * years that the locale observed daylight saving time year round. ** Since we rely on the platform APIs for conversions between local * time and UTC, we can't guarantee the above. Rather, if the platform * has historical DST rules they will be applied. This seems to be the * general preferred direction in Ecmascript standardization (or at least * implementations) anyway, and even the equivalent year mapping should * be disabled if the platform is known to handle DST properly for the * full Ecmascript range. ** The following has useful discussion and links: ** https://bugzilla.mozilla.org/show_bug.cgi?id=351066

37071. local

37072. DUK_USE_DATE_FMT_STRFTIME

37073. tm_isdst is both an input and an output to mktime(), use 0 to * avoid DST handling in mktime(): * - <https://github.com/svaarala/duktape/issues/406> * - <http://stackoverflow.com/questions/8558919/mktime-and-tm-isdst>

37074. flags

37075. For this to work, DATEMSK must be set, so this is not very * convenient for an embeddable interpreter.

37076. If not within Ecmascript range, some integer time calculations * won't work correctly (and some asserts will fail), so bail out * if so. This fixes test-bug-date-insane-setyear.js. There is * a +/- 24h leeway in this range check to avoid a test262 corner * case documented in test-bug-date-timeval-edges.js.

37077. DUK_USE_DATE_NOW_WINDOWS

37078. input 'd' in Windows UTC, 100ns units

37079. Difference is in 100ns units, convert to milliseconds w/o fractions

37080. millisec -> 100ns units since jan 1, 1970

37081. XXX: handling of timestamps outside Windows supported range. * How does Windows deal with dates before 1600? Does windows * support all Ecmascript years (like -200000 and +200000)? * Should equivalent year mapping be used here too? If so, use * a shared helper (currently integrated into timeval-to-parts).

37082. The necessary #includes are in place in duk_config.h.

37083. * Windows Date providers ** Platform specific links: * - [http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473(v=vs.85).aspx)

37084. Use the approach described in "Remarks" of FileTimeToLocalFileTime: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx)

37085. Suggested step-by-step method from documentation of RtTimeToSecondsSince1970: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928(v=vs.85).aspx)

37086. Shared Windows helpers.

37087. Positive if local time ahead of UTC.

37088. **comment:** not sure whether or not needed; Thursday

label: requirement

37089. seconds

37090. defined(DUK_USE_DATE_NOW_WINDOWS) || defined(DUK_USE_DATE_TZO_WINDOWS)

37091. DUK_USE_DATE_TZO_WINDOWS

37092. address

37093. -> [val arr]

37094. XXX: Not sure what the best return value would be in the API. * Return a boolean for now. Note that rc == 0 is success (true).

37095. type tag (public)

37096. return the argument object

37097. result array

37098. idx_value

37099. idx_space

37100. **comment:** XXX: primitive to make array from valstack slice

label: code-design

37101. XXX: version specific array format instead?

37102. **comment:** * Duktape built-ins ** Size optimization note: it might seem that vararg multipurpose functions * like fin(), enc(), and dec() are not very size optimal, but using a single * user-visible Ecmascript function saves a lot of run-time footprint; each * Function instance takes >100 bytes. Using a shared native helper and a * 'magic' value won't save much if there are multiple Function instances * anyway.

label: code-design

37103. refcount

37104. **comment:** When alloc_size == 0 the second allocation may not * actually exist.

label: code-design

37105. because arg count is 1

37106. heapdr size and additional allocation size, followed by * type specific stuff (with varying value count)

37107. **comment:** Providing access to e.g. act->lex_env would be dangerous: these * internal structures must never be accessible to the application. * Duktape relies on them having consistent data, and this consistency * is only asserted for, not checked for.

label: code-design

37108. Raw helper to extract internal information / statistics about a value. * The return values are version specific and must not expose anything * that would lead to security issues (e.g. exposing compiled function * 'data' buffer might be an issue). Currently only counts and sizes and * such are given so there should not be a security impact.

37109. Note: e_next indicates the number of gc-reachable entries * in the entry part, and also indicates the index where the * next new property would be inserted. It does *not* indicate * the number of non-NULL keys present in the object. That * value could be counted separately but requires a pass through * the key list.

37110. set values into ret array

37111. **comment:** Set: currently a finalizer is disabled by setting it to * undefined; this does not remove the property at the moment. * The value could be type checked to be either a function * or something else; if something else, the property could * be deleted.

label: code-design

37112. [level obj func pc line]

37113. idx_replacer

37114. Relevant PC is just before current one because PC is * post-incremented. This should match what error augment * code does.

37115. * Compact an object

37116. -1 = top callstack entry, callstack[callstack_top - 1] * -callstack_top = bottom callstack entry, callstack[0]

37117. flags

37118. **comment:** Vararg function: must be careful to check/require arguments. * The JSON helpers accept invalid indices and treat them like * non-existent optional parameters.

label: code-design

37119. internal type tag

37120. Get.

37121. [... this]

37122. fixed arg count: value

37123. **comment:** * Traceback handling ** The unified helper decodes the traceback and produces various requested * outputs. It should be optimized for size, and may leave garbage on stack, * only the topmost return value matters. For instance, traceback separator * and decoded strings are pushed even when looking for filename only. ** NOTE: although _Tracedata is an internal property, user code can currently * write to the array (or replace it with something other than an array). * The code below must tolerate arbitrary _Tracedata. It can throw errors * etc, but cannot cause a segfault or memory unsafe behavior.

label: code-design

37124. noblame_fileline

37125. **comment:** If message is undefined, the own property 'message' is not set at * all to save property space. An empty message is inherited anyway.
label: code-design

37126. When looking for .fileName/.lineNumber, blame first * function which has a .fileName.

37127. [... this tracedata sep this str1 ... strN]

37128. may be NULL

37129. constants arbitrary, chosen for small loads

37130. * Ecmascript/native function call or lightfunc call

37131. not enumerable

37132. The 'this' after 'sep' will get ToString() coerced by * duk_join() automatically. We don't want to do that * coercion when providing .fileName or .lineNumber (GH-254).

37133. **comment:** XXX: Are steps 6 and 7 in E5 Section 15.11.4.4 duplicated by * accident or are they actually needed? The first ToString() * could conceivably return 'undefined'.
label: code-design

37134. **comment:** XXX: skip null filename?
label: code-design

37135. NULL for lightfunc

37136. **comment:** XXX: remove this native function and map 'stack' accessor * to the toString() implementation directly.
label: code-design

37137. unknown, ignore

37138. [... v1 v2 name filename]

37139. traceback depth ensures fits into 16 bits

37140. [... obj key value]

37141. * Traceback handling when tracebacks disabled. ** The fileName / lineNumber stubs are now necessary because built-in * data will include the accessor properties in Error.prototype. If those * are removed for builds without tracebacks, these can also be removed. * 'stack' should still be present and produce a ToString() equivalent: * this is useful for user code which prints a stacktrace and expects to * see something useful. A normal stacktrace also begins with a ToString() * of the error so this makes sense.

37142. fixed arg count

37143. [... this name message]

37144. name is empty -> return message

37145. **comment:** XXX: Output type could be encoded into native function 'magic' value to * save space. For setters the stridx could be encoded into 'magic'.
label: code-design

37146. Augment the error if called as a normal function. __FILE__ and __LINE__ * are not desirable in this case.

37147. [... v1(filename) v2(line+flags)]

37148. [... v1 v2 name filename str] -> [... str v2 name filename]

37149. Behavior for constructor and non-constructor call is * the same except for augmenting the created error. When * called as a constructor, the caller (duk_new()) will handle * augmentation; when called as normal function, we need to do * it here.

37150. -> [... str]

37151. * __FILE__ / __LINE__ entry, here 'pc' is line number directly. * Sometimes __FILE__ / __LINE__ is reported as the source for * the error (fileName, lineNumber), sometimes not.

37152. Possibly truncated; there is no explicit truncation * marker so this is the best we can do.

37153. [... v1 v2 str] -> [... str v2]

37154. DUK_USE_TRACEBACKS

37155. traceback depth fits into 16 bits

37156. [message error message]

37157. same for both error and each subclass like TypeError

37158. Attempt to write 'stack', 'fileName', 'lineNumber' works as if * user code called Object.defineProperty() to create an overriding * own property. This allows user code to overwrite .fileName etc * intuitively as e.g. "err.fileName = 'dummy'" as one might expect. * See <https://github.com/svaarala/duktape/issues/387>.

37159. [... v1(func) v2(pc+flags)]

37160. ... name ': ' message

37161. **comment:** XXX: optimize with more direct internal access
label: code-design

37162. **comment:** XXX: Could be improved by coercing to a readable duk_tval (especially string escaping)
label: code-design

37163. [... this name]

37164. message is empty -> return name

37165. [... this tracedata sep this]

37166. * Error built-ins

37167. **comment:** XXX: Change 'anon' handling here too, to use empty string for anonymous functions?
label: code-design

37168. count, not including sep

37169. stack type fits into 16 bits

37170. When looking for .fileName/.lineNumber, blame compilation * or C call site unless flagged not to do so.

37171. Current tracedata contains 2 entries per callstack entry.

37172. vararg function, careful arg handling (e.g. thisArg may not be present)

37173. create bound function object

37174. not a vararg function

37175. these non-standard properties are copied for convenience

37176. step 15.a

37177. only outer_lex_env matters, as functions always get a new * variable declaration environment.

37178. [body formals source template closure]

37179. add_auto_proto

37180. [body formals source]

37181. **comment:** Function name: missing/undefined is mapped to empty string, * otherwise coerce to string.
label: code-design

37182. XXX: the implementation now assumes "chained" bound functions, * whereas "collapsed" bound functions (where there is ever only * one bound function which directly points to a non-bound, final * function) would require a "collapsing" implementation which * merges argument lists etc here.

37183. [body formals], formals is comma separated list that needs to be parsed

37184. [arg1 ... argN-1 body] -> [body arg1 ... argN-1]

37185. attrs in E5 Section 15.3.5.1

37186. **comment:** * E5 Section 15.3.4.2 places few requirements on the output of * this function: ** - The result is an implementation dependent representation * of the function; in particular ** - The result must follow the syntax of a FunctionDeclaration. * In particular, the function must have a name (even in the * case of an anonymous function or a function with an empty * name). ** - Note in particular that the output does NOT need to compile * into anything useful.
label: code-design

37187. bound function 'length' property is interesting

37188. caller and arguments must use the same thrower, [[ThrowTypeError]]

37189. [func thisArg arg1 ... argN]

37190. thisArg

37191. vararg function, thisArg needs special handling

37192. **comment:** XXX: 'copy properties' API call?
label: code-design

37193. **comment:** XXX: currently no handling for non-allowed identifier characters, * e.g. a '{' in the function name.
label: code-design

37194. Step 1 is not necessary because duk_call_method() will take * care of it.

37195. * Function built-ins

37196. **comment:** XXX: cover this with the generic >1 case?
label: code-design

37197. **comment:** XXX: make this an internal helper
label: code-design

37198. = 1 + arg count

37199. [thisArg arg1 ... argN func] (thisArg+args == nargs total)

37200. [thisArg arg1 ... argN]

37201. lightfunc

37202. XXX: this placeholder is not always correct, but use for now. * It will fail in corner cases; see test-dev-func-cons-args.js.

37203. [thisArg arg1 ... argN func boundFunc argArray]

37204. **comment:** The 'strict' flag is copied to get the special [[Get]] of E5.1 * Section 15.3.5.4 to apply when a 'caller' value is a strict bound * function. Not sure if this is correct, because the specification * is a bit ambiguous on this point but it would make sense.
label: code-design

37205. [thisArg arg1 ... argN func boundFunc]

37206. Lightfuncs are always strict.

37207. [body formals source template]

37208. ignore arguments, return undefined (E5 Section 15.3.4)

37209. **comment:** XXX: ignored now
label: code-design

37210. func

37211. For lightfuncs, simply read the virtual property.

37212. **comment:** XXX: faster internal way to get this
label: code-design

37213. 'func' in the algorithm

37214. [func thisArg argArray]

37215. ToUint32() coercion required

37216. **comment:** XXX: copy from caller?
label: code-design

37217. normal and constructor calls have identical semantics

37218. Indicate function type in the function body using a dummy * directive.

37219. strictness is not inherited, intentional

37220. Duktape object

37221. Finish the wrapped module source. Force module.filename as the * function .fileName so it gets set for functions defined within a * module. This also ensures loggers created within the module get * the module ID (or overridden filename) as their default logger name. * (Note capitalization: .filename matches Node.js while .fileName is * used elsewhere in Duktape.)

37222. ignore non-strings

37223. Duktape.modSearch(resolved_id, fresh_require, exports, module).

37224. Module id requested

37225. DUK_USE_COMMONJS_MODULES

37226. Module loading failed. Node.js will forget the module * registration so that another require() will try to load * the module again. Mimic that behavior.

37227. Module object containing module.exports, etc

37228. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func]

37229. 2 when called by debugger

37230. return module.exports

37231. eat dup slashes

37232. at most sizeof(buf) - 1

37233. requested identifier

37234. E5.1 Section 15.1.3.2: empty

37235. **comment:** NOTE: we try to minimize code size by avoiding unnecessary pops, * so the stack looks a bit cluttered in this function. DUK_ASSERT_TOP() * assertions are used to ensure stack configuration is correct at each * step.
label: code-design

37236. spaces (nargs - 1) + newline

37237. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded Duktape.modLoaded[id]]

37238. [... Duktape.modSearch resolved_id last_comp fresh_require exports module]

37239. * Parsing of ints and floats

37240. Module has now evaluated to a wrapped module function. Force its * .name to match module.name (defaults to last component of resolved * ID) so that it is shown in stack traces too. Note that we must not * introduce an actual name binding into the function scope (which is * usually the case with a named function) because it would affect the * scope seen by the module and shadow accesses to globals of the same name. * This is now done by compiling the function as anonymous and then forcing * its .name without setting a "has name binding" flag.

37241. Characters outside BMP cannot be escape()'d. We could * encode them as surrogate pairs (for codepoints inside * valid UTF-8 range, but not extended UTF-8). Because * escape() and unescape() are legacy functions, we don't.

37242. Duktape.modLoaded[] module cache

37243. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module]

37244. 1110 xxxx; 3 bytes

37245. [source template result]

37246. if direct eval, calling activation must exist

37247. If argument count is 1 and first argument is a buffer, write the buffer * as raw data into the file without a newline; this allows exact control * over stdout/stderr without an additional entrypoint (useful for now). * * Otherwise current print/alert semantics are to ToString() coerce * arguments, join them with a single space, and append a newline.

37248. last component

37249. Term was '.', backtrack resolved name by one component. * q[-1] = previous slash (or beyond start of buffer) * q[-2] = last char of previous component (or beyond start of buffer)

37250. Duktape.modLoaded

37251. p overshoots

37252. **comment:** XXX: copy from caller?
label: code-design

37253. output is never longer than input during resolution

37254. DUK_USE_PREFER_SIZE

37255. fall through

37256. 0x10-0x1f

37257. Output #2: last component name

37258. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined]

37259. print provider
37260. Eval code doesn't need an automatic .prototype object.
37261. **comment:** * A few notes on the algorithm: ** - Terms are not allowed to begin with a period unless the term * is either '.' or '..'. This simplifies implementation (and * is within CommonJS modules specification). ** - There are few output bound checks here. This is on purpose: * the resolution input is length checked and the output is never * longer than the input. The resolved output is written directly * over the input because it's never longer than the input at any * point in the algorithm. ** - Non-ASCII characters are processed as individual bytes and * need no special treatment. However, U+0000 terminates the * algorithm; this is not an issue because U+0000 is not a * desirable term character anyway.
label: code-design
37262. Term was '!' and is eaten entirely (including dup slashes).
37263. E5.1 Section B.2.2, step 7.
37264. Backtrack to previous slash or start of buffer.
37265. * Number checkers
37266. Decode UTF-8 codepoint from a sequence of hex escapes. The * first byte of the sequence has been decoded to 't'. ** Note that UTF-8 validation must be strict according to the * specification: E5.1 Section 15.1.3, decode algorithm step * 4.d.vii.8. URIError from non-shortest encodings is also * specifically noted in the spec.
37267. Resolved, normalized absolute module ID
37268. Here 'p' always points to the start of a term. ** We can also unconditionally reset q_last here: if this is * the last (non-empty) term q_last will have the right value * on loop exit.
37269. add_auto_proto
37270. 0x20-0x2f
37271. module.filename for .fileName, default to resolved ID if * not present.
37272. E5.1 Section 15.1.3.1: uriReserved + '#'
37273. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func exports fresh_require exports module]
37274. New require() function for module, updated resolution base
37275. this may have side effects, so re-lookup act
37276. a fresh require() with require.id = resolved target module id
37277. Delete entry in Duktape.modLoaded[] and rethrow.
37278. 110x xxxx; 2 bytes
37279. Module table: * - module.exports: initial exports table (may be replaced by user) * - module.id is non-writable and non-configurable, as the CommonJS * spec suggests this if possible * - module.filename: not set, defaults to resolved ID if not explicitly * set by modSearch() (note capitalization, not .fileName, matches Node.js) * - module.name: not set, defaults to last component of resolved ID if * not explicitly set by modSearch()
37280. 0x30-0x3f
37281. initial size guess
37282. decode '%xx' to '%xx' if decoded char in reserved set
37283. The base ID of the current require() function, resolution base
37284. [requested_id require require.id resolved_id last_comp]
37285. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module result(ignored)]
37286. Current require() function
37287. require.id of current module
37288. mod_id may be NULL
37289. **comment:** Higher footprint, less churn.
label: code-design
37290. DUK_USE_SECTION_B
37291. compact the exports table
37292. Needs lookahead
37293. 'undefined', artifact of lookup
37294. 1111 0xxx; 4 bytes
37295. **comment:** NOTE: level is used only by the debugger and should never be present * for an EcmaScript eval().
label: code-design
37296. not reachable
37297. 0x50-0x5f
37298. If user callback did not return source code, module loading * is finished (user callback initialized exports table directly).
37299. Last component name in resolved path
37300. -> [... source]
37301. module table remains registered to modLoaded, minimize its size
37302. Although we can allow non-BMP characters (they'll decode * back into surrogate pairs), we don't allow extended UTF-8 * characters; they would encode to URIs which won't decode * back because of strict UTF-8 checks in URI decoding. * (However, we could just as well allow them here.)
37303. this is just beneath bottom
37304. p points to digit part ('%xy', p points to 'x')
37305. * Set up the resolution input which is the requested ID directly * (if absolute or no current module path) or with current module * ID prepended (if relative and current module path exists). ** Suppose current module is 'foo/bar' and relative path is './quux'. * The 'bar' component must be replaced so the initial input here is * 'foo/bar/.././quux'.
37306. by default, use caller's environment
37307. eat (first) input slash
37308. module.id = resolved_id
37309. indirect eval
37310. %6xx%xx...%xx', p points to char after first '%'
37311. [source template closure this]
37312. module
37313. E5.1 Section 15.1.3.4: uriUnescaped
37314. invalidated
37315. E5.1 Section 15.1.3.3: uriReserved + uriUnescaped + '#'
37316. this function
37317. exports (this binding)
37318. * Resolve module identifier into canonical absolute form.
37319. continuation byte
37320. * CommonJS require() and modules support
37321. Copy term name until end or '/'.
37322. Non-BMP characters within valid UTF-8 range: encode as is. * They'll decode back into surrogate pairs if the escaped * output is decoded.
37323. delete Duktape.modLoaded[resolved_id]
37324. Newline allows module last line to contain a // comment.
37325. * Module not loaded (and loading not started previously). ** Create a new require() function with 'id' set to resolved ID * of module being loaded. Also create 'exports' and 'module' * tables but don't register exports to the loaded table yet. * We don't want to do that unless the user module search callbacks * succeeds in finding the module.
37326. bytes left
37327. [source template closure]
37328. * Encoding/decoding helpers

37329. Have a calling activation, check for direct eval (otherwise * assume indirect eval).

37330. **comment:** Compact but lots of churn.
label: code-design

37331. **comment:** The E5.1 algorithm checks whether or not a decoded codepoint * is below 0x80 and perhaps may be in the "reserved" set. * This seems pointless because the single byte UTF-8 case is * handled separately, and non-shortest encodings are rejected. * So, 'cp' cannot be below 0x80 here, and thus cannot be in * the reserved set.
label: code-design

37332. Potentially truncated, NUL not guaranteed in any case. * The (int_rc < 0) case should not occur in practice.

37333. 0x40-0x4f

37334. **comment:** Fresh require: require.id is left configurable (but not writable) * so that is not easy to accidentally tweak it, but it can still be * done with Object.defineProperty(). * * XXX: require.id could also be just made non-configurable, as there * is no practical reason to touch it.
label: code-design

37335. * callstack_top - 1 --> this function * callstack_top - 2 --> caller (may not exist) * * If called directly from C, callstack_top might be 1. If calling * activation doesn't exist, call must be indirect.

37336. This was the last term, and q_last was * updated to match this term at loop top.

37337. User callback did not return source code, so module loading * is finished: just update modLoaded with final module.exports * and we're done.

37338. e.g. require('foo'), empty terms not allowed

37339. at least this function exists

37340. 0x70-0x7f

37341. **comment:** XXX: check flags
label: requirement

37342. write on next loop

37343. Default exports table

37344. Duktape.modLoaded[resolved_id] = module

37345. * Call user provided module search function and build the wrapped * module source code (if necessary). The module search function * can be used to implement pure Ecmascript, pure C, and mixed * Ecmascript/C modules. * * The module search function can operate on the exports table directly * (e.g. DLL code can register values to it). It can also return a * string which is interpreted as module source code (if a non-string * is returned the module is assumed to be a pure C one). If a module * cannot be found, an error must be thrown by the user callback. * * Because Duktape.modLoaded[] already contains the module being * loaded, circular references for C modules should also work * (although expected to be quite rare).

37346. stash to bottom of value stack to keep new_env reachable for duration of eval

37347. XXX: Could add fast path (for each transform callback) with direct byte * lookups (no shifting) and no explicit check for x < 0x80 before table * lookup.

37348. extended utf-8 not allowed for URIs

37349. This is guaranteed by debugger code.

37350. Maximum write size: XUTF8 path writes max DUK_UNICODE_MAX_XUTF8_LENGTH, * percent escape path writes max two times CESU-8 encoded BMP length.

37351. E5 Section 10.4.2

37352. Only direct eval inherits strictness from calling code * (E5.1 Section 10.1.1).

37353. Macros for creating and checking bitmasks for character encoding. * Bit number is a bit counterintuitive, but minimizes code size.

37354. exports

37355. return arg as-is

37356. * Global object built-ins

37357. **comment:** * Eval * * Eval needs to handle both a "direct eval" and an "indirect eval". * Direct eval handling needs access to the caller's activation so that its * lexical environment can be accessed. A direct eval is only possible from * Ecmascript code; an indirect eval call is possible also from C code. * When an indirect eval call is made from C code, there may not be a * calling activation at all which needs careful handling.
label: code-design

37358. module.name for .name, default to last component if * not present.

37359. rethrow original error

37360. module.exports = exports

37361. 0x60-0x6f

37362. UTF-8 encoded bytes escaped as %xx%xx%xx... -> 3 * nbytes. * Codepoint range is restricted so this is a slightly too large * but doesn't matter.

37363. Output #1: resolved absolute name

37364. [source]

37365. **comment:** Stack indices for better readability
label: code-design

37366. **comment:** XXX: refactor and share with other code
label: code-design

37367. Specification stripPrefix maps to DUK_S2N_FLAG_ALLOW_AUTO_HEX_INT. * * Don't autodetect octals (from leading zeroes), require user code to * provide an explicit radix 8 for parsing octal. See write-up from Mozilla: * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt#ECMAScript_5_Removes_Octal_Interpretation

37368. * Cached module check. * * If module has been loaded or its loading has already begun without * finishing, return the same cached value ('exports'). The value is * registered when module load starts so that circular references can * be supported to some extent.

37369. Register the module table early to modLoaded[] so that we can * support circular references even in modSearch(). If an error * is thrown, we'll delete the reference.

37370. Duktape.modSearch

37371. module (argument)

37372. fresh require (argument)

37373. resolved id: require(id) must return this same module

37374. caller

37375. avoid dereference after potential callstack realloc

37376. relookup exports from module.exports in case it was changed by modSearch

37377. must be signed

37378. * Resolution loop. At the top of the loop we're expecting a valid * term: ':', '.', or a non-empty identifier not starting with a period.

37379. * URI handling

37380. * Call the wrapped module function. * * Use a protected call so that we can update Duktape.modLoaded[resolved_id] * even if the module throws an error.

37381. utf-8 validation ensures these

37382. 0x00-0x0f

37383. nargs

37384. compiler's responsibility

37385. backtrack to last output slash (dups already eliminated)

37386. [source template]

37387. 'buf' contains the string to write, 'sz_buf' contains the length * (which may be zero).

37388. **comment:** Certain boxed types are required to go through * automatic unboxing. Rely on internal value being * sane (to avoid infinite recursion).
label: code-design

37389. DUK_USE_JX

37390. There is no need to check the first character specially here * (i.e. reject digits): the caller only accepts valid initial * characters and won't call us if the first character is a digit. * This also ensures that the plain string won't be empty.

37391. Note: intentionally signed.

37392. XXX: push_uint_string / push_u32_string

37393. [... buf loop]

37394. idx_value
37395. backtrack (safe)
37396. * Top level wrappers
37397. Loop check.
37398. len: 9
37399. required to keep recursion depth correct
37400. [... buf loop (proplist) (gap) holder ""]
37401. Final result is at stack top.
37402. ToInteger() coercion; NaN -> 0, infinities are clamped to 0 and 10
37403. * Local defines and forward declarations.
37404. already zero
37405. [... val root ""] -> [... val val']
37406. **comment:** * JSON built-ins. * * See doc/json.rst. * * Codepoints are handled as duk_uint_fast32_t to ensure that the full * unsigned 32-bit range is supported. This matters to e.g. JX. * * Input parsing doesn't do an explicit end-of-input check at all. This is * safe: input string data is always NUL-terminated (0x00) and valid JSON * inputs never contain plain NUL characters, so that as long as syntax checks * are correct, we'll never read past the NUL. This approach reduces code size * and improves parsing performance, but it's critical that syntax checks are * indeed correct!
label: code-design
37407. flags
37408. Guaranteed by recursion_limit setup so we don't have to * check twice.
37409. The Quote(value) operation: quote a string. * * Stack policy: [] -> [].
37410. [... obj]
37411. The coercion potentially invokes user .valueOf() and .toString() * but can't result in a function value because [[DefaultValue]] would * reject such a result: test-dev-json-stringify-coercion-1.js.
37412. if gap is empty, behave as if not given at all
37413. When JX/JC not in use, the type mask above will avoid this case if needed.
37414. Select a safe loop count where no output checks are * needed assuming we won't encounter escapes. Input * bound checks are not necessary as a NUL (guaranteed) * will cause a SyntaxError before we read out of bounds.
37415. Shared handler to minimize parser size. Cause will be * hidden, unfortunately, but we'll have an offset which * is often quite enough.
37416. DUK_USE_JSON_EATWHITE_FASTPATH
37417. "" was eaten by caller
37418. DUK_USE_HEX_FASTPATH
37419. parse value
37420. -> [... key val replacer holder key]
37421. as is
37422. **comment:** Buffer values are encoded in (lowercase) hex to make the * binary data readable. Base64 or similar would be more * compact but less readable, and the point of JX/JC * variants is to be as useful to a programmer as possible.
label: code-design
37423. unconditional block
37424. any other printable -> as is
37425. emitted
37426. -> [... holder name val val ToString(i)]
37427. replacer is a mutation risk
37428. skip ''
37429. toJSON() can also be a lightfunc
37430. parse key and value
37431. default attrs ok
37432. Trailing whitespace has been eaten by duk__dec_value(), so if * we're not at end of input here, it's a SyntaxError.
37433. * Parsing implementation. * * JSON lexer is now separate from duk_lexer.c because there are numerous * small differences making it difficult to share the lexer. * * The parser here works with raw bytes directly; this works because all * JSON delimiters are ASCII characters. Invalid xUTF-8 encoded values * inside strings will be passed on without normalization; this is not a * compliance concern because compliant inputs will always be valid * CESU-8 encodings.
37434. 0x7F is special
37435. -> [... key val val']
37436. skip ')'
37437. [... val]
37438. Fast path, decode as is.
37439. A non-Array object should not have an array part in practice. * But since it is supported internally (and perhaps used at some * point), check and abandon if that's the case.
37440. syntax error
37441. -> [... proplist enum_obj key]
37442. avoid attempts to add/remove object keys
37443. XXX: a "first" flag would suffice
37444. avail_bytes += need_bytes
37445. Parse a number, other than NaN or +/- Infinity
37446. 0x00: finish (non-white) * 0x01: continue
37447. Plus sign must be accepted for positive exponents * (e.g. '1.5e+2'). This clause catches NULs.
37448. nothing now
37449. caller guarantees
37450. DUK_USE_JX && DUK_USE_JC
37451. [... buf]
37452. -> [... key val replacer]
37453. serialize the wrapper with empty string key
37454. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor). * * Using duk_put_prop() works incorrectly with '__proto__' * if the own property with that name has been deleted. This * does not happen normally, but a clever reviver can trigger * that, see complex reviver case in: test-bug-json-parse-__proto__.js.
label: code-design
37455. Note that we currently parse -bytes-, not codepoints. * All non-ASCII extended UTF-8 will encode to bytes >= 0x80, * so they'll simply pass through (valid UTF-8 or not).
37456. Encode a double (checked by caller) from stack top. Stack top may be * replaced by serialized string but is not popped (caller does that).
37457. Function values are handled completely above (including * coercion results):
37458. '\Udeadbeef'
37459. enum_index
37460. eat closing bracket
37461. Encode string in small chunks, estimating the maximum expansion so that * there's no need to ensure space while processing the chunk.
37462. Loop check using a hybrid approach: a fixed-size visited[] array * with overflow in a loop check object.
37463. Must decrease recursion depth before returning.
37464. * Create wrapper object and serialize
37465. All other object types.
37466. * Helper tables

37467. Flag handling currently assumes that flags are consistent. This is OK * because the call sites are now strictly controlled.

37468. **comment:** XXX: duplicates should be eliminated here

label: code-design

37469. **comment:** In compatible mode and standard JSON mode, output * something useful for non-BMP characters. This won't * roundtrip but will still be more or less readable and * more useful than an error.

label: code-design

37470. error message doesn't matter, ignored anyway

37471. must not truncate

37472. negative top-relative indices not allowed now

37473. Encode a fastint from duk_tval ptr, no value stack effects.

37474. avoid attempt to compact any objects

37475. Ensure space for 1:1 output plus one escape.

37476. **comment:** XXX: Could just lookup . toJSON() and continue in fast path, * as it would almost never be defined.

label: code-design

37477. DUK_USE_JSON_DECSTRING_FASTPATH

37478. [... number] -> [... string]

37479. **comment:** XXX: share helper from lexer; duk_lexer.c / hexval().

label: code-design

37480. **comment:** We've ensured space for one escaped input; then * bail out and recheck (this makes escape handling * quite slow but it's uncommon).

label: code-design

37481. **comment:** Use recursion_limit to ensure we don't overwrite js_ctx->visiting[] * array so we don't need two counter checks in the fast path. The * slow path has a much larger recursion limit which we'll use if * necessary.

label: code-design

37482. Recursive value reviver, implements the Walk() algorithm. No C recursion * check is done here because the initial parsing step will already ensure * there is a reasonable limit on C recursion depth and hence object depth.

37483. Note: space must cater for both JX and JC.

37484. **comment:** Indent helper. Calling code relies on js_ctx->recursion_depth also being * directly related to indent depth.

label: code-design

37485. temp copy, write back for next loop

37486. [... buf loop (proplist)]

37487. unsigned

37488. but don't allow leading plus

37489. **comment:** XXX: Here a "slice copy" would be useful.

label: code-design

37490. 0x00: finish (not part of number) * 0x01: continue

37491. -> [... val root ""]

37492. [... holder name val enum obj_key val obj_key]

37493. "-Infinity", '-' has been eaten

37494. JSON parsing code is allowed to read [p_start,p_end]: p_end is * valid and points to the string NUL terminator (which is always * guaranteed for duk_hstrings).

37495. Could also rely on native sprintf(), but it will handle * values like NaN, Infinity, -0, exponent notation etc in * a JSON-incompatible way.

37496. original target at entry_top - 1

37497. catches EOF and invalid digits

37498. safe, because matched (NUL causes a break)

37499. not emitted

37500. [... arr val]

37501. maxval

37502. To handle deeper indents efficiently, make use of copies we've * already emitted. In effect we can emit a sequence of 1, 2, 4, * 8, etc copies, and then finish the last run. Byte counters * avoid multiply with gap_len on every loop.

37503. Must prevent finalizers which may have arbitrary side effects.

37504. expensive flag

37505. Accept ASCII strings which conform to identifier requirements * as being emitted without key quotes. Since we only accept ASCII * there's no need for actual decoding: 'p' is intentionally signed * so that bytes >= 0x80 extend to negative values and are rejected * as invalid identifier codepoints.

37506. The stack has a variable shape here, so force it to the * desired one explicitly.

37507. The Str(key, holder) operation. ** Stack policy: [... key] -> [...]

37508. -> [... holder name val new_elem]

37509. NUL term or -1 (EOF), NUL check would suffice

37510. proplist is very rare

37511. DUK_USE_JC

37512. -> [... key val replacer holder key val]

37513. Decode a plain string consisting entirely of identifier characters. * Used to parse plain keys (e.g. "foo: 123").

37514. **comment:** 0x00 ... 0x7f: as is * 0x80: escape generically * 0x81: slow path * 0xa0 ... 0xff: backslash + one char

label: code-design

37515. -> [...]

37516. -> [... key val toJSON val key]

37517. Caller ensures space for at least DUK_JSON_MAX_ESC_LEN.

37518. Zero length string is not accepted without quotes

37519. Getter might have arbitrary side effects, * so bail out.

37520. nret

37521. Negative zero needs special handling in JX/JC because * it would otherwise serialize to '0', not '-0'.

37522. -> [... voidp voidp]

37523. [... holder name val enum obj_key new_elem]

37524. Disabled until fixed, see above.

37525. Error message doesn't matter: the error is ignored anyway.

37526. -> [... key val]

37527. Value would normally be omitted, replace with 'null'.

37528. Value would yield 'undefined', so skip key altogether. * Side effects have already happened.

37529. XXX: array internal?

37530. For JX, expressing the whole unsigned 32-bit range matters.

37531. Caller must ensure 'tv' is indeed a fastint!

37532. safe

37533. will result in undefined

37534. **comment:** Execute the fast path in a protected call. If any error is thrown, * fall back to the slow path. This includes e.g. recursion limit * because the fast path has a smaller recursion limit (and simpler, * limited loop detection).

label: code-design

37535. typed for duk_unicode_decode_xutf8()

37536. **comment:** XXX: Would be nice to share the fast path loop from duk_hex_decode() * and avoid creating a temporary buffer. However, there are some * differences which prevent trivial sharing: * * - Pipe char detection * - EOF detection * - Unknown length of input and output * * The best approach here would be a bufwriter and a reasonably sized * safe inner loop (e.g. 64 output bytes at a time).

label: code-design

37537. Once recursion depth is increased, exit path must decrease * it (though it's OK to abort the fast path).
37538. If object has a .toJSON() property, we can't be certain * that it wouldn't mutate any value arbitrarily, so bail * out of the fast path. ** If an object is a Proxy we also can't avoid side effects * so abandon.
37539. **comment:** Unlike in duk_hex_encode() 'dst' is not necessarily aligned by 2. * For platforms where unaligned accesses are not allowed, shift 'dst' * ahead by 1 byte to get alignment and then DUK_MEMMOVE() the result * in place. The faster encoding loop makes up the difference. * There's always space for one extra byte because a terminator always * follows the hex data and that's been accounted for by the caller.
- label:** code-design
37540. catches EOF (NUL)
37541. The code below is incorrect if .toString() or .valueOf() have * have been overridden. The correct approach would be to look up * the method(s) and if they resolve to the built-in function we * can safely bypass it and look up the internal value directly. * Unimplemented for now, abort fast path for boxed values.
37542. accept comma, expect new value
37543. Array part may be larger than 'length'; if so, iterate * only up to array 'length'.
37544. -> [... reviver holder name val]
37545. [... holder name val]
37546. We rely on a few object flag / class number relationships here, * assert for them.
37547. Standard JSON omits functions
37548. unreachable
37549. nargs
37550. The JA(value) operation: encode array. ** Stack policy: [array] -> [array].
37551. x == 0x00 (EOF) causes syntax_error
37552. **comment:** XXX: move masks to js_ctx? they don't change during one * fast path invocation.
- label:** code-design
37553. Note: duk_dec_req_stridx() backtracks one char
37554. -> [... key val toJSON val]
37555. Here the specification requires correct array index enumeration * which is a bit tricky for sparse arrays (it is handled by the * enum setup code). We now enumerate ancestors too, although the * specification is not very clear on whether that is required.
37556. First pass parse is very lenient (e.g. allows '1.2.3') and extracts a * string for strict number parsing.
37557. -> [... key val]
37558. 0x00: slow path * other: as is
37559. value was undefined/unsupported
37560. **comment:** spaces[] must be static to allow initializer with old compilers like BCC
- label:** code-design
37561. Ordinary gap, undefined encodes to 'null' in * standard JSON (and no JX/JC support here now).
37562. Conceptually we'd extract the plain underlying buffer * or its slice and then do a type mask check below to * see if we should reject it. Do the mask check here * instead to avoid making a copy of the buffer slice.
37563. * JSON.stringify() fast path ** Otherwise supports full JSON, JX, and JC features, but bails out on any * possible side effect which might change the value being serialized. The * fast path can take advantage of the fact that the value being serialized * is unchanged so that we can walk directly through property tables etc.
37564. **comment:** XXX: helper
- label:** code-design
37565. coerce in-place
37566. Number serialization has a significant impact relative to * other fast path code, so careful fast path for fastints.
37567. [... num]
37568. * Entry points
37569. Caller has already eaten the first character ('(') which we don't need.
37570. C recursion check.
37571. standard JSON; array gaps
37572. catches EOF (0x00)
37573. **comment:** XXX: refactor into an internal helper, pretty awkward
- label:** code-design
37574. EOF (-1) will be cast to an unsigned value first * and then re-cast for the switch. In any case, it * will match the default case (syntax error).
37575. idx_space
37576. Normal object which doesn't get automatically coerced to a * primitive value. Functions are checked for specially. The * primitive value coercions for Number, String, Pointer, and * Boolean can't result in functions so suffices to check here.
37577. -> [... val']
37578. Caller must ensure 'tv' is indeed a double and not a fastint!
37579. Must set 'length' explicitly when using duk_xdef_prop_xxx() to * set the values.
37580. Caller has already eaten the first character ('!') which we don't need.
37581. Shared entry handling for object/array serialization.
37582. **comment:** XXX: There are no format strings in duk_config.h yet, could add * one for formatting duk_int64_t. For now, assumes "%lld" and that * "long long" type exists. Could also rely on C99 directly but that * won't work for older MSVC.
- label:** code-design
37583. Previous entry was inside visited[], nothing to do.
37584. slow path decode
37585. We come here for actual aborts (like encountering .toJSON()) * but also for recursion/loop errors. Bufwriter size can be * kept because we'll probably need at least as much as we've * allocated so far.
37586. **comment:** This fast path is pretty marginal in practice. * XXX: candidate for removal.
- label:** code-design
37587. [... proplist enum_obj key val]
37588. **comment:** XXX: substring in-place at idx_place?
- label:** code-design
37589. * Process replacer/proplist (2nd argument to JSON.stringify)
37590. Gap in array; check for inherited property, * bail out if one exists. This should be enough * to support gappy arrays for all practical code.
37591. handle comma and closing brace
37592. '-0'
37593. * Process space (3rd argument to JSON.stringify)
37594. caller checks
37595. [... holder name val enum obj_key]
37596. XXX: duk_dup_unvalidated(ctx, -2) etc.
37597. accept anything, expect first value (EOF will be * caught by key parsing below).
37598. First character has already been eaten and checked by the caller. * We can scan until a NUL in stridx string because no built-in strings * have internal NULS.
37599. Assume that the native representation never contains a closing * parenthesis.
37600. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor).
- label:** code-design
37601. get_value
37602. -> [... res]
37603. Initial '{' has been checked and eaten by caller.

37604. [... key]
37605. **comment:** This seems faster than emitting bytes one at a time and * then potentially rewinding.
 label: code-design
37606. pop enum
37607. DUK_USE_JSON_STRINGIFY_FASTPATH
37608. leave stack unbalanced on purpose
37609. Most input bytes go through here.
37610. -> [... holder name val]
37611. [... buf loop (proplist) (gap)]
37612. disabled for now
37613. extra coercion of strings is OK
37614. ascii fast path: avoid decoding utf-8
37615. We only get here when doing non-standard JSON encoding
37616. idx_replacer
37617. Handle both full and partial slice (as long as covered).
37618. End of input (NUL) goes through slow path and causes SyntaxError.
37619. Assume arrays are dense in the fast path.
37620. handle comma and closing bracket
37621. Maximum expansion per input byte is 6: * - invalid UTF-8 byte causes "\uXXXX" to be emitted (6/1 = 6). * - 2-byte UTF-8 encodes as "\uXXXX" (6/2 = 3). * - 4-byte UTF-8 encodes as "\Uxxxxxxxx" (10/4 = 2.5).
37622. **comment:** XXX: keys is an internal object with all keys to be processed * in its (gapless) array part. Because nobody can touch the keys * object, we could iterate its array part directly (keeping in mind * that it can be reallocated).
 label: code-design
37623. need_bytes may be zero
37624. clamp to 10 chars
37625. A lightfunc might also inherit a . toJSON() so just bail out.
37626. **comment:** XXX: Stack discipline is annoying, could be changed in numconv.
 label: code-design
37627. XXX: A fast path for usual integers would be useful when * fastint support is not enabled.
37628. We could use a switch-case for the class number but it turns out * a small if-else ladder on class masks is better. The if-ladder * should be in order of relevancy.
37629. **comment:** This approach is a bit shorter than a straight * if-else-ladder and also a bit faster.
 label: code-design
37630. **comment:** XXX: for real world code, could just ignore array inheritance * and only look at array own properties.
 label: code-design
37631. This loop intentionally does not ensure characters are valid * ([0-9a-fA-F]) because the hex decode call below will do that.
37632. -> [... key val replacer holder]
37633. Select appropriate escape format automatically, and set 'tmp' to a * value encoding both the escape format character and the nybble count: * * (nybble_count << 16) | (escape_char1) | (escape_char2)
37634. * Fast path: assume no mutation, iterate object property tables * directly; bail out if that assumption doesn't hold.
37635. [... holder key] -> [... holder]
37636. **comment:** Fastint range is signed 48-bit so longest value is -2^47 = -140737488355328 * (16 chars long), longest signed 64-bit value is -2^63 = -9223372036854775808 * (20 chars long). Alloc space for 64-bit range to be safe.
 label: code-design
37637. Same coercion behavior as for Number.
37638. **comment:** For objects JSON.stringify() only looks for own, enumerable * properties which is nice for the fast path here. * * For arrays JSON.stringify() uses [[Get]] so it will actually * inherit properties during serialization! This fast path * supports gappy arrays as long as there's no actual inherited * property (which might be a getter etc). * * Since recursion only happens for objects, we can have both * recursion and loop checks here. We use a simple, depth-limited * loop check in the fast path because the object-based tracking * is very slow (when tested, it accounted for 50% of fast path * execution time for input data with a lot of small objects!).
 label: code-design
37639. DUK_USE_JSON_DECNUMBER_FASTPATH
37640. -> [... voidp true]
37641. If any non-Array value had enumerable virtual own * properties, they should be serialized here. Standard * types don't.
37642. eat closing brace
37643. Catches EOF of JSON input.
37644. number
37645. radix
37646. end-of-input breaks
37647. **comment:** XXX: would be nice to enumerate an object at specified index
 label: code-design
37648. DUK_USE_JSON_QUOTESTRING_FASTPATH
37649. slow path is shared
37650. **comment:** bytes of indent available for copying
 label: code-design
37651. [... target] -> [... target keys]
37652. [... str]
37653. -> [... value]
37654. object or array
37655. DUK_USE_PREFER_SIZE
37656. overflow not possible, buffer limits
37657. XXX: duk_push_uint_string()
37658. DUK_USE_JX || DUK_USE_JC
37659. E5 Section 15.12.3, main algorithm, step 4.b.ii steps 1-4.
37660. Array length is larger than 'a size'. This shouldn't * happen in practice. Bail out just in case.
37661. not undefined
37662. Restore stack top after unbalanced code paths.
37663. **comment:** When ptr == NULL, the format argument is unused.
 label: code-design
37664. * Context init
37665. -> [... val root val]
37666. [... obj key val]
37667. [...]
37668. * Stringify implementation.
37669. NUL also comes here. Comparison order matters, 0x20 * is most common whitespace.
37670. [... ptr]
37671. double quote or backslash
37672. [... arr]
37673. **comment:** XXX: how to figure correct size?

label: code-design

37674. c recursion check

37675. catches EOF (NUL) and initial comma

37676. The #ifdef clutter here needs to handle the three cases: * (1) JX+JC, (2) JX only, (3) JC only.

37677. eat trailing comma

37678. disabled

37679. The #ifdef clutter here handles the JX/JC enable/disable * combinations properly.

37680. [... key val]

37681. Decode failed.

37682. end switch

37683. Result is undefined.

37684. **comment:** XXX: non-callable .toJSON() doesn't need to cause an abort * but does at the moment, probably not worth fixing.

label: code-design

37685. [... proplist]

37686. digits

37687. accept anything, expect first value (EOF will be * caught by duk__dec_value() below.

37688. [... key val] -> [...]

37689. minval

37690. -> [... key]

37691. [... obj key]

37692. Here again we parse bytes, and non-ASCII UTF-8 will cause end of * parsing (which is correct except if there are non-shortest encodings). * There is also no need to check explicitly for end of input buffer as * the input is NUL padded and NUL will exit the parsing loop. * * Because no unescaping takes place, we can just scan to the end of the * plain string and intern from the input buffer.

37693. The JO(value) operation: encode object. * * Stack policy: [object] -> [object].

37694. Caller has already eaten the first char so backtrack one byte.

37695. **comment:** bytes of indent still needed

label: code-design

37696. nop

37697. If XUTF-8 decoding fails, treat the offending byte as a codepoint directly * and go forward one byte. This is of course very lossy, but allows some kind * of output to be produced even for internal strings which don't conform to * XUTF-8. All standard Ecmascript strings are always CESU-8, so this behavior * does not violate the Ecmascript specification. The behavior is applied to * all modes, including Ecmascript standard JSON. Because the current XUTF-8 * decoding is not very strict, this behavior only really affects initial bytes * and truncated codepoints. * * Another alternative would be to scan forwards to start of next codepoint * (or end of input) and emit just one replacement codepoint.

37698. Initial 'T' has been checked and eaten by caller.

37699. We already ate 'x', so backup one byte.

37700. Convert buffer to result string.

37701. Steps 8-10 have been merged to avoid a "partial" variable.

37702. Shared exit handling for object/array serialization.

37703. first character has been matched

37704. There is no need to NUL delimit the sscanf() call: trailing garbage is * ignored and there is always a NUL terminator which will force an error * if no error is encountered before it. It's possible that the scan * would scan further than between [js_ctx->p,p[though and we'd advance * by less than the scanned value. * * Because pointers are platform specific, a failure to scan a pointer * results in a null pointer which is a better placeholder than a missing * value or an error.

37705. [arg1 ... argN this loggerLevel loggerName buffer]

37706. the stack is unbalanced here on purpose; we only rely on the * initial two values: [name this].

37707. **comment:** XXX: better multiline

label: code-design

37708. [arg undefined]

37709. * Logging support

37710. [arg toLogString]

37711. Default function to format objects. Tries to use toLogString() but falls * back to toString(). Any errors are propagated out without catching.

37712. **comment:** Stripping the filename might be a good idea * (""/foo/bar/quux.js" -> logger name "quux"), * but now used verbatim.

label: code-design

37713. separators: space, space, colon

37714. Do debugger forwarding before raw() because the raw() function * doesn't get the log level right now.

37715. **comment:** * Logger arguments are: * * magic: log level (0-5) * this: logger * stack: plain log args * * We want to minimize memory churn so a two-pass approach * is used: first pass formats arguments and computes final * string length, second pass copies strings either into a * pre-allocated and reused buffer (short messages) or into a * newly allocated fixed buffer. If the backend function plays * nice, it won't coerce the buffer to a string (and thus * intern it).

label: code-design

37716. [arg result]

37717. time

37718. * Pass 1

37719. duk_pcall_prop() may itself throw an error, but we're content * in catching the obvious errors (like toLogString() throwing an * error).

37720. **comment:** log level could be popped but that's not necessary

label: code-design

37721. **comment:** XXX: There used to be a shared log buffer here, but it was removed * when dynamic buffer spare was removed. The problem with using * bufwriter is that, without the spare, the buffer gets passed on * as an argument to the raw() call so it'd need to be resized * (reallocated) anyway. If raw() call convention is changed, this * could be made more efficient.

label: code-design

37722. Keep the error as the result (coercing it might fail below, * but we don't catch that now).

37723. Constructor

37724. obj_index

37725. keep default instance

37726. nop

37727. [arg1 ... argN this loggerLevel loggerName buffer 'raw' buffer]

37728. * Log level check

37729. don't set 'n' at all, inherited value is used as name

37730. call: this(fmt(arg)

37731. [name this]

37732. 3-letter log level strings

37733. [arg1 ... argN this loggerLevel loggerName]

37734. this.raw(buffer)

37735. * Pass 2

37736. level string

37737. **comment:** Default function to write a formatted log line. Writes to stderr, * appending a newline to the log line. * * The argument is a buffer whose visible size contains the log message. * This function should avoid coercing the buffer to a string to avoid * string table traffic.

label: code-design

37738. Line format: <time> <entryLev> <loggerName>: <msg>

37739. **comment:** Calling as a non-constructor is not meaningful.

label: code-design

37740. [arg1 ... argN this]

37741. sep (even before first one)

37742. nargs

37743. **comment:** Automatic defaulting of logger name from caller. This would * work poorly with tail calls, but constructor calls are currently * never tail calls, so tail calls are not an issue now.

label: code-design

37744. **comment:** Call this.raw(msg); look up through the instance allows user to override * the raw() function in the instance or in the prototype for maximum * flexibility.

label: code-design

37745. **comment:** When formatting an argument to a string, errors may happen from multiple * causes. In general we want to catch obvious errors like a toLogString() * throwing an error, but we don't currently try to catch every possible * error. In particular, internal errors (like out of memory or stack) are * not caught. Also, we expect Error.toString() to not throw an error.

label: code-design

37746. **comment:** XXX: sanitize to printable (and maybe ASCII)

label: code-design

37747. loggerName

37748. **comment:** Log frontend shared helper, magic value indicates log level. Provides * frontend functions: trace(), debug(), info(), warn(), error(), fatal(). * This needs to have small footprint, reasonable performance, minimal * memory churn, etc.

label: code-design

37749. [arg1 ... argN this loggerLevel loggerName 'fmt' arg]

37750. A stubbed built-in is useful for e.g. compilation torture testing with BCC.

37751. fmin() with args -0 and +0 is not guaranteed to return * -0 as Ecmascript requires.

37752. Not odd, or y == -Infinity

37753. fmax() with args -0 and +0 is not guaranteed to return * +0 as Ecmascript requires.

37754. Wrappers for calling standard math library methods. These may be required * on platforms where one or more of the math built-ins are defined as macros * or inline functions and are thus not suitable to be used as function pointers.

37755. Numbers half-way between integers must be rounded towards +Infinity, * e.g. -3.5 must be rounded to -3 (not -4). When rounded to zero, zero * sign must be set appropriately. E5.1 Section 15.8.2.15. ** Note that ANSI C round() is "round to nearest integer, away from zero", * which is incorrect for negative values. Here we make do with floor().

37756. DUK_USE_MATH_BUILTIN

37757. See test-bug-netbsd-math-pow.js: NetBSD 6.0 on x86 (at least) does not * correctly handle some cases where x=+/-0. Specific fixes to these * here.

37758. * Note: fmax() does not match the E5 semantics. E5 requires * that if -any- input to Math.max() is a NaN, the result is a * NaN. fmax() will return a NaN only if both- inputs are NaN. * Same applies to fmin(). * * Note: every input value must be coerced with ToNumber(), even * if we know the result will be a NaN anyway: ToNumber() may have * side effects for which even order of evaluation matters.

37759. * x is finite and non-zero ** -1.6 -> floor(-1.1) -> -2 * -1.5 -> floor(-1.0) -> -1 (towards +Inf) * -1.4 -> floor(-0.9) -> -1 * -0.5 -> -0.0 (special case) * -0.1 -> -0.0 (special case) * +0.1 -> +0.0 (special case) * +0.5 -> floor(+1.0) -> 1 (towards +Inf) * +1.4 -> floor(+1.9) -> 1 * +1.5 -> floor(+2.0) -> 2 (towards +Inf) * +1.6 -> floor(+2.1) -> 2

37760. order must match constants in genbuiltins.py

37761. DUK_USE_AVOID_PLATFORM_FUNCPTRS

37762. * Math built-ins

37763. +0.5 is handled by floor, this is on purpose

37764. Math.pow(-0,y) where y<0 should be: * - -Infinity if y<0 and an odd integer * - Infinity otherwise * NetBSD pow() returns -Infinity for all finite y<0. The * if- clause also catches y == -Infinity (which works even * without the fix).

37765. Note: not normalized, but duk_push_number() will normalize

37766. **comment:** XXX: what's the safest way of creating a negative zero?

label: code-design

37767. Math.pow(+0,y) should be Infinity when y<0. NetBSD pow() * returns -Infinity instead when y is <0 and finite. The * if-clause also catches y == -Infinity (which works even * without the fix).

37768. The ANSI C pow() semantics differ from Ecmascript. ** E.g. when x==1 and y is +/- infinite, the Ecmascript required * result is NaN, while at least Linux pow() returns 1.

37769. fmod() return value has same sign as input (negative) so * the result here will be in the range]-2,0], 1 indicates * odd. If x is -Infinity, NaN is returned and the odd check * always concludes "not odd" which results in desired outcome.

37770. * Use static helpers which can work with math.h functions matching * the following signatures. This is not portable if any of these math * functions is actually a macro. ** Typing here is intentionally 'double' wherever values interact with * the standard library APIs.

37771. The specification has quite awkward order of coercion and * checks for toPrecision(). The operations below are a bit * reordered, within constraints of observable side effects.

37772. * E5 Section 15.7.2.1 requires that the constructed object * must have the original Number.prototype as its internal * prototype. However, since Number.prototype is non-writable * and non-configurable, this doesn't have to be enforced here: * The default object (bound to 'this') is OK, though we have * to change its class. * * Internal value set to ToNumber(arg) or +0; if no arg given, * ToNumber(undefined) = NaN, so special treatment is needed * (above). String internal value is immutable.

37773. digits

37774. leading digit + fractions

37775. Used when precision is undefined; also used for NaN (-> "NaN"), * and +/- infinity (-> "Infinity", "-Infinity").

37776. XXX: shared helper fortoFixed(), toExponential(), toPrecision()

37777. * Number built-ins

37778. radix

37779. for side effects

37780. Number built-in accepts a plain number or a Number object (whose * internal value is operated on). Other types cause TypeError.

37781. **comment:** XXX: helper

label: code-design

37782. -> [val obj val]

37783. flags

37784. no return value -> don't replace created value

37785. **comment:** XXX: just use toString() for now; permitted although not recommended. * nargs==1, so radix is passed to toString().

label: code-design

37786. * toFixed(), toExponential(), toPrecision()

37787. **comment:** * The Number constructor uses ToNumber(arg) for number coercion * (coercing an undefined argument to NaN). However, if the * argument is not given at all, +0 must be used instead. To do * this, a vararg function is used.

label: code-design

37788. target

37789. [hobject props enum(props) key desc]

37790. fall thru

37791. Lightfunc, always success.

37792. [hobject props enum(props)]

37793. **comment:** XXX: missing trap result validation for non-configurable target keys * (must be present), for non-extensible target all target keys must be * present and no extra keys can be present. * http://www.ecma-international.org/ecma-262/6.0/#sec-proxy-object-internal-methods-and-internal-slots-ownpropertykeys

label: code-design

37794. Shared helper for Object.getOwnPropertyNames() and Object.keys(). * Magic: 0=getOwnPropertyNames, 1=Object.keys.
37795. -> [obj trap_result]
37796. enum_flags
37797. magic: 0=getter call, 1=Object.getPrototypeOf
37798. * Validate and convert argument property descriptor (an EcmaScript * object) into a set of defprop_flags and possibly property value, * getter, and/or setter values on the value stack. * Lightfunc set/get values are coerced to full Functions.
37799. **comment:** XXX: for Object.keys() the [[OwnPropertyKeys]] result (trap result) * should be filtered so that only enumerable keys remain. Enumerability * should be checked with [[GetOwnProperty]] on the original object * (i.e., the proxy in this case). If the proxy has a getOwnPropertyDescriptor * trap, it should be triggered for every property. If the proxy doesn't have * the trap, enumerability should be checked against the target object instead. * We don't do any of this now, so Object.keys() and Object.getOwnPropertyNames() * return the same result now for proxy traps. We still do clean up the trap * result, so that Object.keys() and Object.getOwnPropertyNames() will return a * clean array of strings without gaps.
label: code-design
37800. **comment:** XXX: no need for indirect call
label: code-design
37801. Just call the "original" Object.defineProperties() to * finish up.
37802. Pointer and buffer primitive values are treated like other * primitives values which have a fully fledged object counterpart: * promote to an object value. Lightfuncs are coerced with * ToObject() even they could also be returned as is.
37803. Loop prevention
37804. -> [hobject props]
37805. h_new_proto may be NULL
37806. -> [O toString O]
37807. [obj Properties]
37808. [obj trap_result res_arr]
37809. properties object
37810. Careful with reachability here: don't pop 'obj' before pushing * proxy target.
37811. [obj trap_result res_arr propname]
37812. **comment:** TODO: implement Proxy object support here
label: requirement
37813. ignore_loop
37814. required_desc_flags
37815. get_value
37816. Shared helper to implement ES6 Object.setPrototypeOf and * Object.prototype.__proto__ setter. * * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__ * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.setprototypeof
37817. * Object built-ins
37818. * Use Object.defineProperty() helper for the actual operation.
37819. Preliminaries for __proto__ and setPrototypeOf (E6 19.1.2.18 steps 1-4); * magic: 0=setter call, 1=Object.setPrototypeOf
37820. [obj handler trap]
37821. h is NULL for lightfunc
37822. * Return target object
37823. A non-extensible object cannot gain any more properties, * so this is a good time to compact.
37824. **comment:** XXX: tail call: return duk_push_false(ctx)
label: code-design
37825. [[SetPrototypeOf]] standard behavior, E6 9.1.2
37826. Sealed and frozen objects cannot gain any more properties, * so this is a good time to compact them.
37827. [obj key desc]
37828. [hobject props enum(props) key desc value? getter? setter?]
37829. frozen and sealed
37830. **comment:** Lightfuncs are currently supported by coercing to a temporary * Function object; changes will be allowed (the coerced value is * extensible) but will be lost.
label: code-design
37831. Ignore the normalize/validate helper outputs on the value stack, * they're popped automatically.
37832. **comment:** XXX: for Object.keys() we should check enumerability of key
label: code-design
37833. * Two pass approach to processing the property descriptors. * On first pass validate and normalize all descriptors before * any changes are made to the target object. On second pass * make the actual modifications to the target object. * * Right now we'll just use the same normalize/validate helper * on both passes, ignoring its outputs on the first pass.
37834. Object.keys
37835. **comment:** XXX: should the API call handle this directly, i.e. attempt * to duk_push_hobject(ctx, null) would push a null instead? * (On the other hand 'undefined' would be just as logical, but * not wanted here.)
label: code-design
37836. __proto__ setter returns 'undefined' on success unlike the * setPrototypeOf() call which returns the target object.
37837. [O Properties obj]
37838. * Return target object.
37839. idx_desc
37840. retval for success path
37841. Lightfunc handling by ToObject() coercion.
37842. -> [obj trap handler target]
37843. nargs
37844. DUK_USE_ES6_PROXY
37845. is_frozen
37846. Object.getOwnPropertyNames
37847. **comment:** E5.1 Section 15.2.4.6, step 3.a, lookup proto once before compare. * Prototype loops should cause an error to be thrown.
label: code-design
37848. Shared helper to implement Object.getPrototypeOf and the ES6 * Object.prototype.__proto__ getter. * * http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__
37849. **comment:** XXX: call method tail call?
label: code-design
37850. Pointer object internal value is immutable
37851. * Constructor
37852. * Pointer built-ins
37853. nop
37854. Must be a "pointer object", i.e. class "Pointer"
37855. Note: unbalanced stack on purpose
37856. * toString(), valueOf()
37857. **comment:** XXX: this behavior is quite useless now; it would be nice to be able * to create pointer values from e.g. numbers or strings. Numbers are * problematic on 64-bit platforms though. Hex encoded strings?
label: code-design
37858. DUK_BUILTIN_PROTOS_H_INCLUDED

37859. * Prototypes for built-in functions not automatically covered by the * header declarations emitted by genbuiltins.py.
37860. Helpers exposed for internal use
37861. Buffer size needed for duk.bi_date_format_timeval(). * Accurate value is 32 + 1 for NUL termination: * >>> len('+123456-01-23T12:34:56.123+12:34') * 32 *
 Include additional space to be safe.
37862. Maximum length of CommonJS module identifier to resolve. Length includes * both current module ID, requested (possibly relative) module ID, and a * slash in
 between.
37863. Built-in providers
37864. Make _Target and _Handler non-configurable and non-writable. * They can still be forcibly changed by C code (both user and * Duktape internal), but not by
 Ecmascript code.
37865. Reject a proxy object as the target because it would need * special handler in property lookups. (ES6 has no such restriction)
37866. Proxy target
37867. Proxy handler
37868. DUK_USE_ES6_PROXY
37869. **comment:** XXX: the returned value is exotic in ES6, but we use a * simple object here with no prototype. Without a prototype, * [[DefaultValue]] coercion fails
 which is abit confusing. * No callable check/handling in the current Proxy subset.
 label: code-design
37870. Reject a proxy object as the handler because it would cause * potentially unbounded recursion. (ES6 has no such restriction)
37871. * Proxy built-in (ES6)
37872. replacement handler
37873. flags: "m"
37874. DUK_USE_REGEXP_SUPPORT
37875. * RegExp built-ins
37876. flags: "gim"
37877. [result]
37878. **comment:** A little tricky string approach to provide the flags string. * This depends on the specific flag values in duk_regexp.h, * which needs to be asserted for. In
 practice this doesn't * produce more compact code than the easier approach in use.
 label: code-design
37879. [... pattern flags]
37880. **comment:** This is a cleaner approach and also produces smaller code than * the other alternative. Use duk_require_string() for format * safety (although the source
 property should always exist).
 label: code-design
37881. require to be safe
37882. three flags
37883. prepend regexp to valstack 0 index
37884. **comment:** XXX: much to improve (code size)
 label: code-design
37885. flags: "gi"
37886. [... bytecode escaped_source]
37887. flags: "gm"
37888. [... RegExp]
37889. flags: ""
37890. Called as a function, pattern has [[Class]] "RegExp" and * flags is undefined -> return object as is.
37891. [regexp input]
37892. Else functionality is identical for function call and constructor * call.
37893. flags: "g"
37894. [regexp]
37895. This should not be necessary because no-one should tamper with the * regexp bytecode, but is prudent to avoid potential segfaults if that * were to happen for
 some reason.
37896. flags: "im"
37897. flags: "i"
37898. [regexp source bytecode]
37899. result object is created and discarded; wasteful but saves code space
37900. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array
37901. * The specification uses RegExp [[Match]] to attempt match at specific * offsets. We don't have such a primitive, so we use an actual RegExp * and tweak
 lastIndex. Since the RegExp may be non-global, we use a * special variant which forces global-like behavior for matching.
37902. * Case conversion
37903. **comment:** XXX: It would be nice to build the string directly but ToUint16() * coercion is needed so a generic helper would not be very * helpful (perhaps coerce
 the value stack first here and then * build a string from a duk_tval number sequence in one go?).
 label: code-design
37904. lastIndex already set up for next match
37905. DUK_USE_REGEXP_SUPPORT
37906. Empty searchstring always matches; cpos must be clamped here. * (If q_blen were < 0 due to clamped coercion, it would also be * caught here.)
37907. [regexp string]
37908. **comment:** This loop is optimized for size. For speed, there should be * two separate loops, and we should ensure that memcmp() can be * used without an extra
 "will searchstring fit" check. Doing * the preconditioning for 'p' and 'p_end' is easy but cpos * must be updated if 'p' is wound back (backward scanning).
 label: code-design
37909. The current implementation of localeCompare() is simply a codepoint * by codepoint comparison, implemented with a simple string compare * because UTF-8
 should preserve codepoint ordering (assuming valid * shortest UTF-8 encoding). * * The specification requires that the return value must be related * to the sort
 order: e.g. negative means that 'this' comes before * 'that' in sort order. We assume an ascending sort order.
37910. prefix matches, lengths matter now
37911. min(incl)
37912. not found
37913. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer * stack[4] = regexp match OR match string
37914. For Ecmascript strings, this check can only match for * initial UTF-8 bytes (not continuation bytes). For other * strings all bets are off.
37915. match == search string, by definition
37916. res_obj
37917. ToUint16() coercion is mandatory in the E5.1 specification, but * this non-compliant behavior makes more sense because we support * non-BMP codepoints. Don't
 use CESU-8 because that'd create * surrogate pairs.
37918. match string
37919. empty match (may happen with empty separator) -> bump and continue
37920. * Various
37921. * toString(), valueOf()
37922. when going backwards, we decrement cpos 'early'; * 'p' may point to a continuation byte of the char * at offset 'cpos', but that's OK because we'll * backtrack all
 the way to the initial byte.
37923. trailer
37924. don't allow an empty match at the end of the string
37925. [... replacer match [captures] match_char_offset input]

37926. XXX: There are several limitations in the current implementation for * strings with $\geq 0x80000000UL$ characters. In some cases one would need * to be able to represent the range [-0xffffffff, 0xffffffff] and so on. * Generally character and byte length are assumed to fit into signed 32 * bits ($< 0x80000000UL$). Places with issues are not marked explicitly * below in all cases, look for signed type usage (duk_int_t etc) for * offsets/lengths.

37927. **comment:** XXX: could share code with duk_js_ops.c, duk_js_compare_helper
label: code-design

37928. * indexOf() and lastIndexOf()
37929. ch1 = (r_increment << 8) + byte
37930. * If matching with a regexp: * - non-global RegExp: lastIndex not touched on a match, zeroed * on a non-match * - global RegExp: on match, lastIndex will be updated by regexp * executor to point to next char after the matching part (so that * characters in the matching part are not matched again) * * If matching with a string: * - always non-global match, find first occurrence * * We need: * - The character offset of start-of-match for the replacer function * - The byte offsets for start-of-match and end-of-match to implement * the replacement values \$\$, \$` and \$', and to copy non-matching * input string portions (including header and trailer) verbatim. * * NOTE: the E5.1 specification is a bit vague how the RegExp should * behave in the replacement process; e.g. is matching done first for * all matches (in the global RegExp case) before any replacer calls * are made? See: test-bi-string-proto-replace.js for discussion.

37931. return as is
37932. repl_value
37933. Not found. Empty string case is handled specially above.
37934. stack[0] = regexp * stack[1] = string
37935. * substring(), substr(), slice()
37936. -> [... regexp string] -> [... res_obj]
37937. [start end str]
37938. never here, but fall through
37939. at index 1
37940. **comment:** this is relatively expensive
label: code-design
37941. no issues with memcmp() zero size, even if broken
37942. switch (ch2)
37943. lastIndex is initialized to zero by new RegExp()
37944. String constructor needs to distinguish between an argument not given at all * vs. given as 'undefined'. We're a vararg function to handle this properly.
37945. pop regexp res_obj or match string
37946. unconditionally
37947. start match from beginning
37948. h_match is borrowed, remains reachable through match_obj
37949. if (is_regexp)
37950. **comment:** XXX: faster implementation
label: code-design
37951. Add trailer if: * a) non-empty input * b) empty input and no (zero size) match found (step 11)
37952. If the separator is a RegExp, make a "clone" of it. The specification * algorithm calls [[Match]] directly for specific indices; we emulate this * by tweaking lastIndex and using a "force global" variant of duk_regexp_match() * which will use global-style matching even when the RegExp itself is non-global.
37953. DUK_USE_SECTION_B
37954. Zero size compare not an issue with DUK_MEMCMP.
37955. out_clamped
37956. **comment:** XXX: could improve bufwriter handling to write multiple codepoints * with one ensure call but the relative benefit would be quite small.
label: code-design
37957. empty separator can always match
37958. leading byte of match string
37959. **comment:** XXX: the current implementation works but is quite clunky; it compiles * to almost 1,4kB of x86 code so it needs to be simplified (better approach, * shared helpers, etc). Some ideas for refactoring: * * - a primitive to convert a string into a regexp matcher (reduces matching * code at the cost of making matching much slower) * - use replace() as a basic helper for match() and split(), which are both * much simpler * - API call to get_prop and to_boolean
label: code-design
37960. track utf-8 non-continuation bytes
37961. **comment:** XXX: optional check, match_caps is zero if no regexp, * so dollar will be interpreted literally anyway.
label: code-design
37962. input size is good output starting point
37963. match_caps == 0 without regexps
37964. **comment:** XXX: very messy now, but works; clean up, remove unused variables (nominally * used so compiler doesn't complain).
label: code-design
37965. unconditionally; is_global==0
37966. Unlike non-obsolete String calls, substr() algorithm in E5.1 * specification will happily coerce undefined and null to strings * ("undefined" and "null").
37967. string limits
37968. Easiest way to implement the search required by the specification * is to do a RegExp test() with lastIndex forced to zero. To avoid * side effects on the argument, "clone" the RegExp if a RegExp was * given as input. * * The global flag of the RegExp should be ignored; setting lastIndex * to zero (which happens when "cloning" the RegExp) should have an * equivalent effect.
37969. regexp res_obj is at index 4
37970. [... re_obj input]
37971. max(incl)
37972. Must be a "string object", i.e. class "String"
37973. The spec algorithm first does "R = ToString(separator)" before checking * whether separator is undefined. Since this is side effect free, we can * skip the ToString() here.
37974. indexOf: NaN should cause pos to be zero. * lastIndexOf: NaN should cause pos to be +Infinity * (and later be clamped to len).
37975. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer
37976. * Character and charcode access
37977. no action
37978. default case
37979. initial estimate for ASCII only codepoints
37980. * replace()
37981. cannot have >4G captures
37982. while repl
37983. return 'res_obj'
37984. never an empty match, so step 13.c.iii can't be triggered
37985. for
37986. no issues with empty memcmp()
37987. * Constructor
37988. Shared helper for match() steps 3-4, search() steps 3-4.
37989. track cpos while scanning
37990. 0=indexOf, 1=lastIndexOf
37991. regexp res_obj is at offset 4
37992. [start length str]
37993. **comment:** Global case is more complex.

label: code-design

37994. * String built-ins
37995. return 'res_arr' or 'null'
37996. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array * stack[4] = regexp res_obj (if is_regexp)
37997. This should be equivalent to match() algorithm step 8.f.iii.2: * detect an empty match and allow it, but don't allow it twice.
37998. combines steps 2 and 5; -len ensures max() not needed for step 5
37999. force_new
38000. No arr_idx update or limit check
38001. set to 1 if any match exists (needed for empty input special case)
38002. * split()
38003. Combined step 11 (empty string special case) and 14-15.
38004. Avoid using RegExp.prototype methods, as they're writable and * configurable and may have been changed.
38005. match followed by capture(s)
38006. -> [... res_obj]
38007. empty match -> bump and continue
38008. -> [res_obj]
38009. [regexp string res_arr]
38010. String object internal value is immutable
38011. index
38012. repl string scan
38013. ditto
38014. Note: unbalanced stack on purpose
38015. **comment:** XXX: any chance of merging these three similar but still slightly * different algorithms so that footprint would be reduced?
label: code-design
38016. ensure full memcmp() fits in while
38017. Use match charlen instead of bytelen, just in case the input and * match codepoint encodings would have different lengths.
38018. **comment:** combines steps 3, 6; step 7 is not needed
label: requirement
38019. [... regexp input] -> [res_obj]
38020. -> [... repl_value]
38021. single match always
38022. constrained by string length
38023. if (is_repl_func)
38024. Handle empty separator case: it will always match, and always * triggers the check in step 13.c.iii initially. Note that we * must skip to either end of string or start of first codepoint, * skipping over any continuation bytes! * * Don't allow an empty string to match at the end of the input.
38025. duk_concat() coerces arguments with ToString() in correct order
38026. input string scan
38027. undefined -> skip (replaced with empty)
38028. [... RegExp val] -> [... res]
38029. The implementation for computing of start_pos and end_pos differs * from the standard algorithm, but is intended to result in the exactly * same behavior. This is not always obvious.
38030. * Thread state and calling context checks
38031. * Constructor
38032. [value]
38033. * Resume a thread. * * The thread must be in resumable state, either (a) new thread which hasn't * yet started, or (b) a thread which has previously yielded. This method * must be called from an Ecmascript function. * * Args: * - thread * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.
38034. in yielder's context
38035. return thread
38036. * Thread builtins
38037. never here
38038. no pre-checks now, assume a previous yield() has left things in * tip-top shape (longjmp handler will assert for these).
38039. XXX: need a duk_require_func_or_lfunc_coerce()
38040. in resumer's context
38041. Note: the only yield-preventing call is Duktape.Thread.yield(), hence check for 1, not 0
38042. value (error) is at stack top
38043. us
38044. * Process yield * * After longjmp(), processing continues in bytecode executor longjmp * handler, which will e.g. update thr->resumer to NULL.
38045. caller
38046. push initial function call to new thread stack; this is * picked up by resume().
38047. Note: there is no requirement that: 'thr->callstack_preventcount == 1' * like for yield.
38048. Note: cannot be a bound function either right now, * this would be easy to relax though.
38049. call is from executor, so we know we have a jmpbuf
38050. side effects
38051. execution resumes in bytecode executor
38052. * The error object has been augmented with a traceback and other * info from its creation point -- usually the current thread. * The error handler, however, is called right before throwing * and runs in the yielder's thread.
38053. * Yield the current thread. * * The thread must be in yieldable state: it must have a resumer, and there * must not be any yield-preventing calls (native calls and constructor calls, * currently) in the thread's call stack (otherwise a resume would not be * possible later). This method must be called from an Ecmascript function. * * Args: * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.
38054. lj value2: thread
38055. **comment:** should never be zero, because we (Duktape.Thread.yield) are on the stack
label: code-design
38056. lj value1: value
38057. Further state-dependent pre-checks
38058. [thread value]
38059. * The error object has been augmented with a traceback and other * info from its creation point -- usually another thread. The * error handler is called here right before throwing, but it also * runs in the resumer's thread. It might be nice to get a traceback * from the resumee but this is not the case now.
38060. * Type error thrower, E5 Section 13.2.3.
38061. DUK_USE_ROM_OBJECTS
38062. * Automatically generated by genbuiltins.py, do not edit!
38063. DUK_USE_ROM_STRINGS
38064. native functions: 149
38065. DUK_USE_BUILTIN_INITJS
38066. 'length'
38067. !DUK_SINGLE_FILE
38068. 'I'
38069. 'Arguments'

38070. 'toLogString'
38071. '\xffNext'
38072. '\xffBytecode'
38073. '\xffLexenv'
38074. 'Date'
38075. 'instanceof'
38076. 'has'
38077. 'do'
38078. 'var'
38079. 'exports'
38080. 'pointer'
38081. 'void'
38082. 'string'
38083. 'protected'
38084. 'filename'
38085. 'try'
38086. 'continue'
38087. 'catch'
38088. 'toJSON'
38089. 'if'
38090. 'index'
38091. 'deleteProperty'
38092. 'value'
38093. 'Object'
38094. 'switch'
38095. 'toGMTString'
38096. 'let'
38097. '[...]
38098. 'enumerable'
38099. 'Uint32Array'
38100. DUK_USE_ROM_STRINGS
38101. ''
38102. 'enumerate'
38103. 'with'
38104. 'join'
38105. 'resume'
38106. 'debugger'
38107. 'get'
38108. ','
38109. '{"_inf":true}'
38110. '\xffFinalizer'
38111. '\xffArgs'
38112. 'Error'
38113. 'Float64Array'
38114. 'enum'
38115. 'default'
38116. '\xffRegbase'
38117. 'Thread'
38118. '\xffThread'
38119. 'import'
38120. 'set'
38121. 'export'
38122. 'prototype'
38123. 'ObjEnv'
38124. 'jx'
38125. 'fmt'
38126. 'typeof'
38127. 'Math'
38128. '-Infinity'
38129. 'name'
38130. 'BYTES_PER_ELEMENT'
38131. 'info'
38132. 'interface'
38133. 'class'
38134. 'Boolean'
38135. 'Buffer'
38136. 'defineProperty'
38137. '\xffVarmap'
38138. 'Uint16Array'
38139. To convert a heap stridx to a token number, subtract * DUK_STRIDX_START_RESERVED and add DUK_TOK_START_RESERVED.
38140. 'Infinity'
38141. '\xffFormals'
38142. 'implements'
38143. '\xffHandler'
38144. '{"_undef":true}'
38145. '\xffSource'
38146. '{_func:true}'
38147. 'env'
38148. 'toLocaleString'
38149. '\n'
38150. 'fatal'
38151. 'hex'
38152. 'trace'
38153. 'jc'
38154. 'case'
38155. 'toUTCString'
38156. '(?:)'
38157. '\xffTarget'

38158. 'private'
38159. 'Invalid Date'
38160. 'multiline'
38161. 'errCreate'
38162. 'warn'
38163. 'finally'
38164. 'ownKeys'
38165. 'configurable'
38166. 'caller'
38167. 'undefined'
38168. 'byteLength'
38169. 'lineNumber'
38170. 'Int8Array'
38171. 'data'
38172. 'type'
38173. '-0'
38174. '{"_ninf":true}'
38175. 'modLoaded'
38176. 'true'
38177. 'fileName'
38178. 'Uint8Array'
38179. 'break'
38180. 'Undefined'
38181. "
38182. 'setPrototypeOf'
38183. '\xffVarenv'
38184. 'number'
38185. '{"_func":true}'
38186. 'return'
38187. 'throw'
38188. 'valueOf'
38189. 'buffer'
38190. 'DecEnv'
38191. 'writable'
38192. DUK_USE_ROM_OBJECTS
38193. 'yield'
38194. 'id'
38195. 'callee'
38196. 'ArrayBuffer'
38197. 'Null'
38198. '\xffPc2line'
38199. '\xffMap'
38200. 'static'
38201. 'package'
38202. 'ignoreCase'
38203. 'message'
38204. 'NaN'
38205. 'byteOffset'
38206. 'extends'
38207. 'source'
38208. 'constructor'
38209. 'clog'
38210. 'raw'
38211. 'Int16Array'
38212. 'base64'
38213. 'in'
38214. 'Array'
38215. 'pc'
38216. 'errThrow'
38217. 'while'
38218. 'toISOString'
38219. 'Float32Array'
38220. 'object'
38221. 'require'
38222. 'compile'
38223. 'const'
38224. 'false'
38225. 'null'
38226. 'boolean'
38227. 'Int32Array'
38228. 'new'
38229. 'Function'
38230. 'stack'
38231. '\xffThis'
38232. 'toString'
38233. 'debug'
38234. '\xffCallee'
38235. 'Uint8ClampedArray'
38236. 'arguments'
38237. 'n'
38238. 'modSearch'
38239. 'for'
38240. DUK_USE_BUILTIN_INITJS
38241. 'eval'
38242. 'function'
38243. '\xffValue'
38244. 'DataView'
38245. 'error'

38246. `{"_nan":true}`
38247. `'delete'`
38248. * Automatically generated by genbuiltins.py, do not edit!
38249. exclusive endpoint
38250. `'RegExp'`
38251. `'this'`
38252. `'public'`
38253. `'__proto__'`
38254. `'\x0ffTracedata'`
38255. `'input'`
38256. `'super'`
38257. `'lastIndex'`
38258. `'JSON'`
38259. `'else'`
38260. `'Number'`
38261. `'global'`
38262. `DUK_BUILTINS_H_INCLUDED`
38263. `'String'`
38264. `'Pointer'`
38265. This should not really happen, but would indicate x64.
38266. **comment:** Minimize warnings for unused internal functions with GCC >= 3.1.1 and * Clang. Based on documentation it should suffice to have the attribute * in the declaration only, but in practice some warnings are generated unless * the attribute is also applied to the definition.
label: code-design
38267. Emscripten (provided explicitly by user), improve if possible
38268. Both MinGW and MSVC have a 64-bit type.
38269. * Platform autodetection
38270. --- TinyC ---
38271. mint lib is missing these
38272. SuperH
38273. --- x64 ---
38274. MSVC does not have sys/param.h
38275. --- Motorola 68k ---
38276. since gcc-2.5
38277. --- MIPS 64-bit ---
38278. **comment:** XXX: DUK_UNREACHABLE for msvc?
label: code-design
38279. OpenBSD
38280. NetBSD
38281. Apple OSX, iOS
38282. Cannot determine byte order; `__ORDER_PDP_ENDIAN__` is related to 32-bit * integer ordering and is not relevant.
38283. Type for public API calls.
38284. **comment:** Rely as little as possible on compiler behavior for NaN comparison, * signed zero handling, etc. Currently never activated but may be needed * for broken compilers.
label: code-design
38285. --- Generic BSD ---
38286. uclibc may be missing these
38287. FreeBSD
38288. not defined by default
38289. **comment:** Windows, both 32-bit and 64-bit
label: code-design
38290. --- OpenBSD ---
38291. **comment:** * Alternative customization header * * If you want to modify the final DUK_USE_xxx flags directly (without * using the available DUK_OPT_xxx flags), define DUK_OPT_HAVE_CUSTOM_H * and tweak the final flags there.
label: code-design
38292. Cygwin
38293. integer endianness is little on purpose
38294. e.g. `getdate_r`
38295. * Autogenerated defaults
38296. Atari Mint
38297. **comment:** GCC older than 4.6: avoid overflow warnings related to using INFINITY
label: code-design
38298. Most portable, wastes space
38299. GCC/clang inaccurate math would break compliance and probably duk_tval, * so refuse to compile. Relax this if -ffast-math is tested to work.
38300. --- Generic ---
38301. VS2012+ has stdint.h, < VS2012 does not (but it's available for download).
38302. **comment:** * Alignment requirement and support for unaligned accesses * * Assume unaligned accesses are not supported unless specifically allowed * in the target platform. Some platforms may support unaligned accesses * but alignment to 4 or 8 may still be desirable.
label: requirement
38303. Clang
38304. AmigaOS. Neither AMIGA nor `__amigaos__` is defined on VBCC, so user must * define 'AMIGA' manually when using VBCC.
38305. * duk_config.h configuration header generated by genconfig.py. * * Git commit: 0a70d7e4c5227c84e3fed5209828973117d02849 * Git describe: v1.8.0 * Git branch: v1.8-maintenance * * Supported platforms: * - Mac OSX, iPhone, Darwin * - OpenBSD * - Generic BSD * - Atari ST TOS * - AmigaOS * - Windows * - Flashplayer (Crossbridge) * - QNX * - TI-Nspire * - Emscripten * - Linux * - Solaris * - Generic POSIX * - Cygwin * - Generic UNIX * - Generic fallback * * Supported architectures: * - x86 * - x64 * - x32 * - ARM 32-bit * - ARM 64-bit * - MIPS 32-bit * - MIPS 64-bit * - PowerPC 32-bit * - PowerPC 64-bit * - SPARC 32-bit * - SPARC 64-bit * - SuperH * - Motorola 68k * - Emscripten * - Generic * * Supported compilers: * - Clang * - GCC * - MSVC * - Emscripten * - TinyC * - VBCC * - Bruce's C compiler * - Generic *
38306. --- GCC ---
38307. Duktape/C function return value, platform int is enough for now to * represent 0, 1, or negative error code. Must be compatible with * assigning truth values (e.g. `duk_ret_t rc = (foo == bar);`).
38308. Low memory algorithm: separate chaining using arrays, fixed size hash
38309. `_OVERRIDE_DEFINES_`
38310. <http://stackoverflow.com/questions/5919996/how-to-detect-reliably-mac-os-x-ios-linux-windows-in-c-preprocessor>
38311. vbcc + AmigaOS has C99 but no inttypes.h
38312. MSVC
38313. * Check whether or not a packed duk_tval representation is possible. * What's basically required is that pointers are 32-bit values * (`sizeof(void *) == 4`). Best effort check, not always accurate. * If guess goes wrong, crashes may result; self tests also verify * the guess.
38314. --- Linux ---
38315. --- AmigaOS ---

38316. BCC, assume we're on x86.
38317. Pointer size determination based on __WORDSIZE or architecture when * that's not available.
38318. Unsigned index variant.
38319. External provider already defined.
38320. Byte order varies, so rely on autodetect.
38321. DUK_USE_UNION_INITIALIZERS: required from compilers, so no fill-in.
38322. **comment:** Special naming to avoid conflict with e.g. DUK_FREE() in duk_heap.h * (which is unfortunately named). May sometimes need replacement, e.g. * some compilers don't handle zero length or NULL correctly in realloc().
label: code-design
38323. empty
38324. --- MSVC ---
38325. We're generally assuming that we're working on a platform with a 32-bit * address space. If DUK_SIZE_MAX is a typecast value (which is necessary * if SIZE_MAX is missing), the check must be avoided because the * preprocessor can't do a comparison.
38326. 64-bit constants. Since LL / ULL constants are not always available, * use computed values. These values can't be used in preprocessor * comparisons; flag them as such.
38327. --- x86 ---
38328. --- MIPS 32-bit ---
38329. XXX: add feature options to force basic types from outside?
38330. Array index values, could be exact 32 bits. * Currently no need for signed duk_arridx_t.
38331. SIZE_MAX may be missing so use an approximate value for it.
38332. DUK_F_BCC
38333. Complex condition broken into separate parts.
38334. POSIX
38335. If not provided, use safe default for alignment.
38336. * Date provider selection * * User may define DUK_USE_DATE_GET_NOW() etc directly, in which case we'll * rely on an external provider. If this is not done, revert to previous * behavior and use Unix/Windows built-in provider.
38337. Cannot determine byte order.
38338. --- Cygwin ---
38339. !defined(DUK_USE_BYTEORDER) && defined(__BYTE_ORDER__)
38340. More or less standard endianness predefines provided by header files. * The ARM hybrid case is detected by assuming that __FLOAT_WORD_ORDER * will be big endian, see: <http://lists.mysql.com/internals/443>. * On some platforms some defines may be present with an empty value which * causes comparisons to fail: <https://github.com/svaarala/duktape/issues/453>.
38341. !defined(DUK_USE_BYTEORDER)
38342. **comment:** For custom platforms allow user to define byteorder explicitly. * Since endianness headers are not standardized, this is a useful * workaround for custom platforms for which endianness detection * is not directly supported. Perhaps custom hardware is used and * user cannot submit upstream patches.
label: code-design
38343. **comment:** Don't know how to declare unreachable point, so don't do it; this * may cause some spurious compilation warnings (e.g. "variable used * uninitialized").
label: code-design
38344. Good default is a bit arbitrary because alignment requirements * depend on target. See <https://github.com/svaarala/duktape/issues/102>.
38345. --- Windows ---
38346. User provided InitJS.
38347. These have been tested from VS2008 onwards; may work in older VS versions * too but not enabled by default.
38348. Error codes are represented with platform int. High bits are used * for flags and such, so 32 bits are needed.
38349. Intermediate define for 'have inttypes.h'
38350. build is not C99 or C++11, play it safe
38351. GCC: test not very accurate; enable only in relatively recent builds * because of bugs in gcc-4.4 (<http://lists.debian.org/debian-gcc/2010/04/msg00000.html>)
38352. Intel x86 (32-bit), x64 (64-bit) or x32 (64-bit but 32-bit pointers), * define only one of DUK_F_X86, DUK_F_X64, DUK_F_X32. * <https://sites.google.com/site/x32abi/>
38353. Enabled with debug/assertions just so that any issues can be caught.
38354. C99 or compatible
38355. --- Mac OSX, iPhone, Darwin ---
38356. C99 or above
38357. MinGW. Also GCC flags (DUK_F_GCC) are enabled now.
38358. This detection is not very reliable.
38359. Support for 48-bit signed integer duk_tval with transparent semantics.
38360. GCC and Clang provide endianness defines as built-in predefines, with * leading and trailing double underscores (e.g. __BYTE_ORDER__). See * output of "make gcpredes" and "make clangpredes". Clang doesn't * seem to provide __FLOAT_WORD_ORDER__; assume not mixed endian for clang. * <http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html>
38361. 64-bit type detection is a bit tricky. * * ULLONG_MAX is a standard define. __LONG_LONG_MAX__ and __ULONGLONG_MAX__ * are used by at least GCC (even if system headers don't provide ULLONG_MAX). * Some GCC variants may provide __LONG_LONG_MAX__ but not __ULONGLONG_MAX__. * * ULL / LL constants are rejected / warned about by some compilers, even if * the compiler has a 64-bit type and the compiler/system headers provide an * unsupported constant (ULL/LL)! Try to avoid using ULL / LL constants. * As a side effect we can only check that e.g. ULONG_MAX is larger than 32 * bits but can't be sure it is exactly 64 bits. Self tests will catch such * cases.
38362. Convert any input pointer into a "void **", losing a const qualifier. * This is not fully portable because casting through duk_uintptr_t may * not work on all architectures (e.g. those with long, segmented pointers).
38363. Small integers (16 bits or more) can fall back to the 'int' type, but * have a typedef so they are marked "small" explicitly.
38364. **comment:** Most portable, potentially wastes space
label: code-design
38365. PowerPC
38366. --- Solaris ---
38367. **comment:** Technically C99 (C++11) but found in many systems. On some systems * __STDC_LIMIT_MACROS and __STDC_CONSTANT_MACROS must be defined before * including stdint.h (see above).
label: code-design
38368. Missing some obvious constants.
38369. No provider for DUK_USE_DATE_PARSE_STRING(), fall back to ISO 8601 only.
38370. no endian.h or stdint.h
38371. --- Atari ST TOS ---
38372. GCC: assume we have __va_copy() in non-C99 mode.
38373. User forced alignment to 4 or 8.
38374. C99 / C++11 and above: rely on va_copy() which is required.
38375. <http://bellard.org/tcc/tcc-doc.html#SEC9>
38376. Check that architecture is two's complement, standard C allows e.g. * INT_MIN to be -2**31+1 (instead of -2**31).
38377. Based on 'make checkalign' there are no alignment requirements on * Linux MIPS except for doubles, which need align by 4. Alignment * requirements vary based on target though.
38378. AmigaOS on M68k
38379. not configured for DLL build
38380. * You may add overriding #define/#undef directives below for * customization. You of course cannot un-#include or un-typedef * anything; these require direct changes above.

38381. **comment:** cannot detect 64-bit type, not always needed so don't error
label: code-design
38382. --- Generic UNIX ---
38383. std::exception
38384. A few types are assumed to always exist.
38385. Float word order not known, assume not a hybrid.
38386. --- Generic fallback ---
38387. **comment:** --- Generic POSIX ---
label: code-design
38388. no parsing (not an error)
38389. **comment:** ANSI C (various versions) and some implementations require that the * pointer arguments to memset(), memcpy(), and memmove() be valid values * even when byte size is 0 (even a NULL pointer is considered invalid in * this context). Zero-size operations as such are allowed, as long as their * pointer arguments point to a valid memory area. The DUK_MEMSET(), * DUK_MEMCPY(), and DUK_MEMMOVE() macros require this same behavior, i.e.: * (1) pointers must be valid and non-NULL, (2) zero size must otherwise be * allowed. If these are not fulfilled, a macro wrapper is needed. * * http://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0 * http://lists.cs.uiuc.edu/pipermail/l1vmdev/2007-October/011065.html * * Not sure what's the required behavior when a pointer points just past the * end of a buffer, which often happens in practice (e.g. zero size memmoves). * For example, if allocation size is 3, the following pointer would not * technically point to a valid memory byte: * * <-- alloc --> * | 0 | 1 | 2 | * ^ - p=3, points after last valid byte (2)
label: code-design
38390. C++11 or above
38391. SPARC byte order varies so rely on autodetection.
38392. DUK_F_PACKED_TVAL_PROVIDED
38393. http://www.monkey.org/openbsd/archive/ports/0401/msg00089.html
38394. VBCC
38395. sigsetjmp() alternative
38396. GCC, Clang also defines __GNUC__ so don't detect GCC if using Clang.
38397. --- SuperH ---
38398. Pre-C99: va_list type is implementation dependent. This replacement * assumes it is a plain value so that a simple assignment will work. * This is not the case on all platforms (it may be a single-array element, * for instance).
38399. **comment:** The best type for an "all around int" in Duktape internals is "at least * 32 bit signed integer" which is most convenient. Same for unsigned type. * Prefer 'int' when large enough, as it is almost always a convenient type.
label: code-design
38400. uclibc
38401. Explicit marker needed; may be 'defined', 'undefined', 'or 'not provided'.
38402. No provider for DUK_USE_DATE_FORMAT_STRING(), fall back to ISO 8601 only.
38403. **comment:** Many platforms are missing fpclassify() and friends, so use replacements * if necessary. The replacement constants (FP_NAN etc) can be anything but * match Linux constants now.
label: code-design
38404. * Checks for config option consistency (DUK_USE_xxx)
38405. autodetect compiler
38406. Rely on C89 headers only; time.h must be here.
38407. These functions don't currently need replacement but are wrapped for * completeness. Because these are used as function pointers, they need * to be defined as concrete C functions (not macros).
38408. --- TI-Nspire ---
38409. **comment:** We need va_copy() which is defined in C99 / C++11, so an awkward * replacement is needed for pre-C99 / pre-C++11 environments. This * will quite likely need portability hacks for some non-C99 * environments.
label: code-design
38410. AmigaOS on M68K or PPC is always big endian.
38411. Feature option forcing.
38412. --- ARM 64-bit ---
38413. * Feature option handling
38414. **comment:** * Wrapper typedefs and constants for integer types, also sanity check types. * * C99 typedefs are quite good but not always available, and we want to avoid * forcibly redefining the C99 typedefs. So, there are Duktape wrappers for * all C99 typedefs and Duktape code should only use these typedefs. Type * detection when C99 is not supported is best effort and may end up detecting * some types incorrectly. * * Pointer sizes are a portability problem: pointers to different types may * have a different size and function pointers are very difficult to manage * portably. * * http://en.wikipedia.org/wiki/C_data_types#Fixed-width_integer_types * * Note: there's an interesting corner case when trying to define minimum * signed integer value constants which leads to the current workaround of * defining e.g. -0x80000000 as (-0x7fffffffL - 1L). See doc/code-issues.txt * for a longer discussion. * * Note: avoid typecasts and computations in macro integer constants as they * can then no longer be used in macro relational expressions (such as * #if DUK_SIZE_MAX < 0xffffffffUL). There is internal code which relies on * being able to compare DUK_SIZE_MAX against a limit.
label: code-design
38415. Not standard but common enough
38416. --- QNX ---
38417. **comment:** Object property allocation layout has implications for memory and code * footprint and generated code size/speed. The best layout also depends * on whether the platform has alignment requirements or benefits from * having mostly aligned accesses.
label: code-design
38418. (v)snprintf() is missing before MSVC 2015. Note that _(v)snprintf() does * NOT NUL terminate on truncation, but Duktape code never assumes that. * http://stackoverflow.com/questions/2915672/snprintf-and-visual-studio-2010
38419. Motorola 68K. Not defined by VBCC, so user must define one of these * manually when using VBCC.
38420. http://msdn.microsoft.com/en-us/library/aa235362(VS.60).aspx
38421. Non-C99 case, still relying on DUK_UINTPTR_MAX, as long as it is not a computed value
38422. http://bellard.org/tcc/tcc-doc.html#SEC7
38423. * Intermediate helper defines
38424. **comment:** Macro hackery to convert e.g. __LINE__ to a string without formatting, * see: http://stackoverflow.com/questions/240353/convert-a-preprocessor-token-to-a-string
label: code-design
38425. Flash player (e.g. Crossbridge)
38426. no endian.h
38427. Shared includes: C89
38428. integer byte order
38429. float word order
38430. * Convert DUK_USE_BYTEORDER, from whatever source, into currently used * internal defines. If detection failed, #error out.
38431. e.g. ptrdiff_t
38432. **comment:** XXX: DUK_NOINLINE, DUK_INLINE, DUK_ALWAYS_INLINE for msvc?
label: code-design
38433. SPARC
38434. VBCC supports C99 so check only for C99 for union initializer support. * Designated union initializers would possibly work even without a C99 check.
38435. C++11 apparently ratified stdint.h

38436. QNX gcc cross compiler seems to define e.g. __LITTLEENDIAN__ or __BIGENDIAN__: * \$ /opt/qnx650/host/linux/x86/usr/bin/i486-pc-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 67:#define __LITTLEENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/mips-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 81:#define __BIGENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/arm-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 70:#define __LITTLEENDIAN__ 1

38437. At least some uclibc versions have broken floating point math. For * example, fpclassify() can incorrectly classify certain NaN formats. * To be safe, use replacements.

38438. Placeholder fix for (detection is wider than necessary): * http://llvm.org/bugs/show_bug.cgi?id=17788

38439. snprintf() is technically not part of C89 but usually available.

38440. --- x32 ---

38441. Basic integer typedefs and limits, preferably from inttypes.h, otherwise * through automatic detection.

38442. **comment:** Only include when compiling Duktape to avoid polluting application build * with a lot of unnecessary defines.
label: code-design

38443. --- VBCC ---

38444. On some systems SIZE_MAX can be smaller than max unsigned 32-bit value * which seems incorrect if size_t is (at least) an unsigned 32-bit type. * However, it doesn't seem useful to error out compilation if this is the * case.

38445. **comment:** On other platforms use layout 2, which requires some padding but * is a bit more natural than layout 3 in ordering the entries. Layout * 3 is currently not used.
label: code-design

38446. VS2005+ should have variadic macros even when they're not C99.

38447. illumos / Solaris

38448. DUK_COMPILING_DUKTAPE

38449. DUK_OPT_FORCE_BYTEORDER

38450. QNX

38451. since gcc-4.5

38452. **comment:** Windows 32-bit and 64-bit are currently the same.
label: code-design

38453. This is optionally used by panic handling to cause the program to segfault * (instead of e.g. abort()) on panic. Valgrind will then indicate the C * call stack leading to the panic.

38454. IEEE float/double typedef.

38455. --- ARM 32-bit ---

38456. **comment:** XXX: This is technically not guaranteed because it's possible to configure * an x86 to require aligned accesses with Alignment Check (AC) flag.
label: code-design

38457. TinyC

38458. **comment:** Note: the funny looking computations for signed minimum 16-bit, 32-bit, and * 64-bit values are intentional as the obvious forms (e.g. -0x80000000L) are * -not- portable. See code-issues.txt for a detailed discussion.
label: code-design

38459. These are necessary wild guesses.

38460. Already provided above

38461. Atari ST TOS. __TOS__ defined by PureC. No platform define in VBCC * apparently, so to use with VBCC user must define __TOS__ manually.

38462. Byte order is big endian but cannot determine IEEE double word order.

38463. --- Flashplayer (Crossbridge) ---

38464. MIPS byte order varies so rely on autodetection.

38465. stdint.h not available

38466. **comment:** The most portable way to figure out local time offset is gmtime(), * but it's not thread safe so use with caution.
label: requirement

38467. Codepoint type. Must be 32 bits or more because it is used also for * internal codepoints. The type is signed because negative codepoints * are used as internal markers (e.g. to mark EOF or missing argument). * (X)UTF-8/CESU-8 encode/decode take and return an unsigned variant to * ensure duk_uint32_t casts back and forth nicely. Almost everything * else uses the signed one.

38468. DUK_USE_VARIADIC_MACROS: required from compilers, so no fill-in.

38469. TOS on M68K is always big endian.

38470. don't use strftime() for now

38471. **comment:** not sure, not needed with C99 anyway
label: requirement

38472. C++ doesn't have standard designated union initializers ({ .foo = 1 }).

38473. --- Emscripten ---

38474. nop

38475. DLL build detection

38476. **comment:** Avoid custom date parsing and formatting for portability.
label: code-design

38477. Workaround for older C++ compilers before including <inttypes.h>, * see e.g.: https://sourceware.org/bugzilla/show_bug.cgi?id=15366

38478. * Compiler autodetection

38479. --- SPARC 32-bit ---

38480. --- Clang ---

38481. On some platforms int is 16-bit but long is 32-bit (e.g. PureC)

38482. **comment:** * Byte order and double memory layout detection * * Endianness detection is a major portability hassle because the macros * and headers are not standardized. There's even variance across UNIX * platforms. Even with "standard" headers, details like underscore count * varies between platforms, e.g. both __BYTE_ORDER and _BYTE_ORDER are used * (Crossbridge has a single underscore, for instance). * * The checks below are structured with this in mind: several approaches are * used, and at the end we check if any of them worked. This allows generic * approaches to be tried first, and platform/compiler specific hacks tried * last. As a last resort, the user can force a specific endianness, as it's * not likely that automatic detection will work on the most exotic platforms. * * Duktape supports little and big endian machines. There's also support * for a hybrid used by some ARM machines where integers are little endian * but IEEE double values use a mixed order (12345678 -> 43218765). This * byte order for doubles is referred to as "mixed endian".
label: code-design

38483. **comment:** Check whether we should use 64-bit integers or not. * * Quite incomplete now. Use 64-bit types if detected (C99 or other detection) * unless they are known to be unreliable. For instance, 64-bit types are * available on VBCC but seem to misbehave.
label: code-design

38484. VS2013+ supports union initializers but there's a bug involving union-inside-struct: * <https://connect.microsoft.com/VisualStudio/feedback/details/805981> * The bug was fixed (at least) in VS2015 so check for VS2015 for now: * <https://blogs.msdn.microsoft.com/vcblog/2015/07/01/c-compiler-front-end-fixes-in-vs2015/> * Manually tested using VS2013, CL reports 18.00.31101, so enable for VS2013 too.

38485. no user declarations

38486. DUK_CONFIG_H_INCLUDED

38487. BCC (Bruce's C compiler): this is a "torture target" for compilation

38488. Generic Unix (includes Cygwin)

38489. There was a curious bug where test-bi-date-canceling.js would fail e.g. * on 64-bit Ubuntu, gcc-4.8.1, -m32, and no -std=c99. Some date computations * using doubles would be optimized which then broke some corner case tests. * The problem goes away by adding 'volatile' to the datetime computations. * Not sure what the actual triggering conditions are, but using this on * non-C99 systems solves the known issues and has relatively little cost * on other platforms.

38490. C99 / C++11 and above: rely on va_copy() which is required. * Omit parenthesis on macro right side on purpose to minimize differences * to direct use.

38491. C99 types

38492. DUK_SIZE_MAX (= SIZE_MAX) is often reliable

38493. --- PowerPC 64-bit ---
38494. See: /opt/qnx650/target/qnx6/usr/include/sys/platform.h
38495. --- SPARC 64-bit ---
38496. On platforms without any alignment issues, layout 1 is preferable * because it compiles to slightly less code and provides direct access * to property keys.
38497. Strict C99 case: DUK_UINTPTR_MAX (= UINTPTR_MAX) should be very reliable
38498. autodetect platform
38499. **comment:** NetBSD 6.0 x86 (at least) has a few problems with pow() semantics, * see test-bug-netbsd-math-pow.js. Use NetBSD specific workaround. * (This might be a wider problem; if so, generalize the define name.)
label: code-design
38500. autodetect architecture
38501. In VBCC (0.0 / 0.0) results in a warning and 0.0 instead of NaN. * In MSVC (VS2010 Express) (0.0 / 0.0) results in a compile error. * Use a computed NaN (initialized when a heap is created at the * latest).
38502. Clang: assume we have __va_copy() in non-C99 mode.
38503. --- Bruce's C compiler ---
38504. Same as 'duk_int_t' but guaranteed to be a 'fast' variant if this * distinction matters for the CPU. These types are used mainly in the * executor where it might really matter.
38505. vbcc + AmigaOS may be missing these
38506. C99 or C++11, no known issues
38507. **comment:** XXX: DUK_LIKELY, DUK_UNLIKELY for msvc?
label: code-design
38508. Initial fix: disable secure CRT related warnings when compiling Duktape * itself (must be defined before including Windows headers). Don't define * for user code including duktape.h.
38509. varargs
38510. * Fill-ins for platform, architecture, and compiler
38511. e.g. strftime
38512. **comment:** Note: PRS and FMT are intentionally left undefined for now. This means * there is no platform specific date parsing/formatting but there is still * the ISO 8601 standard format.
label: code-design
38513. vsnprintf() is technically not part of C89 but usually available.
38514. Rely on autodetection for byte order, alignment, and packed tval.
38515. AmigaOS + M68K seems to have math issues even when using GCC cross * compilation. Use replacements for all AmigaOS versions on M68K * regardless of compiler.
38516. Byte order is little endian but cannot determine IEEE double word order.
38517. C++
38518. MIPS. Related defines: __MIPSEB__, __MIPSEL__, __mips_isa_rev, __LP64__
38519. VBCC is missing the built-ins even in C99 mode (perhaps a header issue).
38520. __MSC_VER
38521. **comment:** Macro for suppressing warnings for potentially unreferenced variables. * The variables can be actually unreferenced or unreferenced in some * specific cases only; for instance, if a variable is only debug printed, * it is unreferenced when debug printing is disabled.
label: code-design
38522. Byte order varies, rely on autodetection.
38523. Use __setjmp() on Apple by default, see GH-55.
38524. TI-Nspire (using Ndless)
38525. --- PowerPC 32-bit ---
38526. Some math functions are C99 only. This is also an issue with some * embedded environments using uclibc where uclibc has been configured * not to provide some functions. For now, use replacements whenever * using uclibc.
38527. Most portable
38528. Boolean values are represented with the platform 'int'.
38529. Shared includes: stdint.h is C99
38530. BSD variant
38531. Old uclibcs have a broken memcpy so use memmove instead (this is overly wide * now on purpose): <http://lists.uclibc.org/pipermail/uclibc-cvs/2008-October/025511.html>
38532. date.h is omitted, and included per platform
38533. * Architecture autodetection
38534. The most portable current time provider is time(), but it only has a * one second resolution.
38535. Convenience, e.g. gcc 4.5.1 == 40501; <http://stackoverflow.com/questions/6031819/emulating-gccs-built-in-unreachable>
38536. defined(DUK_USE_BYTEORDER)
38537. In VBCC (1.0 / 0.0) results in a warning and 0.0 instead of infinity. * Use a computed infinity (initialized when a heap is created at the * latest).
38538. For now, hash part is dropped if and only if 16-bit object fields are used.
38539. MSVC dllexport/dllimport: appropriate __declspec depends on whether we're * compiling Duktape or the application.
38540. **comment:** On Windows, assume we're little endian. Even Itanium which has a * configurable endianness runs little endian in Windows.
label: code-design
38541. **comment:** Compiler specific hackery needed to force struct size to match alignment, * see e.g. duk_hbuffer.h. * * <http://stackoverflow.com/questions/11130109/c-struct-size-alignment> * <http://stackoverflow.com/questions/10951039/specifying-64-bit-alignment>
label: code-design
38542. Index values must have at least 32-bit signed range.
38543. **comment:** Workaround for GH-323: avoid inlining control when compiling from * multiple sources, as it causes compiler portability trouble.
label: code-design
38544. byte order
38545. Linux
38546. Based on 'make checkalign' there are no alignment requirements on * Linux SH4, but align by 4 is probably a good basic default.
38547. MSVC preprocessor defines: <http://msdn.microsoft.com/en-us/library/b0084kay.aspx> * __MSC_FULL_VER includes the build number, but it has at least two formats, see e.g. * BOOST_MSVC_FULL_VER in http://www.boost.org/doc/libs/1_52_0/boost/config/compiler/visualc.hpp
38548. ARM
38549. Fast variants of small integers, again for really fast paths like the * executor.
38550. **comment:** When C99 types are not available, we use heuristic detection to get * the basic 8, 16, 32, and (possibly) 64 bit types. The fast/least * types are then assumed to be exactly the same for now: these could * be improved per platform but C99 types are very often now available. * 64-bit types are not available on all platforms; this is OK at least * on 32-bit platforms. * * This detection code is necessarily a bit hacky and can provide typedefs * and defines that won't work correctly on some exotic platform.
label: code-design
38551. **comment:** parameter passing, not thread safe
label: requirement
38552. args go here in parens
38553. * Structs
38554. Note: combining __FILE__, __LINE__, and __func__ into fmt would be * possible compile time, but waste some space with shared function names.
38555. DUK_USE_VARIADIC_MACROS
38556. * Prototypes

38557. **comment:** * Debugging macros, DUK_DPRINT() and its variants in particular. * * DUK_DPRINT() allows formatted debug prints, and supports standard * and Duktape specific formatters. See duk_debug_vsnprintf.c for details. * * DUK_D(x), DUK_DD(x), and DUK_DDD(x) are used together with log macros * for technical reasons. They are concretely used to hide 'x' from the * compiler when the corresponding log level is disabled. This allows * clean builds on non-C99 compilers, at the cost of more verbose code. * Examples: * * DUK_D(DUK_DPRINT("foo")); * DUK_DD(DUK_DPRINT("foo")); * DUK_DDD(DUK_DDDPRINT("foo")); * * This approach is preferable to the old "double parentheses" hack because * double parentheses make the C99 solution worse: __FILE__ and __LINE__ can * no longer be added transparently without going through globals, which * works poorly with threading.
label: code-design

38558. DUK_DEBUG_H_INCLUDED

38559. omit

38560. **comment:** Without variadic macros resort to comma expression trickery to handle debug * prints. This generates a lot of harmless warnings. These hacks are not * needed normally because DUK_D() and friends will hide the entire debug log * statement from the compiler.
label: code-design

38561. args go here as a comma expression in parens

38562. * Exposed debug macros: debugging disabled

38563. DUK_USE_DEBUG

38564. args

38565. * Exposed debug macros: debugging enabled

38566. **comment:** unused
label: code-design

38567. (maybe) truncated

38568. normal

38569. * Fixed buffer helper useful for debugging, requires no allocation * which is critical for debugging.

38570. actual chars dropped (not just NUL term)

38571. DUK_USE_DEBUG

38572. error

38573. * Debug dumping of duk_heap.

38574. Note: there is no standard formatter for function pointers

38575. DUK_USE_DEBUG

38576. heap->strs: not worth dumping

38577. **comment:** unused
label: code-design

38578. **comment:** * Debugging macro calls.
label: code-design

38579. http://en.wikipedia.org/wiki/ANSI_escape_code

38580. DUK_USE_DPRINT_COLORS

38581. DUK_USE_VARIADIC_MACROS

38582. * Debugging disabled

38583. DUK_USE_DEBUG

38584. * Debugging enabled

38585. %lf, %ld etc

38586. encode \xffBar as _Bar if no quotes are applied, this is for * readable internal keys.

38587. XXX: print built-ins array?

38588. %ld

38589. terminal type: no depth check

38590. may be NULL

38591. XXX: use DUK_HSTRING_FLAG_INTERNAL?

38592. **comment:** * Custom formatter for debug printing, allowing Duktape specific data * structures (such as tagged values and heap objects) to be printed with * a nice format string. Because debug printing should not affect execution * state, formatting here must be independent of execution (see implications * below) and must not allocate memory. * * Custom format tags begin with a '%!' to safely distinguish them from * standard format tags. The following conversions are supported: * * %!T tagged value (duk_tval *) * %!O heap object (duk_heaphdr *) * %!I decoded bytecode instruction * %!C bytecode instruction opcode name (arg is long) * * Everything is serialized in a JSON-like manner. The default depth is one * level, internal prototype is not followed, and internal properties are not * serialized. The following modifiers change this behavior: * * @ print pointers * # print binary representations (where applicable) * d deep traversal of own properties (not prototype) * p follow prototype chain (useless without 'd') * i include internal properties (other than prototype) * x hexdump buffers * h heavy formatting * * For instance, the following serializes objects recursively, but does not * follow the prototype chain nor print internal properties: "%!do". * * Notes: * * * Standard snprintf return value semantics seem to vary. This * implementation returns the number of bytes it actually wrote * (excluding the null terminator). If retval == buffer size, * output was truncated (except for corner cases). * * * Output format is intentionally different from EcmaScript * formatting requirements, as formatting here serves debugging * of internals. * * * Depth checking (and updating) is done in each type printer * separately, to allow them to call each other freely. * * * Some pathological structures might take ages to print (e.g. * self recursion with 100 properties pointing to the object * itself). To guard against these, each printer also checks * whether the output buffer is full; if so, early exit. * * * Reference loops are detected using a loop stack.
label: code-design

38593. **comment:** XXX: option to fix opcode length so it lines up nicely
label: code-design

38594. %p

38595. after this, return paths should 'goto finished' for decrement

38596. %lx

38597. * Notation: double underscore used for internal properties which are not * stored in the property allocation (e.g. '__valstack').

38598. %f and %lf both consume a 'long'

38599. %x; only 16 bits are guaranteed

38600. maximum length of standard format tag that we support

38601. **comment:** loop_stack_index could be perhaps be replaced by 'depth', but it's nice * to not couple these two mechanisms unnecessarily.
label: code-design

38602. leave out trailing 'unused' elements

38603. depth check is done when printing an actual type

38604. IEEE double is approximately 16 decimal digits; print a couple extra

38605. ignore

38606. char format: use int

38607. maximum recursion depth for loop detection stacks

38608. unsupported: would consume multiple args

38609. currently implicitly also DUK_USE_DOUBLE_LINKED_HEAP

38610. DUK_USE_DEBUG

38611. Should not happen.

38612. helpers

38613. %s

38614. '%p' and NULL is portable, but special case it * anyway to get a standard NULL marker in logs.

38615. list of conversion specifiers that terminate a format tag; * this is unfortunately guesswork.

38616. **comment:** unused
label: code-design

38617. must match bytecode defines now; build autogenerate?

38618. %lu
38619. Note: string is a terminal heap object, so no depth check here
38620. two special escapes: '\ and "", other printables as is
38621. from curr pc
38622. **comment:** Formatting function pointers is tricky: there is no standard pointer for * function pointers and the size of a function pointer may depend on the * specific pointer type. This helper formats a function pointer based on * its memory layout to get something useful on most platforms.
 label: code-design
38623. %c', passed concretely as int
38624. **comment:** format is too large, abort
 label: code-design
38625. Quite approximate but should be useful for little and big endian.
38626. %d; only 16 bits are guaranteed
38627. %s and NULL is not portable, so special case * it for debug printing.
38628. **comment:** prototype should be last, for readability
 label: code-design
38629. fall through
38630. **comment:** dump all allocated entries, unused entries print as 'unused', * note that these may extend beyond current 'length' and look * a bit funny.
 label: code-design
38631. assume exactly 1 arg, which is why '*' is forbidden; arg size still * depends on type though.
38632. %u; only 16 bits are guaranteed
38633. heapobj recursion depth when deep printing is selected
38634. * Format tag parsing. Since we don't understand all the * possible format tags allowed, we just scan for a terminating * specifier and keep track of relevant modifiers that we do * understand. See man 3 printf.
38635. from next pc
38636. if property key begins with underscore, encode it with * forced quotes (e.g. "_Foo") to distinguish it from encoded * internal properties (e.g. \xffBar -> _Bar).
38637. **comment:** XXX: limit to quoted strings only, to save keys from being cluttered?
 label: code-design
38638. return total chars written excluding terminator
38639. own pointer
38640. throw_flag
38641. nargs == 2 so we can pass a callstack level to eval().
38642. Write unsigned 32-bit integer.
38643. Can happen if duk_throw() is called on an empty * callstack.
38644. PC/line semantics here are: * - For callstack top we're conceptually between two * opcodes and current PC indicates next line to * execute, so report that (matches Status). * - For other activations we're conceptually still * executing the instruction at PC-1, so report that * (matches error stacktrace behavior). * - See: <https://github.com/svaarala/duktape/issues/281>
38645. Use result value as is.
38646. * Debug connection peek and flush primitives
38647. Reply with tvals pushed by request callback
38648. Request callback should push values for reply to client onto valstack
38649. * DumpHeap command
38650. **comment:** NULL not needed here
 label: code-design
38651. **comment:** XXX: this should be optimized to be a raw query and avoid valstack * operations if possible.
 label: code-design
38652. push to stack
38653. Report next pc/line to be executed.
38654. DUK_USE_DEBUGGER_DUMPHEAP || DUK_USE_DEBUGGER_INSPECT
38655. tolerates NULL h_buf
38656. The actual detached_cb call is postponed to message loop so * we don't need any special precautions here (just skip to EOM * on the already closed connection).
38657. Avoid doing an actual write callback with length == 0, * because that's reserved for a write flush.
38658. As with all inspection code, we rely on the debug client providing * a valid, non-stale pointer: there's no portable way to safely * validate the pointer here.
38659. **comment:** unused
 label: code-design
38660. [...] func varmap enum key value this]
38661. [...] result]
38662. side effects
38663. * Duktape debugger
38664. * Halt execution helper
38665. **comment:** XXX: several nice-to-have improvements here: * - Use direct reads avoiding value stack operations * - Avoid triggering getters, indicate getter values to debug client * - If side effects are possible, add error catching
 label: code-design
38666. ptr may be NULL
38667. save stack top
38668. * NFY <int: 5> <int: fatal> <str: msg> <str: filename> <int: linenumber> EOM
38669. DUK_USE_DEBUGGER_INSPECT
38670. **comment:** Called on a read/write error: NULL all callbacks except the detached * callback so that we never accidentally call them after a read/write * error has been indicated. This is especially important for the transport * I/O callbacks to fulfill guaranteed callback semantics.
 label: code-design
38671. flags
38672. For strings, special form for short lengths.
38673. XXX: Decrementing and restoring act->curr_pc works now, but if the * debugger message loop gains the ability to adjust the current PC * (e.g. a forced jump) restoring the PC here will break. Another * approach would be to use a state flag for the "decrement 1 from * topmost activation's PC" and take it into account whenever dealing * with PC values.
38674. unsigned
38675. Read tvals from the message and push them onto the valstack, * then call the request callback to process the request.
38676. **comment:** XXX: comes out as signed now
 label: code-design
38677. Report thrown value to client coerced to string
38678. heap->dbg_udata: keep
38679. **comment:** Process debug messages until we are no longer paused.
 label: code-design
38680. Caller must trigger recomputation of active breakpoint list. To * ensure stale values are not used if that doesn't happen, clear the * active breakpoint list here.
38681. switch initial byte
38682. may be NULL
38683. Note: array dump will include elements beyond * 'length'.
38684. Halt execution and enter a debugger message loop until execution is resumed * by the client. PC for the current activation may be temporarily decremented * so that the "current" instruction will be shown by the client. This helper * is callable from anywhere, also outside bytecode executor.

38685. **comment:** SCANBUILD: warning about 'thr' potentially being NULL here, * warning is incorrect because thr != NULL always here.
label: code-design
38686. **comment:** XXX: differentiate null pointer from empty string?
label: code-design
38687. Native function pointer may be different from a void pointer, * and we serialize it from memory directly now (no byte swapping etc).
38688. To use the shared helper need the virtual index.
38689. Breakpoint entries above the used area are left as garbage.
38690. Skip fully.
38691. Skip dvalue.
38692. Note: peek cannot currently trigger a detach * so the dbg_detaching == 0 assert outside the * loop is correct.
38693. **comment:** prevent multiple in-progress detaches
label: requirement
38694. DUK_USE_DEBUGGER_THROW_NOTIFY
38695. processed one or more messages
38696. index
38697. no_block
38698. * Simple commands
38699. Safe to call multiple times.
38700. **comment:** Can be called multiple times with no harm. Mark the transport * bad (dbg_read_cb == NULL) and clear state except for the detached * callback and the udata field. The detached callback is delayed * to the message loop so that it can be called between messages; * this avoids corner cases related to immediate debugger reattach * inside the detached callback.
label: code-design
38701. The debugger protocol doesn't support a plain integer encoding for * the full 32-bit unsigned range (only 32-bit signed). For now, * unsigned 32-bit values simply written as signed ones. This is not * a concrete issue except for 32-bit heaphdr fields. Proper solutions * would be to (a) write such integers as IEEE doubles or (b) add an * unsigned 32-bit dvalue.
38702. tentative, checked later
38703. Write signed 32-bit integer.
38704. Peek ahead in the stream one byte.
38705. * Breakpoint management
38706. **comment:** Numbers are normalized to big (network) endian. We can * (but are not required) to use integer dvalues when there's * no loss of precision. * * XXX: share check with other code; this check is slow but * reliable and doesn't require careful exponent/mantissa * mask tricks as in the fastint downgrade code.
label: code-design
38707. Decrement PC if that was requested, this requires a PC sync.
38708. **comment:** For short strings, use a fixed temp buffer.
label: code-design
38709. restore stack top
38710. Pretend like we got EOM
38711. always push some string
38712. * Detach handling
38713. duk_hnativefunction specific fields.
38714. [...] eval "eval" eval_input level]
38715. Process one debug message. Automatically restore value stack top to its * entry value, so that individual message handlers don't need exact value * stack handling which is convenient.
38716. For now shared handler is fine.
38717. Signed integers always map to 4 bytes now.
38718. No debugger support.
38719. * Debug connection message write helpers
38720. terminator
38721. enum_flags
38722. this cannot initiate a detach
38723. true even with reattach
38724. XXX: Current putvar implementation doesn't have a success flag, * add one and send to debug client?
38725. keep heap->dbg_detached_cb
38726. num_stack_args
38727. **comment:** It is important not to call this if the last byte read was an EOM. * Reading ahead in this scenario would cause unnecessary blocking if * another message is not available.
label: code-design
38728. We're conceptually between two opcodes; act->pc indicates the next * instruction to be executed. This is usually the correct pc/line to * indicate in Status. (For the 'debugger' statement this now reports * the pc/line after the debugger statement because the debugger opcode * has already been executed.)
38729. Return 1: got EOM
38730. heap->dbg_processing: keep on purpose to avoid debugger re-entry in detaching state
38731. NOTE: length may be zero
38732. Read fully.
38733. No activation, no variable access. Could also pretend * we're in the global program context and read stuff off * the global object.
38734. get_value
38735. * Process incoming debug requests * * Individual request handlers can push temporaries on the value stack and * rely on duk_debug_process_message() to restore the value stack top * automatically.
38736. even after a detach and possible reattach
38737. avoid calling write callback in detach1()
38738. **comment:** XXX: Not needed for now, so not implemented. Note that * function pointers may have different size/layout than * a void pointer.
label: requirement
38739. **comment:** XXX: optimize to use direct reads, i.e. avoid * value stack operations.
label: code-design
38740. pruned
38741. switch cmd
38742. For anything other than an Error instance, we calculate the * error location directly from the current activation if one * exists.
38743. NOTE: act may be NULL if an error is thrown outside of any activation, * which may happen in the case of, e.g. syntax errors.
38744. Careful here: state must be wiped before the call * so that we can cleanly handle a re-attach from * inside the callback.
38745. **comment:** Ensure there are no stale active breakpoint pointers. * Breakpoint list is currently kept - we could empty it * here but we'd need to handle refcounts correctly, and * we'd need a 'thr' reference for that. * * XXX: clear breakpoint on either attach or detach?
label: code-design
38746. Short circuit if is safe: if act->curr_pc != NULL, 'fun' is * guaranteed to be a non-NULL EcmaScript function.
38747. **comment:** unused: not accepted in inbound messages
label: code-design
38748. Allow NULL 'msg'
38749. * Object inspection commands: GetHeapObjInfo, GetObjPropDesc, * GetObjPropDescRange
38750. detached
38751. XXX: Currently no inspection of threads, e.g. value stack, call * stack, catch stack, etc.

38752. * Debug connection skip primitives
38753. isAccessor
38754. * Helper structs
38755. At least: ... [err]
38756. Error instance, use augmented error data directly
38757. just in case callback is broken and won't write 'x'
38758. Accept any pointer-like value; for 'object' dvalue, read * and ignore the class number.
38759. DUK_USE_DEBUGGER_DUMPHEAP
38760. -> [... func varmap enum]
38761. duk_hobject specific fields.
38762. The index range space is conceptually the array part followed by the * entry part. Unlike normal enumeration all slots are exposed here as * is and return 'unused' if the slots are not in active use. In particular * the array part is included for the full a_size regardless of what the * array .length is.
38763. enum_index
38764. this is especially critical to avoid another write call in detach1()
38765. * Debug connection write primitives
38766. For errors a string coerced result is most informative * right now, as the debug client doesn't have the capability * to traverse the error object.
38767. As an initial implementation, write flush after every EOM (and the * version identifier). A better implementation would flush only when * Duktape is finished processing messages so that a flush only happens * after all outbound messages are finished on that occasion.
38768. Inspect a property using a virtual index into a conceptual property list * consisting of (1) all array part items from [0,a_size[(even when above * .length) and (2) all entry part items from [0,e_next[. Unused slots are * indicated using dvalue 'unused'.
38769. Use b[] to access the size of the union, which is strictly not * correct. Can't use fixed size unless there's feature detection * for pointer byte size.
38770. **comment:** XXX: this has some overlap with object inspection; remove this and make * DumpHeap return lists of heapptrs instead?
 label: code-design
38771. Detach is pending; can be triggered from outside the * debugger loop (e.g. Status notify write error) or by * previous message handling. Call detached callback * here, in a controlled state, to ensure a possible * reattach inside the detached_cb is handled correctly. * * Recheck for detach in a while loop: an immediate * reattach involves a call to duk_debugger_attach() * which writes a debugger handshake line immediately * inside the API call. If the transport write fails * for that handshake, we can immediately end up in a * "transport broken, detaching" case several times here. * Loop back until we're either cleanly attached or * fully detached. * * NOTE: Reset dbg_processing = 1 forcibly, in case we * re-attached; duk_debugger_attach() sets dbg_processing * to 0 at the moment.
38772. The eval code is executed within the lexical environment of a specified * activation. For now, use global object eval() function, with the eval * considered a 'direct call to eval'. * * Callstack level for debug commands only affects scope -- the callstack * as seen by, e.g. Duktape.act() will be the same regardless.
38773. not an error
38774. As an initial implementation, read flush after exiting the message * loop. If transport is broken, this is a no-op (with debug logs).
38775. **comment:** NOTE: This is a bit fragile. It's important to ensure that * duk_debug_process_messages() never throws an error or * act->curr_pc will never be reset.
 label: code-design
38776. thr->heap->dbg_detaching may be != 0 if a debugger write outside * the message loop caused a transport error and detach1() to run.
38777. DUK_USE_STRTAB_PROBE
38778. Process messages until we're no longer paused or we peek * and see there's nothing to read right now.
38779. may be set to 0 by duk_debugger_attach() inside callback
38780. Variant for writing duk_tvals so that any heap allocated values are * written out as tagged heap pointers.
38781. DUK_USE_DEBUGGER_SUPPORT
38782. Easy to get wrong, so assert for it.
38783. optional callstack level
38784. restore PC
38785. w/o refcounting
38786. Write fully.
38787. * Debug message processing
38788. * Debug connection read primitives
38789. compensate for eval() call
38790. Since we already started writing the reply, just emit nothing.
38791. DUK_USE_STRTAB_CHAIN
38792. Skip dvalues to EOM.
38793. heap->dbg_detached_cb: keep
38794. nargs
38795. 'this' binding shouldn't matter here
38796. * Status message and helpers
38797. Ensure dirty state causes a Status even if never process any * messages. This is expected by the bytecode executor when in * the running state.
38798. **comment:** Direct eval requires that there's a current * activation and it is an Ecmascript function. * When Eval is executed from e.g. cooperate API * call we'll need to do an indirect eval instead.
 label: code-design
38799. **comment:** XXX: exposed duk_debug_read_pointer
 label: code-design
38800. Error codes.
38801. **comment:** unused
 label: code-design
38802. Initial bytes for markers.
38803. Commands initiated by debug client.
38804. Commands and notifies initiated by Duktape.
38805. **comment:** XXX: exposed duk_debug_read_hbuffer
 label: code-design
38806. Debugger protocol version is defined in the public API header.
38807. The short string/integer initial bytes starting from 0x60 don't have * defines now.
38808. The low 8 bits map directly to duk_hobject.h DUK_PROPDESC_FLAG_xxx. * The remaining flags are specific to the debugger.
38809. DUK_DEBUGGER_H_INCLUDED
38810. **comment:** XXX: exposed duk_debug_read_buffer
 label: code-design
38811. Other initial bytes.
38812. **comment:** * Error handling macros, assertion macro, error codes. * * There are three level of 'errors': * * 1. Ordinary errors, relative to a thread, cause a longjmp, catchable. * 2. Fatal errors, relative to a heap, cause fatal handler to be called. * 3. Panic errors, unrelated to a heap and cause a process exit. * * Panics are used by the default fatal error handler and by debug code * such as assertions. By providing a proper fatal error handler, user * code can avoid panics in non-debug builds.
 label: code-design
38813. DUK_ERR_URI_ERROR: no macros needed
38814. this is added to checks to allow for Duktape * API calls in addition to function's own use
38815. assertion omitted
38816. DUK_USE_PANIC_HANDLER
38817. * Assertion helpers
38818. this variant is used when an assert would generate a compile warning by * being always true (e.g. >= 0 comparison for an unsigned value
38819. no refcount check
38820. * Fatal error * * There are no fatal error macros at the moment. There are so few call * sites that the fatal error handler is called directly.

38821. DUK_ERR_ERROR: no macros needed
38822. Verbose errors with key/value summaries (non-paranoid) or without key/value * summaries (paranoid, for some security sensitive environments), the paranoid * vs. non-paranoid distinction affects only a few specific errors.
38823. DUK_USE_ASSERTIONS
38824. **comment:** * Panic error * * Panic errors are not relative to either a heap or a thread, and cause * DUK_PANIC() macro to be invoked. Unless a user provides DUK_USE_PANIC_HANDLER, * DUK_PANIC() calls a helper which prints out the error and causes a process * exit. * * The user can override the macro to provide custom handling. A macro is * used to allow the user to have inline panic handling if desired (without * causing a potentially risky function call). * * Panics are only used in debug code such as assertions, and by the default * fatal error handler.
label: code-design
38825. * Prototypes
38826. nop
38827. already defined, good
38828. DUK_ERR_ASSERTION_ERROR: no macros needed
38829. **comment:** * Error throwing helpers * * The goal is to provide verbose and configurable error messages. Call * sites should be clean in source code and compile to a small footprint. * Small footprint is also useful for performance because small cold paths * reduce code cache pressure. Adding macros here only makes sense if there * are enough call sites to get concrete benefits.
label: code-design
38830. **comment:** Because there are quite many call sites, pack error code (require at most * 8-bit) into a single argument.
label: code-design
38831. * Assert macro: failure causes panic.
38832. DUK_ERROR_H_INCLUDED
38833. DUK_USE_VERBOSE_ERRORS
38834. DUK_ERR_REFERENCE_ERROR: no macros needed
38835. **comment:** * Error codes: defined in duktape.h * * Error codes are used as a shorthand to throw exceptions from inside * the implementation. The appropriate Ecmascript object is constructed * based on the code. Ecmascript code throws objects directly. The error * codes are defined in the public API header because they are also used * by calling code.
label: code-design
38836. **comment:** XXX: resolve macro definition issue or call through a helper function?
label: code-design
38837. Non-verbose errors for low memory targets: no file, line, or message.
38838. DUK_VERBOSE_ERRORS
38839. **comment:** the message should be a compile time constant without formatting (less risk); * we don't care about assertion text size because they're not used in production * builds.
label: code-design
38840. no valstack space check
38841. Assertion compatible inside a comma expression, evaluates to void. * Currently not compatible with DUK_USE_PANIC_HANDLER() which may have * a statement block.
38842. DUK_ERR_UNCAUGHT_ERROR: no macros needed
38843. assertion disabled
38844. DUK_USE_PARANOIJD_ERRORS
38845. DUK_ERR_EVAL: no macros needed
38846. **comment:** * Normal error * * Normal error is thrown with a longjmp() through the current setjmp() * catchpoint record in the duk_heap. The 'curr_thread' of the duk_heap * identifies the throwing thread. * * Error formatting is usually unnecessary. The error macros provide a * zero argument version (no formatting) and separate macros for small * argument counts. Variadic macros are not used to avoid portability * issues and avoid the need for stash-based workarounds when they're not * available. Vararg calls are avoided for non-formatted error calls * because vararg call sites are larger than normal, and there are a lot * of call sites with no formatting. * * Note that special formatting provided by debug macros is NOT available. * * The _RAW variants allow the caller to specify file and line. This makes * it easier to write checked calls which want to use the call site of the * checked function, not the error macro call inside the checked function.
label: code-design
38847. * Helper for valstack space * * Caller of DUK_ASSERT_VALSTACK_SPACE() estimates the number of free stack entries * required for its own use, and any child calls which are not (a) Duktape API calls * or (b) Duktape calls which involve extending the valstack (e.g. getter call).
38848. **comment:** * Augment an error at creation time with _Tracedata/fileName/lineNumber * and allow a user error handler (if defined) to process/replace the error. * The error to be augmented is at the stack top. * * thr: thread containing the error value * thr_callstack: thread which should be used for generating callstack etc. * c_filename: C __FILE__ related to the error * c_line: C __LINE__ related to the error * noblame_fileline: if true, don't fileName/line as error source, otherwise use traceback * (needed because user code filename/line are reported but internal ones * are not) * * XXX: rename noblame_fileline to flags field; combine it to some existing * field (there are only a few call sites so this may not be worth it).
label: code-design
38849. unsigned
38850. **comment:** * Augmenting errors at their creation site and their throw site. * * When errors are created, traceback data is added by built-in code * and a user error handler (if defined) can process or replace the * error. Similarly, when errors are thrown, a user error handler * (if defined) can process or replace the error. * * Augmentation and other processing at error creation time is nice * because an error is only created once, but it may be thrown and * rethrown multiple times. User error handler registered for processing * an error at its throw site must be careful to handle rethrowing in * a useful manner. * * Error augmentation may throw an internal error (e.g. alloc error). * * Ecmascript allows throwing any values, so all values cannot be * augmented. Currently, the built-in augmentation at error creation * only augments error values which are Error instances (= have the * built-in Error.prototype in their prototype chain) and are also * extensible. User error handlers have no limitations in this respect.
label: code-design
38851. [...] errval]
38852. 3 entries actually needed below
38853. no need to check now: both success and error are OK
38854. **comment:** XXX: optimize: allocate an array part to the necessary size (upwards * estimate) and fill in the values directly into the array part; finally * update 'length'.
label: code-design
38855. assume PC is at most 32 bits and non-negative
38856. No activation matches, use undefined for both .fileName and * .lineNumber (matches what we do with a _Tracedata based * no-match lookup).
38857. Append a "(line NNN)" to the "message" property of any error * thrown during compilation. Usually compilation errors are * SyntaxErrors but they can also be out-of-memory errors and * the like.
38858. DUK_USE_ERRTHROW
38859. num args
38860. If tracebacks are enabled, the '_Tracedata' property is the only * thing we need: 'fileName' and 'lineNumber' are virtual properties * which use '_Tracedata'.
38861. Lightfunc, not blamed now.
38862. * If tracebacks are disabled, 'fileName' and 'lineNumber' are added * as plain own properties. Since Error.prototype has accessors of * the same name, we need to define own properties directly (cannot * just use e.g. duk_put_prop_stridx). Existing properties are not * overwritten in case they already exist.
38863. -> [...] ... error]
38864. the argument only used for thr->heap, so specific thread doesn't matter
38865. traceback depth doesn't take into account the filename/line * special handling above (intentional)
38866. -> [...] ... errhandler errval]
38867. DUK_USE_ERRTHROW || DUK_USE_ERRCREATE
38868. * DUK_CALL_FLAG_IGNORE_RECLIMIT causes duk_handle_call() to ignore C * recursion depth limit (and won't increase it either). This is * dangerous, but useful because it allows the error handler to run * even if the original error is caused by C recursion depth limit. * * The heap level

DUK_HEAP_FLAG_ERRHANDLER_RUNNING is set for the * duration of the error handler and cleared afterwards. This flag * prevents the error handler from running recursively. The flag is * heap level so that the flag properly controls even coroutines * launched by an error handler. Since the flag is heap level, it is * critical to restore it correctly. * * We ignore errors now: a success return and an error value both * replace the original error value. (This would be easy to change.)

38869. Finally, blame the innermost callstack entry which has a * .fileName property.

38870. [... error]

38871. ignore reclimit, not constructor

38872. Native function, no relevant lineNumber.

38873. !DUK_USE_TRACEBACKS

38874. Without tracebacks the concrete .fileName and .lineNumber need * to be added directly.

38875. PC points to next instruction, find offending PC, * PC == 0 for native code.

38876. [... error func]

38877. ignore_loop

38878. callstack limits

38879. When creating built-ins, some of the built-ins may not be set * and we want to tolerate that when throwing errors.

38880. * The traceback format is pretty arcane in an attempt to keep it compact * and cheap to create. It may change arbitrarily from version to version. * It should be decoded/accessible through version specific accessors only. * * See doc/error-objects.rst

38881. [... error lineNumber fileName]

38882. * Add line number to a compiler error.

38883. DUK_USE_AUGMENT_ERROR_CREATE

38884. * Add .fileName and .lineNumber to an error on the stack top.

38885. * Criteria for augmenting: * * - augmentation enabled in build (naturally) * - error value internal prototype chain contains the built-in * Error prototype object (i.e. 'val instanceof Error') * * Additional criteria for built-in augmenting: * * - error value is an extensible object

38886. since no recursive error handler calls

38887. DUK_USE_PC2LINE

38888. **comment:** * Check whether or not we have an error handler. * * We must be careful of not triggering an error when looking up the * property. For instance, if the property is a getter, we don't want * to call it, only plain values are allowed. The value, if it exists, * is not checked. If the value is not a function, a TypeError happens * when it is called and that error replaces the original one.

label: code-design

38889. invalidated by pushes, so get out of the way

38890. Compiler SyntaxError (or other error) gets the primary blame. * Currently no flag to prevent blaming.

38891. [... errhandler undefined errval]

38892. [... errval errhandler]

38893. -> [... arr num]

38894. [... error func fileName]

38895. DUK_USE_TRACEBACKS

38896. If the value has a prototype loop, it's critical not to * throw here. Instead, assume the value is not to be * augmented.

38897. DUK_USE_AUGMENT_ERROR_THROW

38898. [... error arr]

38899. [... error func fileName lineNumber]

38900. call_flags

38901. -> [... errhandler undefined(= this) errval]

38902. Compiler SyntaxErrors (and other errors) come first, and are * blamed by default (not flagged "noblame").

38903. [... arr]

38904. Add function object.

38905. **comment:** * Helper for calling a user error handler. * * 'thr' must be the currently active thread; the error handler is called * in its context. The valstack of 'thr' must have the error value on * top, and will be replaced by another error value based on the return * value of the error handler. * * The helper calls duk_handle_call() recursively in protected mode. * Before that call happens, no longjmps should happen; as a consequence, * we must assume that the valstack contains enough temporary space for * arguments and such. * * While the error handler runs, any errors thrown will not trigger a * recursive error handler call (this is implemented using a heap level * flag which will "follow" through any coroutines resumed inside the * error handler). If the error handler is not callable or throws an * error, the resulting error replaces the original error (for Duktape * internal errors, duk_error_throw.c further substitutes this error with * a DoubleError which is not ideal). This would be easy to change and * even signal to the caller. * * The user error handler is stored in 'Duktape.errCreate' or * 'Duktape.errThrow' depending on whether we're augmenting the error at * creation or throw time. There are several alternatives to this approach, * see doc/error-objects.rst for discussion. * * Note: since further longjmp(s) may occur while calling the error handler * (for many reasons, e.g. a labeled 'break' inside the handler), the * caller can make no assumptions on the thr->heap->lj state after the * call (this affects especially duk_error_throw.c). This is not an issue * as long as the caller writes to the lj state only after the error handler * finishes.

label: code-design

38906. * Add ._Tracedata to an error on the stack top.

38907. XXX: specify array size, as we know it

38908. (flags<<32) + (line), flags = 0

38909. * Augment an error being created using Duktape specific properties * like ._Tracedata or .fileName/.lineNumber.

38910. * Augment an error at throw time; allow a user error handler (if defined) * to process/replace the error. The error to be augmented is at the * stack top.

38911. unreferenced w/o tracebacks

38912. Filename/line from C macros (_FILE_, _LINE_) are added as an * entry with a special format: (string, number). The number contains * the line and flags.

38913. **comment:** * Note: each API operation potentially resizes the callstack, * so be careful to re-lookup after every operation. Currently * these is no issue because we don't store a temporary 'act' * pointer at all. (This would be a non-issue if we operated * directly on the array part.)

label: code-design

38914. XXX: set with duk_object_set_length() when tracedata is filled directly

38915. unreferenced w/o asserts

38916. **comment:** XXX: using duk_put_prop_index() would cause obscure error cases when Array.prototype * has write-protected array index named properties. This was seen as DoubleErrors * in e.g. some test262 test cases. Using duk_xdef_prop_index() is better but heavier. * The best fix is to fill in the tracedata directly into the array part. There are * no side effect concerns if the array part is allocated directly and only INCREFs * happen after that.

label: code-design

38917. Add a number containing: pc, activation flags. * * PC points to next instruction, find offending PC. Note that * PC == 0 for native code.

38918. C call site gets blamed next, unless flagged not to do so. * XXX: file/line is disabled in minimal builds, so disable this * too when appropriate.

38919. **comment:** If we don't have a jmpbuf_ptr, there is little we can do * except panic. The caller's expectation is that we never * return. * * With C++ exceptions we now just propagate an uncaught error * instead of invoking the fatal error handler. Because there's * a dummy jmpbuf for C++ exceptions now, this could be changed.

label: code-design

38920. DUK_USE_CPP_EXCEPTIONS

38921. dummy

38922. * Do a longjmp call, calling the fatal error handler if no * catchpoint exists.

38923. * Error throwing helpers

38924. The file/line arguments are NULL and 0, they're ignored by DUK_ERROR_RAW() * when non-verbose errors are used.

38925. DUK_USE_VERBOSE_ERRORS

38926. **comment:** size for formatting buffers

label: code-design

38927. * Default fatal error handler

38928. !DUK_USE_PANIC_HANDLER

38929. omit print

38930. **comment:** dead code, but ensures portability (see Linux man page notes)
label: code-design

38931. SCANBUILD: "Dereference of null pointer", normal

38932. * Default panic handler

38933. * Error, fatal, and panic handling.

38934. exit() afterwards to satisfy "noreturn"

38935. DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT

38936. Don't intercept a DoubleError, we may have caused the initial double * fault and attempting to intercept it will cause us to be called * recursively and exhaust the C stack.

38937. Report it to the debug client

38938. * Helper to walk the thread chain and see if there is an active error * catcher. Protected calls or finally blocks aren't considered catching.

38939. * Get prototype object for an integer error code.

38940. **comment:** XXX: more specific error classes?
label: code-design

38941. DUK_USE_DEBUGGER_SUPPORT

38942. side effects

38943. XXX: Allow customizing the pause and notify behavior at runtime * using debugger runtime flags. For now the behavior is fixed using * config options.

38944. all we need to know

38945. **comment:** * XXX: As noted above, a protected API call won't be counted as a * catcher. This is usually convenient, e.g. in the case of a top- * level duk_pcall(), but may not always be desirable. Perhaps add an * argument to treat them as catchers?
label: code-design

38946. * Error helpers

38947. * Exposed helper for setting up heap longjmp state.

38948. use_prev_pc

38949. DUK_USE_DEBUGGER_SUPPORT && (DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT)

38950. If something is thrown with the debugger attached and nobody will * catch it, execution is paused before the longjmp, turning over * control to the debug client. This allows local state to be examined * before the stack is unwound. Errors are not intercepted when debug * message loop is active (e.g. for Eval).

38951. **comment:** * The __FILE__ and __LINE__ information is intentionally not used in the * creation of the error object, as it isn't useful in the tracedata. The * tracedata still contains the function which returned the negative return * code, and having the file/line of this function isn't very useful.
label: code-design

38952. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.

38953. **comment:** * Create and throw an error (originating from Duktape internally) * * Push an error object on top of the stack, possibly throw augmenting * the error, and finally longjmp. * * If an error occurs while we're dealing with the current error, we might * enter an infinite recursion loop. This is prevented by detecting a * "double fault" through the heap->handling_error flag; the recursion * then stops at the second level.
label: code-design

38954. * Create and throw an Ecmascript error object based on a code and a message. * * Used when we throw errors internally. Ecmascript generated error objects * are created by Ecmascript code, and the throwing is handled by the bytecode * executor.

38955. **comment:** XXX: if attempt to push beyond allocated valstack, this double fault * handling fails miserably. We should really write the double error * directly to thr->heap->lj.value1 and avoid valstack use entirely.
label: code-design

38956. * Create and push an error object onto the top of stack. * If a "double error" occurs, use a fixed error instance * to avoid further trouble.

38957. **comment:** XXX: unnecessary '%s' formatting here, but cannot use * 'msg' as a format string directly.
label: code-design

38958. reset callstack limit

38959. **comment:** XXX: this generates quite large code - perhaps select the error * class based on the code and then just use the error 'name'?
label: code-design

38960. just making sure

38961. * Finally, longjmp

38962. **comment:** Allow headroom for calls during error handling (see GH-191). * We allow space for 10 additional recursions, with one extra * for, e.g. a print() call at the deepest level.
label: code-design

38963. **comment:** * Augment error (throw time), unless alloc/double error
label: code-design

38964. **comment:** XXX: shared strings
label: code-design

38965. Error object is augmented at its creation here.

38966. * Helper for C function call negative return values.

38967. * Exception for Duktape internal throws when C++ exceptions are used * for long control transfers. * * Doesn't inherit from any exception base class to minimize the chance * that user code would accidentally catch this exception.

38968. intentionally empty

38969. DUK_EXCEPTION_H_INCLUDED

38970. duk_tval intentionally skipped

38971. * Forward declarations

38972. * Forward declarations for all Duktape structures.

38973. DUK_FORWDECL_H_INCLUDED

38974. no typedef

38975. Current size (not counting a dynamic buffer's "spare").

38976. External buffer with 'curr_alloc' managed by user code and pointing to an * arbitrary address. When heap pointer compression is not used, this struct * has the same layout as duk_hbuffer_dynamic.

38977. No pointer compression because pointer is potentially outside of * Duktape heap.

38978. **comment:** It's not strictly necessary to track the current size, but * it is useful for writing robust native code.
label: code-design

38979. Stored in duk_heaphdr h_extra16.

38980. * Flags * * Fixed buffer: 0 * Dynamic buffer: DUK_HBUFFER_FLAG_DYNAMIC * External buffer: DUK_HBUFFER_FLAG_DYNAMIC | DUK_HBUFFER_FLAG_EXTERNAL

38981. * Data following the header depends on the DUK_HBUFFER_FLAG_DYNAMIC * flag. * * If the flag is clear (the buffer is a fixed size one), the buffer * data follows the header directly, consisting of 'size' bytes. * * If the flag is set, the actual buffer is allocated separately, and * a few control fields follow the header. Specifically: * * - a "void *" pointing to the current allocation * - a duk_size_t indicating the full allocated size (always >= 'size') * * If DUK_HBUFFER_FLAG_EXTERNAL is set, the buffer has been allocated * by user code, so that Duktape won't be able to resize it and won't * free it. This allows buffers to point to e.g. an externally * allocated structure such as a frame buffer. * * Unlike strings, no terminator byte (NUL) is guaranteed after the * data. This would be convenient, but would pad aligned user buffers * unnecessarily upwards in size. For instance, if user code requested * a 64-byte dynamic buffer, 65 bytes would actually be allocated which * would then potentially round upwards to perhaps 68 or 72 bytes.

38982. **comment:** * Data follows the struct header. The struct size is padded by the * compiler based on the struct members. This guarantees that the * buffer data will be aligned-by-4 but not necessarily aligned-by-8. * * On platforms where alignment does not matter, the struct padding * could be removed (if there is any). On platforms where alignment * by 8 is required, the struct size must be forced to be a multiple * of 8 by some means. Without it, some user code may break, and also * Duktape itself breaks (e.g. the compiler stores duk_tvals in a * dynamic buffer).
label: code-design

38983. size stored in duk_heapheader unused flag bits
38984. * Misc defines
38985. indirect allocs
38986. may be NULL if alloc_size == 0
38987. * Prototypes
38988. **comment:** * Allocation size for 'curr_alloc' is alloc_size. There is no * automatic NUL terminator for buffers (see above for rationale). * * 'curr_alloc' is explicitly allocated with heap allocation * primitives and will thus always have alignment suitable for * e.g. duk_tval and an IEEE double.
label: code-design
38989. Dynamic buffer with 'curr_alloc' pointing to a dynamic area allocated using * heap allocation primitives. Also used for external buffers when low memory * options are not used.
38990. * Heap buffer representation. * * Heap allocated user data buffer which is either: * * 1. A fixed size buffer (data follows header statically) * 2. A dynamic size buffer (data pointer follows header) * * The data pointer for a variable size buffer of zero size may be NULL.
38991. Get/set the current user visible size, without accounting for a dynamic * buffer's "spare" (= usable size).
38992. Intentionally not 0x7fffffffUL; at least JSON code expects that * 2*len + 2 fits in 32 bits.
38993. buffer pointer is to an externally allocated buffer
38994. assume 0 <=> NULL
38995. dynamic buffer ops
38996. DUK_HBUFFER_H_INCLUDED
38997. **comment:** A union is used here as a portable struct size / alignment trick: * by adding a 32-bit or a 64-bit (unused) union member, the size of * the struct is effectively forced to be a multiple of 4 or 8 bytes * (respectively) without increasing the size of the struct unless * necessary.
label: code-design
38998. no extra padding
38999. * Field access
39000. Cannot be compressed as a heap pointer because may point to * an arbitrary address.
39001. Without heap pointer compression duk_hbuffer_dynamic and duk_hbuffer_external * have the same layout so checking for fixed vs. dynamic (or external) is enough.
39002. * Structs
39003. Get a pointer to the current buffer contents (matching current allocation * size). May be NULL for zero size dynamic/external buffer.
39004. Stored in duk_heapheader unused flags.
39005. Shared prefix for all buffer types.
39006. buffer is behind a pointer, dynamic or external
39007. Fixed buffer; data follows struct, with proper alignment guaranteed by * struct size.
39008. Impose a maximum buffer length for now. Restricted artificially to * ensure resize computations or adding a heap header length won't * overflow size_t and that a signed duk_int_t can hold a buffer * length. The limit should be synchronized with DUK_HSTRING_MAX_BYTelen.
39009. Because size > 0, NULL check is correct
39010. **comment:** Size sanity check. Should not be necessary because caller is * required to check this, but we don't want to cause a segfault * if the size wraps either in duk_size_t computation or when * storing the size in a 16-bit field.
label: code-design
39011. alloc external with size zero
39012. the compressed pointer is zeroed which maps to NULL, so nothing to do.
39013. For indirect allocs.
39014. no need to write 'out_bufdata'
39015. zero everything unless requested not to do so
39016. * duk_hbuffer allocation and freeing.
39017. Allocate a new duk_hbuffer of a certain type and return a pointer to it * (NULL on error). Write buffer data pointer to 'out_bufdata' (only if * allocation successful).
39018. no wrapping
39019. **comment:** * duk_hbuffer operations such as resizing and inserting/appending data to * a dynamic buffer. * * Append operations append to the end of the buffer and they are relatively * efficient: the buffer is grown with a "spare" part relative to the buffer * size to minimize reallocations. Insert operations need to move existing * data forward in the buffer with memmove() and are not very efficient. * They are used e.g. by the regexp compiler to "backpatch" regexp bytecode.
label: code-design
39020. * Resizing
39021. 'res' may be NULL if new allocation size is 0.
39022. **comment:** * Note: use indirect realloc variant just in case mark-and-sweep * (finalizers) might resize this same buffer during garbage * collection.
label: code-design
39023. * Maximum size check
39024. * The entire allocated buffer area, regardless of actual used * size, is kept zeroed in resizes for simplicity. If the buffer * is grown, zero the new part.
39025. All element accessors are host endian now (driven by TypedArray spec).
39026. Validate that the whole slice [0,length] is contained in the underlying * buffer. Caller must ensure 'buf' != NULL.
39027. DUK_HBUFFEROBJECT_H_INCLUDED
39028. byte index limit for element access, exclusive
39029. **comment:** Slice and accessor information. * * Because the underlying buffer may be dynamic, these may be * invalidated by the buffer being modified so that both offset * and length should be validated before every access. Behavior * when the underlying buffer has changed doesn't need to be clean: * virtual 'length' doesn't need to be affected, reads can return * zero/NaN, and writes can be ignored. * * Note that a data pointer cannot be precomputed because 'buf' may * be dynamic and its pointer unstable.
label: code-design
39030. Validate byte read/write for virtual 'offset', i.e. check that the * offset, taking into account h->offset, is within the underlying * buffer size. This is a safety check which is needed to ensure * that even a misconfigured duk_hbufferobject never causes memory * unsafe behavior (e.g. if an underlying dynamic buffer changes * after being setup). Caller must ensure 'buf' != NULL.
39031. Shared object part.
39032. element type
39033. Underlying buffer (refcounted), may be NULL.
39034. Get the current data pointer (caller must ensure buf != NULL) as a * duk_uint8_t ptr.
39035. True if slice is full, i.e. offset is zero and length covers the entire * buffer. This status may change independently of the duk_hbufferobject if * the underlying buffer is dynamic and changes without the hbufferobject * being changed.
39036. element size shift: * 0 = u8/i8 * 1 = u16/i16 * 2 = u32/i32/float * 3 = double
39037. No assertions for offset or length; in particular, \ * it's OK for length to be longer than underlying \ * buffer. Just ensure they don't wrap when added. \
39038. byte offset to buf
39039. * Heap Buffer object representation. Used for all Buffer variants.
39040. Clamp an input byte length (already assumed to be within the nominal * duk_hbufferobject 'length') to the current dynamic buffer limits to * yield a byte length limit that's safe for memory accesses. This value * can be invalidated by any side effect because it may trigger a user * callback that resizes the underlying buffer.
39041. Slice starting point is beyond current length.
39042. DUK_USE_BUFFEROBJECT_SUPPORT
39043. Line number range for function. Needed during debugging to * determine active breakpoints.
39044. number of arguments allocated to regs
39045. * Additional control information is placed into the object itself * as internal properties to avoid unnecessary fields for the * majority of functions. The compiler tries to omit internal * control fields when possible. * * Function templates: * * { * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * }

_Pc2line: <debug info for pc-to-line mapping>, * } ** Function instances: * * { * length: 2, * prototype: { constructor: <func> }, * caller: <thrower>, * arguments: <thrower>, * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * _Varenv: <variable environment of closure>, * _Lexenv: <lexical environment of closure (if differs from _Varenv)> * } ** More detailed description of these properties can be found * in the documentation.

39046. **comment:** XXX: casts could be improved, especially for GET/SET DATA

label: code-design

39047. * Heap compiled function (Ecmascript function) representation. * * There is a single data buffer containing the Ecmascript function's * bytecode, constants, and inner functions.

39048. shared object part

39049. **comment:** * 'nregs' registers are allocated on function entry, at most 'nargs' * are initialized to arguments, and the rest to undefined. Arguments * above 'nregs' are not mapped to registers. All registers in the * active stack range must be initialized because they are GC reachable. * 'nargs' is needed so that if the function is given more than 'nargs' * arguments, the additional arguments do not 'clobber' registers * beyond 'nregs' which must be consistently initialized to undefined. * * Usually there is no need to know which registers are mapped to * local variables. Registers may be allocated to variable in any * way (even including gaps). However, a register-variable mapping * must be the same for the duration of the function execution and * the register cannot be used for anything else. * * When looking up variables by name, the '_Varmap' map is used. * When an activation closes, registers mapped to arguments are * copied into the environment record based on the same map. The * reverse map (from register to variable) is not currently needed * at run time, except for debugging, so it is not maintained.

label: code-design

39050. * Field accessor macros

39051. DUK_HCOMPILEDFUNCTION_H_INCLUDED

39052. reg to allocate

39053. **comment:** No need for constants pointer (= same as data). * * When using 16-bit packing alignment to 4 is nice. 'funcs' will be * 4-byte aligned because 'constants' are duk_tvals. For now the * inner function pointers are not compressed, so that 'bytecode' will * also be 4-byte aligned.

label: code-design

39054. Data area, fixed allocation, stable data ptrs.

39055. Note: assumes 'data' is always a fixed buffer

39056. * Main struct

39057. * Pointers to function data area for faster access. Function * data is a buffer shared between all closures of the same * "template" function. The data buffer is always fixed (non- * dynamic, hence stable), with a layout as follows: * * constants (duk_tval) * inner functions (duk_hobject *) * bytecode (duk_instr_t) * * Note: bytecode end address can be computed from 'data' buffer * size. It is not strictly necessary functionally, assuming * bytecode never jumps outside its allocated area. However, * it's a safety/robustness feature for avoiding the chance of * executing random data as bytecode due to a compiler error. * * Note: values in the data buffer must be incref'd (they will * be decref'd on release) for every compiledfunction referring * to the 'data' element.

39058. * Accessor macros for function specific data areas

39059. **comment:** XXX: double evaluation of DUK_HCOMPILEDFUNCTION_GET_DATA()

label: code-design

39060. callback for indirect reallocs, request for current pointer

39061. Retry allocation after mark-and-sweep for this * many times. A single mark-and-sweep round is * not guaranteed to free all unreferenced memory * because of finalization (in fact, ANY number of * rounds is strictly not enough).

39062. **comment:** XXX: static alloc is OK until separate chaining stringtable * resizing is implemented.

label: code-design

39063. heap destruction ongoing, finalizer rescue no longer possible

39064. resend state next time executor is about to run

39065. No field needed when strings are in ROM.

39066. indicates a deleted string; any fixed non-NULL, non-hstring pointer works

39067. **comment:** alloc size in elements

label: code-design

39068. value1 -> label number, pseudo-type to indicate a break continuation (for ENDFIN)

39069. don't run finalizers; queue finalizable objects back to heap_allocated

39070. **comment:** unused

label: code-design

39071. DUK_USE_ROM_STRINGS

39072. don't compact objects; needed during object property allocation resize

39073. borrowed; NULL if no step state (NULLed in unwind)

39074. breakpoints: [0,breakpoint_count[gc reachable

39075. value1 -> resume value, value2 -> resume thread, iserror -> error/normal

39076. * Other heap related defines

39077. Allocator functions.

39078. don't run finalizers; leave finalizable objects in finalize_list for next round

39079. Step types

39080. used entries + approx 100% -> reset load to 50%

39081. current thread

39082. * Mark-and-sweep flags * * These are separate from heap level flags now but could be merged. * The heap structure only contains a 'base mark-and-sweep flags' * field and the GC caller can impose further flags.

39083. * Memory constants

39084. force executor restart to recheck breakpoints; used to handle function returns (see GH-303)

39085. required, NULL implies detached

39086. no value, pseudo-type to indicate a normal continuation (for ENDFIN)

39087. heap thread, used internally and for finalization

39088. * Thread switching * * To switch heap->curr_thread, use the macro below so that interrupt counters * get updated correctly. The macro allows a NULL target thread because that * happens e.g. in call handling.

39089. * Longjmp types, also double as identifying continuation type for a rethrow (in 'finally')

39090. * Longjmp state, contains the information needed to perform a longjmp. * Longjmp related values are written to value1, value2, and iserror.

39091. **comment:** * String cache should ideally be at duk_hthread level, but that would * cause string finalization to slow down relative to the number of * threads; string finalization must check the string cache for "weak" * references to the string being finalized to avoid dead pointers. * * Thus, string caches are now at the heap level now.

label: code-design

39092. Stringcache is used for speeding up char-offset-to-byte-offset * translations for non-ASCII strings.

39093. For manual debugging: instruction count based on executor and * interrupt counter book-keeping. Inspect debug logs to see how * they match up.

39094. Heap udata, used for allocator functions but also for other heap * level callbacks like pointer compression, etc.

39095. load factor min 25%

39096. helper to insert a (non-string) heap object into heap allocated list

39097. initial stringtable size, must be prime and higher than DUK_UTIL_MIN_HASH_PRIME

39098. used elements (includes DELETED)

39099. marker for detecting internal "double faults", see duk_error_throw.c

39100. callbacks and udata; dbg_read_cb != NULL is used to indicate attached state

39101. debugger state, only relevant when attached

39102. allocated heap objects

39103. 4'294'967'295

39104. longjmp type
39105. Mark-and-sweep interval is relative to combined count of objects and * strings kept in the heap during the latest mark-and-sweep pass. * Fixed point .8 multiplier and .0 adder. Trigger count (interval) is * decreased by each (re)allocation attempt (regardless of size), and each * refzero processed object. ** 'SKIP' indicates how many (re)allocations to wait until a retry if * GC is skipped because there is no thread do it with yet (happens * only during init phases).
39106. Precomputed pointers when using 16-bit heap pointer packing.
39107. Maximum number of breakpoints. Only breakpoints that are set are * consulted so increasing this has no performance impact.
39108. Milliseconds between status notify and transport peeks.
39109. value1 -> error object
39110. mark-and-sweep is currently running
39111. step type: none, step into, step over, step out
39112. refcount code is processing refzero list
39113. emergency mode: try extra hard
39114. time when status/peek was last done (Date-based rate limit)
39115. Opcode interval for a Date-based status/peek rate limit check. Only * relevant when debugger is attached. Requesting a timestamp may be a * slow operation on some platforms so this shouldn't be too low. On the * other hand a high value makes Duktape react to a pause request slowly.
39116. mark-and-sweep control
39117. if 0, 'str16' used, if > 0, 'strlist16' used
39118. * Stringtable entry for fixed size stringtable
39119. no refcounting
39120. currently processing messages or breakpoints: don't enter message processing recursively (e.g. no breakpoints when processing debugger eval)
39121. probe sequence (open addressing)
39122. * Heap structure. ** Heap contains allocated heap objects, interned strings, and built-in * strings for one or more threads.
39123. starting line number
39124. Fatal error handling, called e.g. when a longjmp() is needed but * lj.jmpbuf_ptr is NULL. fatal_func must never return; it's not * declared as "noreturn" because doing that for typedefs is a bit * challenging portability-wise.
39125. if 0, 'str' used, if > 0, 'strlist' used
39126. heap level "stash" object (e.g., various reachability roots)
39127. * Heap flags
39128. value1 -> return value, pseudo-type to indicate a return continuation (for ENDFIN)
39129. * Raw memory calls: relative to heap, but no GC interaction
39130. mark-and-sweep flags automatically active (used for critical sections)
39131. heap string indices are autogenerated in duk_strings.h
39132. * Prototypes
39133. 50x heap size
39134. string access cache (codepoint offset -> byte offset) for fast string * character looping; 'weak' reference which needs special handling in GC.
39135. currently running thread
39136. fixed top level hashtable size (separate chaining)
39137. longjmp state
39138. debugger detaching; used to avoid calling detach handler recursively
39139. required
39140. **comment:** A 16-bit listlen makes sense with 16-bit heap pointers: there * won't be space for 64k strings anyway.
 label: code-design
39141. built-in strings
39142. executor interrupt running (used to avoid nested interrupts)
39143. **comment:** * Memory calls: relative to heap, GC interaction, but no error throwing. ** XXX: Currently a mark-and-sweep triggered by memory allocation will run * using the heap->heap_thread. This thread is also used for running * mark-and-sweep finalization; this is not ideal because it breaks the * isolation between multiple global environments. ** Notes: * * - DUK_FREE() is required to ignore NULL and any other possible return * value of a zero-sized alloc/realloc (same as ANSI C free()). * * - There is no DUK_REALLOC_ZEROED because we don't assume to know the * old size. Caller must zero the reallocated memory. * * - DUK_REALLOC_INDIRECT() must be used when a mark-and-sweep triggered * by an allocation failure might invalidate the original 'ptr', thus * causing a realloc retry to use an invalid pointer. Example: we're * reallocating the value stack and a finalizer resizes the same value * stack during mark-and-sweep. The indirect variant requests for the * current location of the pointer being reallocated using a callback * right before every realloc attempt; this circuitous approach is used * to avoid strict aliasing issues in a more straightforward indirect * pointer (void **) approach. Note: the pointer in the storage * location is read but is NOT updated; the caller must do that.
 label: code-design
39144. current setjmp() catchpoint
39145. isError flag for yield
39146. **comment:** XXX: make active breakpoints actual copies instead of pointers?
 label: code-design
39147. value1 -> label number, pseudo-type to indicate a continue continuation (for ENDFIN)
39148. These are for rate limiting Status notifications and transport peeking.
39149. Used to support single-byte stream lookahead.
39150. duk_handle_call / duk_handle_safe_call recursion depth limiting
39151. **comment:** unused
 label: code-design
39152. * Main heap structure
39153. currently active breakpoints: NULL term, borrowed pointers
39154. 2nd related value (type specific)
39155. resizing parameters
39156. 1st related value (type specific)
39157. an error handler (user callback to augment/replace error) is running
39158. mix-in value for computing string hashes; should be reasonably unpredictable
39159. Starting from this round, use emergency mode * for mark-and-sweep.
39160. work list for objects to be finalized (by mark-and-sweep)
39161. value of dbg_exec_counter when we last did a Date-based check
39162. * Stringtable
39163. DUK_USE_DEBUGGER_SUPPORT
39164. mark-and-sweep marking reached a recursion limit and must use multi-pass marking
39165. callstack index
39166. debugger
39167. DUK_HEAP_H_INCLUDED
39168. cumulative opcode execution count (overflows are OK)
39169. **comment:** alloc function typedefs in duktape.h
 label: code-design
39170. value1 -> yield value, isError -> error / normal
39171. work list for objects whose refcounts are zero but which have not been * "finalized"; avoids recursive C calls when refcounts go to zero in a * chain of objects.
39172. strings up to this length are not cached
39173. * Built-in strings
39174. * Debugger support

39175. 1x heap size
39176. currently paused: talk with debug client until step/resume
39177. don't resize stringtable (but may sweep it); needed during stringtable resize
39178. rnd_state for duk_util_tinyrandom.c
39179. string intern table (weak refs)
39180. load factor max 75%
39181. Currently nothing to free
39182. No str[s] pointer.
39183. explicit NULL inits
39184. some derived types
39185. **comment:** * Allocate heap struct ** Use a raw call, all macros expect the heap to be initialized
label: code-design
39186. **comment:** suppress warning, not used
label: code-design
39187. Bernstein hash init value is normally 5381; XOR it in in case pointer low bits are 0
39188. Run mark-and-sweep a few times just in case (unreachable object * finalizers run already here). The last round must rescue objects * from the previous round without running any more finalizers. This * ensures rescued objects get their FINALIZED flag cleared so that * their finalizer is called once more in forced finalization to * satisfy finalizer guarantees. However, we don't want to run any * more finalizer because that'd required one more loop, and so on.
39189. **comment:** Similar workaround for INFINITY.
label: code-design
39190. **comment:** XXX: make a common DUK_USE_ option, and allow custom fixed seed?
label: code-design
39191. Note: heap->heap_thread, heap->curr_thread, and heap->heap_object * are on the heap allocated list.
39192. * All done
39193. 'thr' is now reachable
39194. **comment:** These is not 100% because format would need to be non-portable "long long". * Also print out as doubles to catch cases where the "long" type is not wide * enough; the limits will then not be printed accurately but the magnitude * will be correct.
label: code-design
39195. **comment:** Silence a few global unused warnings here.
label: code-design
39196. Internal keys are prefixed with 0xFF in the stringtable * (which makes them invalid UTF-8 on purpose).
39197. **comment:** XXX: here again finalizer thread is the heap_thread which needs * to be coordinated with finalizer thread fixes.
label: code-design
39198. DUK_USE_ROM_STRINGS
39199. DUK_USE_HEAPPTR16
39200. assumes that allocated pointers and alloc funcs are valid * if res exists
39201. **comment:** No need to length check string: it will never exceed even * the 16-bit length maximum.
label: code-design
39202. basic platform types
39203. Currently nothing to free; 'data' is a heap object
39204. mark-and-sweep not running -> must be empty
39205. The casts through duk_intr_pt is to avoid the following GCC warning: ** warning: cast from pointer to integer of different size [-Wpointer-to-int-cast] ** This still generates a /Wp64 warning on VS2010 when compiling for x86.
39206. Only objects in heap_allocated may have finalizers. Check that * the object itself has a _Finalizer property (own or inherited) * so that we don't execute finalizers for e.g. Proxy objects.
39207. * Init the heap object
39208. **comment:** XXX: zero assumption
label: code-design
39209. unsigned
39210. **comment:** XXX: Add a flag to reject an attempt to re-attach? Otherwise * the detached callback may immediately reattach.
label: code-design
39211. Cannot wrap: each object is at least 8 bytes so count is * at most 1/8 of that.
39212. * Init stringtable: fixed variant
39213. Fixed seed value used with ROM strings.
39214. **comment:** Workaround for some exotic platforms where NAN is missing * and the expression (0.0 / 0.0) does NOT result in a NaN. * Such platforms use the global 'duk_computed_nan' which must * be initialized at runtime. Use 'volatile' to ensure that * the compiler will actually do the computation and not try * to do constant folding which might result in the original * problem.
label: code-design
39215. refzero not running -> must be empty
39216. We don't log or warn about freeing zero refcount objects * because they may happen with finalizer processing.
39217. * Init the heap thread
39218. Constants for built-in string data depacking.
39219. structs from duk_forwdecl.h
39220. * Zero the struct, and start initializing roughly in order
39221. * duk_heap allocation and freeing.
39222. convenience
39223. just one 'int' for C++ exceptions
39224. **comment:** XXX: use the pointer as a seed for now: mix in time at least
label: code-design
39225. 0 = uppercase, 32 = lowercase (= 'a' - 'A')
39226. * Init built-in strings
39227. **comment:** XXX: with 'caller' property the callstack would need * to be unwound to update the 'caller' properties of * functions in the callstack.
label: code-design
39228. **comment:** Note: first argument not really used
label: code-design
39229. * Free the heap. ** Frees heap-related non-heap-tracked allocations such as the * string intern table; then frees the heap allocated objects; * and finally frees the heap structure itself. Reference counts * and GC markers are ignored (and not updated) in this process, * and finalizers won't be called. ** The heap pointer and heap object pointers must not be used * after this call.
39230. rescue no longer supported
39231. Special flags checks. Since these strings are always * reachable and a string cannot appear twice in the string * table, there's no need to check/set these flags elsewhere. * The 'internal' flag is set by string intern code.
39232. res->mark_and_sweep_trigger_counter == 0 -> now causes immediate GC; which is OK
39233. important chosen base types
39234. * Allocate a heap. ** String table is initialized with built-in strings from genbuiltins.py, * either by dynamically creating the strings or by referring to ROM strings.
39235. **comment:** Execute finalizers before freeing the heap, even for reachable * objects, and regardless of whether or not mark-and-sweep is * enabled. This gives finalizers the chance to free any native * resources like file handles, allocations made outside Duktape, * etc. This is quite tricky to get right, so that all finalizer * guarantees are honored. ** XXX: this perhaps requires an execution time limit.

label: code-design

39236. * Computed values (e.g. INFINITY)
39237. skip finalizers; queue finalizable objects to heap_allocated
39238. tval
39239. DUK_USE_DEBUG
39240. basic types from duk_features.h
39241. * Init stringcache
39242. no res->strs[]
39243. DUK_USE_EXPLICIT_NULL_INIT
39244. * Free a heap object. ** Free heap object and its internal (non-heap) pointers. Assumes that * caller has removed the object from heap allocated list or the string * intern table, and any weak references (which strings may have) have * been already dealt with.
39245. Detach a debugger if attached (can be called multiple times) * safely.
39246. res->strs16[] is zeroed and zero decodes to NULL, so no NULL inits.
39247. Decrease by 25% every round
39248. Each round of finalizer execution may spawn new finalizable objects * which is normal behavior for some applications. Allow multiple * rounds of finalization, but use a shrinking limit based on the * first round to detect the case where a runaway finalizer creates * an unbounded amount of new finalizable objects. Finalizer rescue * is not supported: the semantics are unclear because most of the * objects being finalized here are already reachable. The finalizer * is given a boolean to indicate that rescue is not possible. ** See discussion in: <https://github.com/svaarala/duktape/pull/473>
39249. nothing to NULL
39250. derived types
39251. strings are only tracked by stringtable
39252. Don't free h->resumer because it exists in the heap. * Callstack entries also contain function pointers which * are not freed for the same reason.
39253. DUK_USE_STRTAB_PROBE
39254. maps to finalizer 2nd argument
39255. * If selftests enabled, run them as early as possible
39256. zero
39257. **comment:** XXX: The incref macro takes a thread pointer but doesn't * use it right now.
label: code-design
39258. **comment:** XXX: error handling is incomplete. It would be cleanest if * there was a setjmp catchpoint, so that all init code could * freely throw errors. If that were the case, the return code * passing here could be removed.
label: code-design
39259. DUK_USE_STRTAB_CHAIN
39260. Prevent mark-and-sweep for the pending finalizers, also prevents * refzero handling from moving objects away from the heap_allocated * list. (The flag meaning is slightly abused here.)
39261. default prototype (Note: 'thr' must be reachable)
39262. basic types
39263. **comment:** XXX: this may now fail, and is not handled correctly
label: code-design
39264. * Debug dump type sizes
39265. With ROM-based strings, heap->strs[] and thr->strs[] are omitted * so nothing to initialize for str[].
39266. * Init stringtable: probe variant
39267. **comment:** XXX: inefficient loop
label: code-design
39268. note: mixing len into seed improves hashing when skipping
39269. **comment:** Use Murmurhash2 directly for short strings, and use "block skipping" * for long strings: hash an initial part and then sample the rest of * the string with reasonably sized chunks. An initial offset for the * sampling is computed based on a hash of the initial part of the string; * this is done to (usually) avoid the case where all long strings have * certain offset ranges which are never sampled. ** Skip should depend on length and bound the total time to roughly * logarithmic. With current values: ** 1M string => 256 * 241 = 61696 bytes (0.06M) of hashing * 1G string => 256 * 16321 = 4178176 bytes (3.98M) of hashing ** XXX: It would be better to compute the skip offset more "smoothly" * instead of having a few boundary values.
label: code-design
39270. Bernstein hash init value is normally 5381
39271. Constants for duk_hashstring().
39272. DUK_USE_STRHASH_DENSE
39273. **comment:** Truncate to 16 bits here, so that a computed hash can be compared * against a hash stored in a 16-bit field.
label: code-design
39274. **comment:** * String hash computation (interning). ** String hashing is performance critical because a string hash is computed * for all new strings which are candidates to be added to the string table. * However, strings actually added to the string table go through a codepoint * length calculation which dominates performance because it goes through * every byte of the input string (but only for strings added). ** The string hash algorithm should be fast, but on the other hand provide * good enough hashes to ensure both string table and object property table * hash tables work reasonably well (i.e., there aren't too many collisions * with real world inputs). Unless the hash is cryptographic, it's always * possible to craft inputs with maximal hash collisions. ** NOTE: The hash algorithms must match src/dukutil.py:duk_heap_hashstring() * for ROM string support!
label: code-design
39275. off >= step, and step >= 1
39276. Slightly modified "Bernstein hash" from: ** http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx ** Modifications: string skipping and reverse direction similar to * Lua 5.1.5, and different hash initializer. ** The reverse direction ensures last byte it always included in the * hash which is a good default as changing parts of the string are * more often in the suffix than in the prefix.
39277. may be NULL, too
39278. last element that was left in the heap
39279. weak refs should be handled here, but no weak refs for * any non-string objects exist right now.
39280. Note: object cannot be a finalizable unreachable object, as * they have been marked temporarily reachable for this round, * and are handled above.
39281. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize or a forced relocating realloc?
label: code-design
39282. * Finish
39283. * If object has been marked finalizable, move it to the * "to be finalized" work list. It will be collected on * the next mark-and-sweep if it is still unreachable * after running the finalizer.
39284. * String table resize check. ** Note: this may silently (and safely) fail if GC is caused by an * allocation call in stringtable resize_hash(). Resize_hash() * will prevent a recursive call to itself by setting the * DUK_MS_FLAG_NO_STRINGTABLE_RESIZE in heap->mark_and_sweep_base_flags.
39285. heaphdr: * - is not reachable * - is an object * - is not a finalized object * - has a finalizer
39286. finally free the struct itself
39287. deal with weak references first
39288. **comment:** XXX: disable error handlers for duration of compaction?
label: code-design
39289. * Fallback marking handler if recursion limit is reached. ** Iterates 'temproots' until recursion limit is no longer hit. Note * that temproots may reside either in heap allocated list or the * refzero work list. This is a slow scan, but guarantees that we * finish with a bounded C stack. ** Note that nodes may have been marked as temproots before this * scan begun, OR they may have been marked during the scan (as * we process nodes recursively also during the scan). This is * intended behavior.
39290. XXX: for threads, compact value stack, call stack, catch stack
39291. only objects have finalizers

39292. **comment:** * Sweep garbage and remove marking flags, and move objects with * finalizers to the finalizer work list. ** Objects to be swept need to get their refcounts finalized before * they are swept. In other words, their target object refcounts * need to be decreased. This has to be done before freeing any * objects to avoid decreffing dangling pointers (which may happen * even without bugs, e.g. with reference loops) ** Because strings don't point to other heap objects, similar * finalization is not necessary for strings.

label: code-design

39293. this case can no longer occur because refcount is unsigned

39294. DUK_USE_HEAPPTR16

39295. free object and all auxiliary (non-heap) allocs

39296. **comment:** XXX: more emergency behavior, e.g. find smaller hash sizes etc

label: code-design

39297. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).

39298. Strings and ROM objects are never placed on the heap allocated list.

39299. queue back to heap_allocated

39300. * Assertions after

39301. 'data' is reachable through every compiled function which * contains a reference.

39302. XXX: skip count_free w/o debug?

39303. * Object will be kept; queue object back to heap_allocated (to tail)

39304. * Finalizer torture. Do one fake finalizer call which causes side effects * similar to one or more finalizers on actual objects.

39305. XXX: will need a force flag if garbage collection is triggered * explicitly during paused state.

39306. DUK_USE_ASSERTIONS

39307. DUK_USE_MARK_AND_SWEEP

39308. * Assertion helpers.

39309. must never longjmp

39310. * Object compaction. ** Compaction is assumed to never throw an error.

39311. Cannot simulate individual finalizers because finalize_list only * contains objects with actual finalizers. But simulate side effects * from finalization by doing a bogus function call and resizing the * stacks.

39312. nothing to mark

39313. If debugger is paused, garbage collection is disabled by default.

39314. main reachability roots

39315. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).

39316. **comment:** remove the string (mark DELETED), could also call * duk_heap_string_remove() but that would be slow and * pointless because we already know the slot.

label: code-design

39317. * Plain, boring reachable object.

39318. XXX: stringtable emergency compaction?

39319. recursion tracking happens here only

39320. reset voluntary gc trigger count

39321. * Main mark-and-sweep function. ** 'flags' represents the features requested by the caller. The current * heap->mark_and_sweep_base_flags is ORed automatically into the flags; * the base flags mask typically prevents certain mark-and-sweep operations * to avoid trouble.

39322. **comment:** If thr != NULL, the thr may still be in the middle of * initialization. * XXX: Improve the thread viability test.

label: code-design

39323. Note: DUK_HEAP_HAS_REFZERO_FREE_RUNNING(heap) may be true; a refcount * finalizer may trigger a mark-and-sweep.

39324. * Run (object) finalizers in the "to be finalized" work list.

39325. **comment:** XXX: use advancing pointers instead of index macros -> faster and smaller?

label: code-design

39326. * Object's finalizer was executed on last round, and * object has been happily rescued.

39327. DUK_USE_MARKANDSWEEP_FINALIZER_TORTURE

39328. * Assertions before

39329. * Unreachable object, free

39330. OK

39331. done so that duk_mark_heaphdr() works correctly

39332. must also check refzero_list

39333. * Mark objects on finalize_list. *

39334. DUK_USE_REFERENCE_COUNTING

39335. XXX: remove this feature entirely? it would only matter for * emergency GC. Disable for lowest memory builds.

39336. free inner references (these exist e.g. when external * strings are enabled)

39337. ignored

39338. Non-zero refcounts should not happen because we refcount * finalize all unreachable objects which should cancel out * refcounts (even for cycles).

39339. * Mark-and-sweep garbage collection.

39340. * Mark refzero_list objects. ** Objects on the refzero_list have no inbound references. They might have * outbound references to objects that we might free, which would invalidate * any references held by the refzero objects. A refzero object might also * be rescued by refcount finalization. Refzero objects are treated as * reachability roots to ensure they (or anything they point to) are not * freed in mark-and-sweep.

39341. * Mark the heap.

39342. **comment:** * Object compaction (emergency only). ** Object compaction is a separate step after sweeping, as there is * more free memory for it to work with. Also, currently compaction * may insert new objects into the heap allocated list and the string * table which we don't want to do during a sweep (the reachability flags of such objects would be incorrect). The objects inserted * are currently: ** - a temporary duk_hbuffer for a new properties allocation * - if array part is abandoned, string keys are interned * * The object insertions go to the front of the list, so they do not * cause an infinite loop (they are not compacted).

label: code-design

39343. mark finalizer work list as reachability roots

39344. **comment:** Require a lot of stack to force a value stack grow/shrink. * Recursive mark-and-sweep is prevented by allocation macros * so this won't trigger another mark-and-sweep.

label: code-design

39345. * Mark unreachable, finalizable objects. ** Such objects will be moved aside and their finalizers run later. They have * to be treated as reachability roots for their properties etc to remain * allocated. This marking is only done for unreachable values which would * be swept later (refzero_list is thus excluded). ** Objects are first marked FINALIZABLE and only then marked as reachability * roots; otherwise circular references might be handled inconsistently.

39346. temproots

39347. refzero_list treated as reachability roots

39348. nothing now

39349. * Finalize refcounts for heap elements just about to be freed. * This must be done for all objects before freeing to avoid any * stale pointer dereferences. ** Note that this must deduce the set of objects to be freed * identically to duk_sweep_heap().

39350. flags have been already cleared

39351. * Reset trigger counter

39352. **comment:** XXX: duk_small_uint_t would be enough for this loop

label: code-design

39353. **comment:** Run fake finalizer. Avoid creating unnecessary garbage.

label: code-design

39354. * Unreachable object about to be swept. Finalize target refcounts * (objects which the unreachable object points to) without doing * refzero processing. Recursive decrefs are also prevented when * refzero processing is disabled. ** Value cannot be a finalizable object, as they have been made * temporarily reachable for this round.

39355. Caller will finish the marking process if we hit a recursion limit.

39356. no mark-and-sweep gc

39357. may have FINALIZED

39358. log this with a normal debug level because this should be relatively rare

39359. * Marking functions for heap types: mark children recursively

39360. **comment:** * Finalize objects in the finalization work list. Finalized * objects are queued back to heap_allocated with FINALIZED set. ** Since finalizers may cause arbitrary side effects, they are * prevented during string table and object property allocation * resizing using the DUK_MS_FLAG_NO_FINALIZERS flag in * heap->mark_and_sweep_base_flags. In this case the objects * remain in the finalization work list after mark-and-sweep * exits and they may be finalized on the next pass. ** Finalization currently happens inside "MARKANDSWEEP_RUNNING" * protection (no mark-and-sweep may be triggered by the * finalizers). As a side effect: ** 1) an out-of-memory error inside a finalizer will not * cause a mark-and-sweep and may cause the finalizer * to fail unnecessarily ** 2) any temporary objects whose refcount decreases to zero * during finalization will not be put into refzero_list; * they can only be collected by another mark-and-sweep ** This is not optimal, but since the sweep for this phase has * already happened, this is probably good enough for now.

label: code-design

39361. hash part is a 'weak reference' and does not contribute

39362. **comment:** Checking this here rather than in memory alloc primitives * reduces checking code there but means a failed allocation * will go through a few retries before giving up. That's * fine because this only happens during debugging.

label: code-design

39363. DUK_HEAPHDR_HAS_FINALIZED may be set if we're doing a * refzero finalization and mark-and-sweep gets triggered * during the finalizer.

39364. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers). A prototype loop must not cause * an error to be thrown here; duk_hobject_hasprop_raw() will ignore a * prototype loop silently and indicate that the property doesn't exist.

39365. nothing to process

39366. * Clear (reachable) flags of refzero work list.

39367. Keep heap->finalize_list up-to-date during the list walk. * This has no functional impact, but does matter e.g. for * duk_push_heapptr() asserts when assertions are enabled.

39368. * Begin

39369. * Mark roots, hoping that recursion limit is not normally hit. * If recursion limit is hit, run additional reachability rounds * starting from "temproots" until marking is complete. ** Marking happens in two phases: first we mark actual reachability * roots (and run "temproots" to complete the process). Then we * check which objects are unreachable and are finalizable; such * objects are marked as FINALIZABLE and marked as reachability * (and "temproots" is run again to complete the process). ** The heap finalize_list must also be marked as a reachability root. * There may be objects on the list from a previous round if the * previous run had finalizer skip flag.

39370. Run the finalizer, duk_hobject_run_finalizer() sets FINALIZED. * Next mark-and-sweep will collect the object unless it has * become reachable (i.e. rescued). FINALIZED prevents the * finalizer from being executed again before that.

39371. nop

39372. DUK_USE_STRTAB_PROBE

39373. * Reachable object, keep

39374. May happen in some out-of-memory corner cases.

39375. DUK_HEAPHDR_HAS_FINALIZED may or may not be set.

39376. * Sweep stringtable

39377. An object may be in heap_allocated list with a zero * refcount also if it is a temporary object created by * a finalizer; because finalization now runs inside * mark-and-sweep, such objects will not be queued to * refzero_list and will thus appear here with refcount * zero.

39378. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed.

39379. **comment:** Select a thread for mark-and-sweep use. ** XXX: This needs to change later.

label: code-design

39380. An object may be in heap_allocated list with a zero * refcount if it has just been finalized and is waiting * to be collected by the next cycle.

39381. * Sweep heap

39382. **comment:** XXX: could clear FINALIZED already here; now cleared in * next mark-and-sweep.

label: code-design

39383. DUK_USE_STRTAB_CHAIN

39384. finalize_list will always be processed completely

39385. approximate

39386. **comment:** XXX: thread selection for mark-and-sweep is currently a hack. * If we don't have a thread, the entire mark-and-sweep is now * skipped (although we could just skip finalizations).

label: code-design

39387. **comment:** XXX: which lists should participate? to be finalized?

label: code-design

39388. nargs

39389. mark finalizable as reachability roots

39390. * Clear (reachable) flags of finalize_list ** We could mostly do in the sweep phase when we move objects from the * heap into the finalize_list. However, if a finalizer run is skipped * during a mark-and-sweep, the objects on the finalize_list will be marked * reachable during the next mark-and-sweep. Since they're already on the * finalize_list, no-one will be clearing their REACHABLE flag so we do it * here. (This now overlaps with the sweep handling in a harmless way.)

39391. Used during heap destruction: don't actually run finalizers * because we're heading into forced finalization. Instead, * queue finalizable objects back to the heap_allocated list.

39392. No finalizers for ROM objects

39393. * Misc

39394. * Memory allocation handling.

39395. Note: key issue here is to re-lookup the base pointer on every attempt. * The pointer being reallocated may change after every mark-and-sweep.

39396. for zero size allocations NULL is allowed

39397. **comment:** * Compared to a direct macro expansion this wrapper saves a few * instructions because no heap dereferencing is required.

label: code-design

39398. Note: must behave like a no-op with NULL and any pointer * returned from malloc/realloc with zero size.

39399. simulate alloc failure on every realloc (except when mark-and-sweep is running)

39400. * Helpers ** The fast path checks are done within a macro to ensure "inlining" * while the slow path actions use a helper (which won't typically be * inlined in size optimized builds).

39401. * Avoid a GC if GC is already running. This can happen at a late * stage in a GC when we try to e.g. resize the stringtable * or compact objects.

39402. * Reallocate memory with garbage collection, using a callback to provide * the current allocated pointer. This variant is used when a mark-and-sweep * (e.g. finalizers) might change the original pointer.

39403. no voluntary gc

39404. * First attempt

39405. simulate alloc failure on every alloc (except when mark-and-sweep is running)

39406. DUK_USE_MARK_AND_SWEEP

39407. assume memset with zero size is OK

39408. * Allocate memory with garbage collection

39409. pt before mark-and-sweep

39410. Must behave like a no-op with NULL and any pointer returned from * malloc/realloc with zero size.

39411. saves a few instructions to have this wrapper (see comment on duk_heap_mem_alloc)

39412. Count free operations toward triggering a GC but never actually trigger * a GC from a free. Otherwise code which frees internal structures would * need to put in NULLs at every turn to ensure the object is always in * consistent state for a mark-and-sweep.

39413. useful for debugging

39414. ptr may be NULL

39415. * Relocate memory with garbage collection

39416. * Retry with several GC attempts. Initial attempts are made without * emergency mode; later attempts use emergency mode which minimizes * memory allocations forcibly.

39417. * Free memory

39418. * Voluntary periodic GC (if enabled)

39419. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC

39420. * Avoid a GC if GC is already running. See duk_heap_mem_alloc().

39421. For initial entry use default value; zero forces an * interrupt before executing the first instruction.

39422. DUK_USE_INTERRUPT_COUNTER

39423. * Support functions for duk_heap.

39424. may be NULL

39425. arbitrary remove only works with double linked heap, and is only required by * reference counting so far.

39426. **comment:** The prev/next pointers of the removed duk_heaphdr are left as garbage. * It's up to the caller to ensure they're written before inserting the * object back.
label: code-design

39427. Copy interrupt counter/init value state to new thread (if any). * It's OK for new_thr to be the same as curr_thr.

39428. yes -> move back to heap allocated

39429. Require a lot of stack to force a value stack grow/shrink.

39430. * Refcount memory freeing loop. ** Frees objects in the refzero_pending list until the list becomes * empty. When an object is freed, its references get decref'd and * may cause further objects to be queued for freeing. ** This could be expanded to allow incremental freeing: just bail out * early and resume at a future alloc/decref/refzero.

39431. 'count' is more or less comparable to normal trigger counter update * which happens in memory block (re)allocation.

39432. * Buffers have no internal references. However, a dynamic * buffer has a separate allocation for the buffer. This is * freed by duk_heap_free_heapdr_raw().

39433. DUK_USE_REFERENCE_COUNTING

39434. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed, * and we're in an infinite loop.

39435. DUK_USE_REFZERO_FINALIZER_TORTURE

39436. no -> decref members, then free

39437. Run fake finalizer. Avoid creating new refzero queue entries * so that we are not forced into a forever loop.

39438. ignored

39439. no refcounting

39440. **comment:** Avoid fake finalization for the duk_refcount_fake_finalizer function * itself, otherwise we're in infinite recursion.
label: code-design

39441. * Finalizer check. ** Note: running a finalizer may have arbitrary side effects, e.g. * queue more objects on refzero_list (tail), or even trigger a * mark-and-sweep.
** Note: quick reject check should match vast majority of * objects and must be safe (not throw any errors, ever).

39442. An object may have FINALIZED here if it was finalized by mark-and-sweep * on a previous run and refcount then decreased to zero. We won't run the * finalizer again here.

39443. refcount is unsigned, so always true

39444. currently, always the case

39445. not emergency

39446. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers).

39447. must never longjmp

39448. Torture option to shake out finalizer side effect issues: * make a bogus function call for every finalizable object, * essentially simulating the case where everything has a * finalizer.

39449. * Rescue or free.

39450. May happen in some out-of-memory corner cases.

39451. * Once the whole refzero cascade has been freed, check for * a voluntary mark-and-sweep.

39452. tail insert: don't disturb head in case refzero is running

39453. * Detect recursive invocation

39454. nothing to finalize

39455. **comment:** unused
label: code-design

39456. nothing now

39457. Refzero head is still the same. This is the case even if finalizer * inserted more refzero objects; they are inserted to the tail.

39458. not strictly necessary

39459. * Strings have no internal references but do have "weak" * references in the string cache. Also note that strings * are not on the heap_allocated list like other heap * elements.

39460. duk_hobject_run_finalizer() sets

39461. * Inref and decref functions. * Decref may trigger immediate refzero handling, which may free and finalize * an arbitrary number of objects. *

39462. * Heap object refcount finalization. ** When an object is about to be freed, all other objects it refers to must * be decref'd. Refcount finalization does NOT free the object or its inner * allocations (mark-and-sweep shares these helpers), it just manipulates * the refcounts. ** Note that any of the decref's may cause a refcount to drop to zero, BUT * it will not be processed inline; instead, because refzero is already * running, the objects will just be queued to refzero list and processed * later. This eliminates C recursion.

39463. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize?
label: code-design

39464. bump refcount to prevent refzero during finalizer processing

39465. * Objects have internal references. Must finalize through * the "refzero" work list.

39466. * Pick an object from the head (don't remove yet).

39467. **comment:** XXX: ideally this would be just one flag (maybe a derived one) so * that a single bit test is sufficient to check the condition.
label: test

39468. approximate

39469. * Remove the object from the refzero list. This cannot be done * before a possible finalizer has been executed; the finalizer * may trigger a mark-and-sweep, and mark-and-sweep must be able * to traverse a complete refzero_list.

39470. **comment:** XXX: better to get base and walk forwards?
label: code-design

39471. nargs

39472. * Churn refzero_list until empty

39473. remove artificial bump

39474. cannot happen: strings are not put into refzero list (they don't even have the next/prev pointers)

39475. hash part is a 'weak reference' and does not contribute

39476. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC

39477. * Reference counting implementation.

39478. **comment:** * Refzero handling is skipped entirely if (1) mark-and-sweep is * running or (2) execution is paused in the debugger. The objects * are left in the heap, and will be freed by mark-and-sweep or * eventual heap destruction. ** This is necessary during mark-and-sweep because refcounts are also * updated during the sweep phase (otherwise objects referenced by a * swept object would have incorrect refcounts) which then calls here. * This could be avoided by using separate decref macros in * mark-and-sweep; however, mark-and-sweep also calls finalizers which * would use the ordinary decref macros anyway and still call this *

function. ** This check must be enabled also when mark-and-sweep support has been * disabled: the flag is also used in heap destruction when running * finalizers for remaining objects, and the flag prevents objects from * being moved around in heap linked lists.

label: code-design

39479. * Misc

39480. **comment:** * Delete references to given hstring from the heap string cache. ** String cache references are 'weak': they are not counted towards * reference counts, nor serve as roots for mark-and-sweep. When an * object is about to be freed, such references need to be removed.

label: code-design

39481. update entry, allocating if necessary

39482. may be equal

39483. * String scanning helpers ** All bytes other than UTF-8 continuation bytes ([0x80,0xbff]) are * considered to contribute a character. This must match how string * character length is computed.

39484. take last entry

39485. * Scan from shortest distance: * - start of string * - end of string * - cache entry (if exists)

39486. LRU: move our entry to first

39487. **comment:** * Convert char offset to byte offset ** Avoid using the string cache if possible: for ASCII strings byte and * char offsets are equal and for short strings direct scanning may be * better than using the string cache (which may evict a more important * entry). ** Typing now assumes 32-bit string byte/char offsets (duk_uint_fast32_t). * Better typing might be to use duk_size_t.

label: code-design

39488. no sce, or sce scan not best

39489. * Update cache entry (allocating if necessary), and move the * cache entry to the first place (in an "LRU" policy).

39490. clen == blen -> pure ascii

39491. **comment:** * For non-ASCII strings, we need to scan forwards or backwards * from some starting point. The starting point may be the start * or end of the string, or some cached midpoint in the string * cache. ** For "short" strings we simply scan without checking or updating * the cache. For longer strings we check and update the cache as * necessary, inserting a new cache entry if none exists.

label: code-design

39492. **comment:** 'sce' points to the wrong entry here, but is no longer used

label: code-design

39493. Scan error: this shouldn't normally happen; it could happen if * string is not valid UTF-8 data, and clen/blen are not consistent * with the scanning algorithm.

39494. **comment:** XXX: the string shouldn't appear twice, but we now loop to the * end anyway; if fixed, add a looping assertion to ensure there * is no duplicate.

label: code-design

39495. * For ASCII strings, the answer is simple.

39496. * String cache. ** Provides a cache to optimize indexed string lookups. The cache keeps * track of (byte offset, char offset) states for a fixed number of strings. * Otherwise we'd need to scan from either end of the string, as we store * strings in (extended) UTF-8.

39497. initialize for debug prints, needed if sce==NULL

39498. * A C * B A * C <- sce ==> B * D D

39499. may be less, since DELETED entries are NULled by rehash

39500. **comment:** load factor too low or high, count actually used entries and resize

label: code-design

39501. Count actually used (non-NULL, non-DELETED) entries.

39502. OK

39503. avoid array abandoning which interns strings

39504. unsigned intentionally

39505. dummy

39506. checking for DUK__DELETED_MARKER is not necessary here, but helper does it now

39507. may be NULL

39508. NUL terminate for convenient C access

39509. Undefine local defines

39510. If blen <= 0xffffUL, clen is also guaranteed to be <= 0xffffUL.

39511. Overflow, relevant mainly when listlen is 16 bits.

39512. find and remove string from stringtable; caller must free the string itself

39513. **comment:** * String table algorithm: fixed size string table with array chaining ** The top level string table has a fixed size, with each slot holding * either NULL, string pointer, or pointer to a separately allocated * string pointer list. ** This is good for low memory environments using a pool allocator: the * top level allocation has a fixed size and the pointer lists have quite * small allocation size, which further matches the typical pool sizes * needed by objects, strings, property tables, etc.

label: code-design

39514. char: use int cast

39515. guaranteed to succeed

39516. * Create a hstring and insert into the heap. The created object * is directly garbage collectable with reference count zero. ** The caller must place the interned string into the stringtable * immediately (without chance of a longjmp); otherwise the string * is lost.

39517. fail

39518. new_used / size <= 1 / DIV <=> new_used <= size / DIV

39519. DUK_USE_STRTAB_PROBE

39520. DUK_USE_DEBUG

39521. caller is responsible for ensuring this

39522. strings may have inner refs (extdata) in some cases

39523. grow by at most one

39524. Note: hstring is in heap but has refcount zero and is not strongly reachable. * Caller should increase refcount and make the hstring reachable before any * operations which require allocation (and possible gc).

39525. looping should never happen

39526. not strictly necessary

39527. Using an explicit 'ASCII' flag has larger footprint (one call site * only) but is quite useful for the case when there's no explicit * 'clen' in duk_hstring.

39528. **comment:** avoid pressure to add/remove strings, invalidation of call data argument, etc.

label: code-design

39529. rehash even if old and new sizes are the same to get rid of * DELETED entries.

39530. DUK_USE_ROM_STRINGS

39531. * Heap stringtable handling, string interning.

39532. Force a resize so that DELETED entries are eliminated. * Another option would be duk__recheck_strtab_size_probe(); * but since that happens on every intern anyway, this whole * check can now be disabled.

39533. **comment:** unused with some debug level combinations

label: code-design

39534. * The attempt to allocate may cause a GC. Such a GC must not attempt to resize * the stringtable (though it can be swept); finalizer execution and object * compaction must also be postponed to avoid the pressure to add strings to the * string table. Call site must prevent these.

39535. Because new_size > duk_count_used_probe(heap), guaranteed to work

39536. DUK_USE_HEAPPTR16

39537. failed

39538. avoid recursive string table call

39539. formatted result limited

39540. Free strings in the stringtable and any allocations needed * by the stringtable itself.

39541. * String table algorithm: closed hashing with a probe sequence ** This is the default algorithm and works fine for environments with * minimal memory constraints.
39542. DUK_USE_STRTAB_CHAIN
39543. Now two entries in the same slot, alloc list
39544. **comment:** XXX: could check for e16 == 0 because NULL is guaranteed to * encode to zero.
 label: code-design
39545. required for rehash to succeed, equality not that useful
39546. * Exposed calls
39547. **comment:** XXX: This is VERY inefficient now, and should be e.g. a * binary search or perfect hash, to be fixed.
 label: code-design
39548. All strings beginning with 0xff are treated as "internal", * even strings interned by the user. This allows user code to * create internal properties too, and makes behavior consistent * in case user code happens to use a string also used by Duktape * (such as string has already been interned and has the 'internal' * flag set).
39549. st_used remains the same, DELETED is counted as used
39550. **comment:** Prevent any side effects on the string table and the caller provided * str/blen arguments while interning is in progress. For example, if * the caller provided str/blen from a dynamic buffer, a finalizer might * resize that dynamic buffer, invalidating the call arguments.
 label: code-design
39551. For manual testing only.
39552. **comment:** unused
 label: code-design
39553. Relies on NULL encoding to zero.
39554. FAIL
39555. * Raw intern and lookup
39556. new_free / size <= 1 / DIV <=> new_free <= size / DIV
39557. Without ROM objects "needs refcount update" == is heap allocated.
39558. **comment:** * Reference counting helper macros. The macros take a thread argument * and must thus always be executed in a specific thread context. The * thread argument is needed for features like finalization. Currently * it is not required for INCREF, but it is included just in case. ** Note that 'raw' macros such as DUK_HEAPHDR_GET_REFCOUNT() are not * defined without DUK_USE_REFERENCE_COUNTING, so caller must #ifdef * around them.
 label: code-design
39559. **comment:** Slow variants, call to a helper to reduce code size. * Can be used explicitly when size is always more important than speed.
 label: code-design
39560. DUK_USE_REFERENCE_COUNTING
39561. refcount macros not defined without refcounting, caller must #ifdef now
39562. round out to 8 bytes
39563. currently nop
39564. result: updated refcount
39565. * Note: type is treated as a field separate from flags, so some masking is * involved in the macros below.
39566. refcounting requires direct heap frees, which in turn requires a dual linked heap
39567. 5 heap flags
39568. mark-and-sweep: finalized (on previous pass)
39569. Original idiom used, minimal code size.
39570. **comment:** XXX: fast int-to-double
 label: code-design
39571. * Assert helpers
39572. Optimized for size.
39573. DUK_USE_ROM_OBJECTS
39574. nop
39575. Optimized for speed.
39576. **comment:** XXX: double evaluation for 'tv' argument.
 label: code-design
39577. **comment:** * Common heap header ** All heap objects share the same flags and refcount fields. Objects other * than strings also need to have a single or double linked list pointers * for insertion into the "heap allocated" list. Strings are held in the * heap-wide string table so they don't need link pointers. ** Technically, 'h_refcount' must be wide enough to guarantee that it cannot * wrap (otherwise objects might be freed incorrectly after wrapping). This * means essentially that the refcount field must be as wide as data pointers. * On 64-bit platforms this means that the refcount needs to be 64 bits even * if an 'int' is 32 bits. This is a bit unfortunate, and compromising on * this might be reasonable in the future. ** Heap header size on 32-bit platforms: 8 bytes without reference counting, * 16 bytes with reference counting.
 label: code-design
39578. init pointer fields to null
39579. Check that prev/next links are consistent: if e.g. h->prev is != NULL, * h->prev->next should point back to h.
39580. DUK_USE_FASTINT
39581. **comment:** XXX: fast-int-to-double
 label: code-design
39582. read-only object, in code section
39583. 2 bits for heap type
39584. side effects
39585. mark-and-sweep: children not processed
39586. mark-and-sweep: finalizable (on current pass)
39587. Casting convenience.
39588. **comment:** Convenience for some situations; the above macros don't allow NULLs * for performance reasons.
 label: code-design
39589. **comment:** 16 bits would be enough for shared heaphdr flags and duk_hstring * flags. The initial parts of duk_heaphdr_string and duk_heaphdr * must match so changing the flags field size here would be quite * awkward. However, to minimize struct size, we can pack at least * 16 bits of duk_hstring data into the flags field.
 label: code-design
39590. * Heap header definition and assorted macros, including ref counting. * Access all fields through the accessor macros.
39591. Default variants. Selection depends on speed/size preference. * Concretely: with gcc 4.8.1 -Os x64 the difference in final binary * is about +1kB for _FAST variants.
39592. get or set a range of flags; m=first bit number, n=number of bits
39593. **comment:** * Macros to set a duk_tval and update refcount of the target (decref the * old value and incref the new value if necessary). This is both performance * and footprint critical; any changes made should be measured for size/speed.
 label: code-design
39594. **comment:** When DUK_USE_HEAPPTR16 (and DUK_USE_REFCOUNT16) is in use, the * struct won't align nicely to 4 bytes. This 16-bit extra field * is added to make the alignment clean; the field can be used by * heap objects when 16-bit packing is used. This field is now * conditional to DUK_USE_HEAPPTR16 only, but it is intended to be * used with DUK_USE_REFCOUNT16 and DUK_USE_DOUBLE_LINKED_HEAP; * this only matter to low memory environments anyway.
 label: code-design
39595. **comment:** Fast variants, inline refcount operations except for refzero handling. * Can be used explicitly when speed is always more important than size. * For a good compiler and a single file build, these are basically the * same as a forced inline.
 label: code-design

39596. 25 user flags
39597. DUK_HEAPHDR_H_INCLUDED
39598. With ROM objects "needs refcount update" is true when the value is * heap allocated and is not a ROM object.
39599. **comment:** XXX: no optimized variants yet
 label: requirement
39600. **comment:** Faster alternative: avoid making a temporary copy of tvptr_dst and use * fast incref/decref macros.
 label: code-design
39601. **comment:** DUK_TVAL_SET_TVAL_UPDREF() is used a lot in executor, property lookups, * etc, so it's very important for performance. Measure when changing. ** NOTE: the source and destination duk_tval pointers may be the same, and * the macros MUST deal with that correctly.
 label: code-design
39602. mark-and-sweep: reachable
39603. * Heap native function representation.
39604. **comment:** The 'magic' field allows an opaque 16-bit field to be accessed by the * Duktape/C function. This allows, for instance, the same native function * to be used for a set of very similar functions, with the 'magic' field * providing the necessary non-argument flags / values to guide the behavior * of the native function. The value is signed on purpose: it is easier to * convert a signed value to unsigned (simply AND with 0xffff) than vice * versa. ** Note: cannot place nargs/magic into the heapdr flags, because * duk_hobject takes almost all flags already (and needs the spare).
 label: code-design
39605. DUK_HNATIVEFUNCTION_H_INCLUDED
39606. shared object part
39607. hobject management functions
39608. !DUK_SINGLE_FILE
39609. LAYOUT 2
39610. misc
39611. * Macros to access the 'props' allocation.
39612. object has an array part (a_size may still be 0)
39613. * Exposed data
39614. **comment:** * Heap object representation. ** Heap objects are used for Ecmascript objects, arrays, and functions, * but also for internal control like declarative and object environment * records. Compiled functions, native functions, and threads are also * objects but with an extended C struct. ** Objects provide the required Ecmascript semantics and exotic behaviors * especially for property access. ** Properties are stored in three conceptual parts: ** 1. A linear 'entry part' contains ordered key-value-attributes triples * and is the main method of string properties. ** 2. An optional linear 'array part' is used for array objects to store a * (dense) range of [0,N] array indexed entries with default attributes * (writable, enumerable, configurable). If the array part would become * sparse or non-default attributes are required, the array part is * abandoned and moved to the 'entry part'. ** 3. An optional 'hash part' is used to optimize lookups of the entry * part; it is used only for objects with sufficiently many properties * and can be abandoned without loss of information. ** These three conceptual parts are stored in a single memory allocated area. * This minimizes memory allocation overhead but also means that all three * parts are resized together, and makes property access a bit complicated.
 label: code-design
39615. E5 Section 8.6.1
39616. push value to stack
39617. object is extensible
39618. 'Arguments' object and has arguments exotic behavior (non-strict callee)
39619. hash size approx. 1.25 times entries size
39620. * Struct defs
39621. **comment:** unused
 label: code-design
39622. maximum length for a SKIP-1 diffstream: 35 bits per entry, rounded up to bytes
39623. **comment:** XXX: when optimizing for guaranteed property slots, use a guaranteed * slot for internal value; this call can then access it directly.
 label: code-design
39624. **comment:** * 'props' contains {key,value,flags} entries, optional array entries, and * an optional hash lookup table for non-array entries in a single 'sliced' * allocation. There are several layout options, which differ slightly in * generated code size/speed and alignment/padding; duk_features.h selects * the layout used. *
 * Layout 1 (DUK_USE_HOBJECT_LAYOUT_1): ** e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_propvalue)
 bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted
 * * Layout 2 (DUK_USE_HOBJECT_LAYOUT_2): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_hstring)
 * bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_uint8_t) + pad bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused,
 0xfffffffffeUL = deleted * * Layout 3 (DUK_USE_HOBJECT_LAYOUT_3): ** e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) *
 a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) *
 h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffffeUL = deleted * e_size * sizeof(duk_uint8_t) bytes
 of entry flags (e_next gc reachable) * * In layout 1, the 'e_next' count is rounded to 4 or 8 on platforms * requiring 4 or 8 byte alignment. This ensures proper
 alignment * for the entries, at the cost of memory footprint. However, it's * probably preferable to use another layout on such platforms instead. * * In layout 2, the
 key and value parts are swapped to avoid padding * the key array on platforms requiring alignment by 8. The flags part * is padded to get alignment for array
 entries. The 'e_next' count does * not need to be rounded as in layout 1. * * In layout 3, entry values and array values are always aligned properly, * and assuming
 pointers are at most 8 bytes, so are the entry keys. Hash * indices will be properly aligned (assuming pointers are at least 4 bytes). * Finally, flags don't need
 additional alignment. This layout provides * compact allocations without padding (even on platforms with alignment * requirements) at the cost of a bit slower
 lookups. ** Objects with few keys don't have a hash index; keys are looked up linearly, * which is cache efficient because the keys are consecutive. Larger objects
 * have a hash index part which contains integer indexes to the entries part. ** A single allocation reduces memory allocation overhead but requires more * work
 when any part needs to be resized. A sliced allocation for entries * makes linear key matching faster on most platforms (more locality) and * skimps on flags size
 (which would be followed by 3 bytes of padding in * most architectures if entries were placed in a struct). ** 'props' also contains internal properties distinguished
 with a non-BMP * prefix. Often used properties should be placed early in 'props' whenever * possible to make accessing them as fast as possible.
 label: code-design
39625. Object flag. There are currently 26 flag bits available. Make sure * this stays in sync with debugger object inspection code.
39626. finalization
39627. probe sequence
39628. **comment:** function: function must not be tail called
 label: code-design
39629. function: create binding for func name (function templates only, used for named function expressions)
39630. Maximum traversal depth for "bound function" chains.
39631. entry part size
39632. additional flags which are passed in the same flags argument as property * flags but are not stored in object properties.
39633. * PC-to-line constants
39634. internal property functions
39635. object is a buffer object (duk_hbufferobject) (always exotic)
39636. * Macro for object validity check ** Assert for currently guaranteed relations between flags, for instance.
39637. internal define property: skip write silently if exists
39638. flags used for property attributes in duk_propdesc and packed flags
39639. object is a thread (duk_hthread)
39640. prop index in 'entry part', < 0 if not there

39641. **comment:** Macro for creating flag initializer from a class number. * Unsigned type cast is needed to avoid warnings about coercing * a signed integer to an unsigned one; the largest class values * have the highest bit (bit 31) set which causes this.
label: code-design

39642. **comment:** hash part size or 0 if unused
label: code-design

39643. Object built-in methods

39644. LAYOUT 1

39645. property is virtual: used in duk_propdesc, never stored * (used by e.g. buffer virtual properties)

39646. implies DUK_HOBJECT_IS_BUFFEROBJECT

39647. * Resizing and hash behavior

39648. custom; implies DUK_HOBJECT_IS_BUFFEROBJECT

39649. function: create new environment when called (see duk_hcompiledfunction)

39650. class masks

39651. $2^3 \rightarrow 1/8 = 12.5\%$ min growth

39652. convenience

39653. custom; implies DUK_HOBJECT_IS_THREAD

39654. for updating (all are set to < 0 for virtual properties)

39655. 'Proxy' object

39656. index for next new key ([0,e_next[are gc reachable)

39657. note: this updates refcounts

39658. alias for above

39659. 'Array' object, array length and index exotic behavior

39660. if density < L, abandon array part, L = 3-bit fixed point, e.g. 2 $\rightarrow 2/8 = 25\%$

39661. 25%, i.e. less than 25% used \rightarrow abandon

39662. Located in duk_heapdr.h_extra16. Subclasses of duk_hobject (like * duk_hcompiledfunction) are not free to use h_extra16 for this reason.

39663. hobject property layout

39664. core property functions

39665. DUK_HOBJECT_H_INCLUDED

39666. prop index in 'array part', < 0 if not there

39667. limit is quite low: one array entry is 8 bytes, one normal entry is 4+1+8+4 = 17 bytes (with hash entry)

39668. flags for duk_hobject_get_own_propdesc() and variants

39669. The get/set pointers could be 16-bit pointer compressed but it * would make no difference on 32-bit platforms because duk_tval is * 8 bytes or more anyway.

39670. * Prototypes

39671. * Macros for accessing size fields

39672. object is a compiled function (duk_hcompiledfunction)

39673. macros

39674. Duktape/C (nativefunction) object, exotic 'length'

39675. internal properties

39676. custom

39677. higher value conserves memory; also note that linear scan is cache friendly

39678. read-only values 'lifted' for ease of use

39679. Maximum prototype traversal depth. Sanity limit which handles e.g. * prototype loops (even complex ones like 1->2->3->4->2->3->4->2->3->4).

39680. envrec: (declarative) record is closed

39681. hash size relative to entries size: for value X, approx. hash_prime(e_size + e_size / X)

39682. LAYOUT 3

39683. internal align target for props allocation, must be 2^n for some n

39684. Sanity limit on max number of properties (allocated, not necessarily used). * This is somewhat arbitrary, but if we're close to 2^{32} properties some * algorithms will fail (e.g. hash size selection, next prime selection). * Also, we use negative array/entry table indices to indicate 'not found', * so anything above 0x80000000 will cause trouble now.

39685. object is constructable

39686. prototype: the only internal property lifted outside 'e' as it is so central

39687. controls for minimum array part growth

39688. * Macros for property handling

39689. pc2line

39690. ES6 proxy

39691. $2^{31} \approx 2G$ properties

39692. function: create an arguments object on function call

39693. accessor

39694. controls for minimum entry part growth

39695. range check not necessary because all 4-bit values are mapped

39696. XXX: duk_uarridx_t?

39697. object has any exotic behavior(s)

39698. E5 Section 8.6.2 + custom classes

39699. prop index in 'hash part', < 0 if not there

39700. don't throw for prototype loop

39701. low-level property functions

39702. object established using Function.prototype.bind()

39703. * Ecmascript [[Class]]

39704. 112.5%, i.e. new size less than 12.5% higher \rightarrow fast resize

39705. helpers for defineProperty() and defineProperties()

39706. object is a native function (duk_hnativefunction)

39707. if new_size < L * old_size, resize without abandon check; L = 3-bit fixed point, e.g. 9 $\rightarrow 9/8 = 112.5\%$

39708. function: function object is strict

39709. enumeration

39710. array part size (entirely gc reachable)

39711. 'String' object, array index exotic behavior

39712. alloc and init

39713. * Misc

39714. * Allocate an duk_hobject. * * The allocated object has no allocation for properties; the caller may * want to force a resize if a desired size is known. * * The allocated object has zero reference count and is not reachable. * The caller MUST make the object reachable and increase its reference * count before invoking any operation that might require memory allocation.

39715. * Allocate a new thread. * * Leaves the built-ins array uninitialized. The caller must either * initialize a new global context or share existing built-ins from * another thread.

39716. **comment:** * obj->props is intentionally left as NULL, and duk_hobject_props.c must deal * with this properly. This is intentional: empty objects consume a minimum * amount of memory. Further, an initial allocation might fail and cause * 'obj' to "leak" (require a mark-and-sweep) since it is not reachable yet.
label: code-design

39717. also goes into flags

39718. different memory layout, alloc size, and init

39719. **comment:** XXX: macro? sets both heaphdr and object flags
label: code-design

39720. * Hobject allocation. ** Provides primitive allocation functions for all object types (plain object, * compiled function, native function, thread). The object return is not yet * in "heap allocated" list and has a refcount of zero, so caller must careful.

39721. Zero encoded pointer is required to match NULL

39722. **comment:** unused now
label: code-design

39723. when nothing is running, API calls are in non-strict mode

39724. NULL pointer is required to encode to zero, so memset is enough.

39725. **comment:** UNUSED, intentionally empty
label: code-design

39726. Note: assumes that these string indexes are 8-bit, genstrings.py must ensure that

39727. * Hobject Ecmascript [[Class]].

39728. recheck that the property still exists

39729. * Virtual properties. ** String and buffer indices are virtual and always enumerable, * 'length' is virtual and non-enumerable. Array and arguments * object props have special behavior but are concrete.

39730. 'length' and other virtual properties are not * enumerable, but are included if non-enumerable * properties are requested.

39731. **comment:** XXX: identify enumeration target with an object index (not top of stack)
label: code-design

39732. * Entries part

39733. [... enum]

39734. * Returns non-zero if a key and/or value was enumerated, and: ** [enum] -> [key] (get_value == 0) * [enum] -> [key value] (get_value == 1) ** Returns zero without pushing anything on the stack otherwise.

39735. still reachable

39736. compact; no need to seal because object is internal

39737. [... res]

39738. -> [... enum key enum_target val]

39739. * Get enumerated keys in an Ecmascript array. Matches Object.keys() behavior * described in E5 Section 15.2.3.14.

39740. return 1 to allow callers to tail call

39741. used for array, stack, and entry indices

39742. [enum_target res]

39743. enumerator must have no keys deleted

39744. [... enum_target res trap_result val true]

39745. * Some E5/E5.1 algorithms require that array indices are iterated * in a strictly ascending order. This is the case for e.g. * Array.prototype.forEach() and JSON.stringify() PropertyList * handling. ** To ensure this property for arrays with an array part (and * arbitrary objects too, since e.g. forEach() can be applied * to an array), the caller can request that we sort the keys * here.

39746. string objects must not created without internal value

39747. control props

39748. **comment:** XXX: not sure what the correct semantic details are here, * e.g. handling of missing values (gaps), handling of non-array * trap results, etc. ** For keys, we simply skip non-string keys which seems to be * consistent with how e.g. Object.keys() will process proxy trap * results (ES6, Section 19.1.2.14).
label: code-design

39749. Enumeration keys are checked against the enumeration target (to see * that they still exist). In the proxy enumeration case _Target will * be the proxy, and checking key existence against the proxy is not * required (or sensible, as the keys may be fully virtual).

39750. -> [... enum_target res]

39751. * Array part * * Note: ordering between array and entry part must match 'abandon array' * behavior in duk_hobject_props.c: key order after an array is abandoned * must be the same.

39752. The internal _Target property is kept pointing to the original * enumeration target (the proxy object), so that the enumerator * 'next' operation can read property values if so requested. The * fact that the _Target is a proxy disables key existence check * during enumeration.

39753. **comment:** * Create an internal enumerator object E, which has its keys ordered * to match desired enumeration ordering. Also initialize internal control * properties for enumeration. ** Note: if an array was used to hold enumeration keys instead, an array * scan would be needed to eliminate duplicates found in the prototype chain.
label: code-design

39754. [enum_target res key true]

39755. -> [... enum key val]

39756. There's intentionally no check for * current underlying buffer length.

39757. -- p_insert -- p_curr * v v * | ... | insert | ... | curr

39758. [... enum_target res trap_result]

39759. must match exactly the number of internal properties inserted to enumerator

39760. **comment:** XXX: avoid this at least when enum_target is an Array, it has an * array part, and no ancestor properties were included? Not worth * it for JSON, but maybe worth it for forEach().
label: code-design

39761. Initialize index so that we skip internal control keys.

39762. -> [... enum key enum_target key]

39763. -> [...]

39764. * Proxy object handling

39765. **comment:** Create a temporary enumerator to get the (non-duplicated) key list; * the enumerator state is initialized without being needed, but that * has little impact.
label: code-design

39766. **comment:** * Helper to sort array index keys. The keys are in the enumeration object * entry part, starting from DUK__ENUM_START_INDEX, and the entry part is dense. ** We use insertion sort because it is simple (leading to compact code,) * works nicely in-place, and minimizes operations if data is already sorted * or nearly sorted (which is a very common case here). It also minimizes * the use of element comparisons in general. This is nice because element * comparisons here involve re-parsing the string keys into numbers each * time, which is naturally very expensive. ** Note that the entry part values are all "true", e.g. * * "1" -> true, "3" -> true, "2" -> true * * so it suffices to only work in the key part without exchanging any keys, * simplifying the sort. **
http://en.wikipedia.org/wiki/Insertion_sort * * (Compiles to about 160 bytes now as a stand-alone function.)
label: code-design

39767. [enum_target enum res]

39768. **comment:** * Hobject enumeration support. ** Creates an internal enumeration state object to be used e.g. with for-in * enumeration. The state object contains a snapshot of target object keys * and internal control state for enumeration. Enumerator flags allow caller * to e.g. request internal/non-enumerable properties, and to enumerate only * "own" properties. ** Also creates the result value for e.g. Object.keys() based on the same * internal structure. ** This snapshot-based enumeration approach is used to simplify enumeration: * non-snapshot-based approaches are difficult to reconcile with mutating * the enumeration target, running multiple long-lived enumerators at the * same time, garbage collection details, etc. The downside is that the * enumerator object is memory inefficient especially for iterating arrays.
label: code-design

39769. No need to replace the 'enum_target' value in stack, only the * enum_target reference. This also ensures that the original * enum target is reachable, which keeps the proxy and the proxy * target reachable. We do need to replace the internal _Target.

39770. **comment:** XXX: may need a 'length' filter for forEach()
label: code-design

39771. **comment:** XXX use get tval ptr, more efficient

label: code-design

39772. Needs to be inserted; scan backwards, since we optimize * for the case where elements are nearly in order.
39773. -> [... key val]
39774. Neutered buffer, zero length seems * like good behavior here.
39775. String and buffer enumeration behavior is identical now, * so use shared handler.
39776. [res]
39777. Target must be stored so that we can recheck whether or not * keys still exist when we enumerate. This is not done if the * enumeration result comes from a proxy trap as there is no * real object to check against.
39778. -> [... key]
39779. nargs
39780. [... enum_target res trap_result val]
39781. DUK_USE_ES6_PROXY
39782. keep val_highest
39783. -> [... enum_target res trap_result]
39784. we know these because enum objects are internally created
39785. Copy trap result keys into the enumerator object.
39786. no array part
39787. [... enum_target res handler trap]
39788. -> [... enum_target res handler undefined target]
39789. only 'length'
39790. -> [... enum_target res trap handler target]
39791. ensure never re-entered until rescue cycle complete
39792. This shouldn't happen; call sites should avoid looking up * _Finalizer "through" a Proxy, but ignore if we come here * with a Proxy to avoid finalizer re-entry.
39793. -> [... obj finalizer]
39794. **comment:** XXX: Finalizer lookup should traverse the prototype chain (to allow * inherited finalizers) but should not invoke accessors or proxy object * behavior.
At the moment this lookup will invoke proxy behavior, so * caller must ensure that this function is not called if the target is * a Proxy.
label: code-design

39795. Note: we rely on duk_safe_call() to fix up the stack for the caller, * so we don't need to pop stuff here. There is no return value; * caller determines rescued status based on object refcount.
39796. [... obj]
39797. nargs
39798. **comment:** * Run an duk_hobject finalizer. Used for both reference counting * and mark-and-sweep algorithms. Must never throw an error. * * There is no return value. Any return value or error thrown by * the finalizer is ignored (although errors are debug logged). * * Notes: * * - The thread used for calling the finalizer is the same as the * 'thr' argument. This may need to change later. * * - The finalizer thread 'top' assertions are there because it is * critical that strict stack policy is observed (i.e. no cruft * left on the finalizer stack).
label: code-design

39799. -> [... obj retval/error]
39800. **comment:** XXX: use a NULL error handler for the finalizer call?
label: code-design

39801. **comment:** * Get and call the finalizer. All of this must be wrapped * in a protected call, because even getting the finalizer * may trigger an error (getter may throw one, for instance).
label: code-design

39802. nrets
39803. [... obj finalizer obj heapDestruct] -> [... obj retval]
39804. this also increases refcount by one
39805. duk_safe_call discipline
39806. -> [...]
39807. Note: we ask for one return value from duk_safe_call to get this * error debugging here.
39808. False if the object is NULL or the prototype 'p' is NULL. * In particular, false if both are NULL (don't compare equal).
39809. * Misc support functions
39810. avoid problems if p == h->prototype
39811. * Iterate the bitstream (line diffs) until PC is reached
39812. 0: no change
39813. 1 0 <2 bits>
39814. 1 1 1 <32 bits>
39815. 0
39816. PC is unsigned. If caller does PC arithmetic and gets a negative result, * it will map to a large PC which is out of bounds and causes a zero to be * returned.
39817. Note: pc is unsigned and cannot be negative
39818. **comment:** workaround: max nbits = 24 now
label: code-design

39819. 1 1 0 <8 bits>
39820. **comment:** XXX: now that pc2line is used by the debugger quite heavily in * checked execution, this should be optimized to avoid value stack * and perhaps also implement some form of pc2line caching (see * future work in debugger.rst).
label: code-design

39821. * Use the index in the header to find the right starting point
39822. **comment:** XXX: add proper spare handling to dynamic buffer, to minimize * reallocs; currently there is no spare at all.
label: code-design

39823. end of diff run
39824. DUK_USE_PC2LINE
39825. Generate pc2line data for an instruction sequence, leaving a buffer on stack top.
39826. end of bytecode
39827. compact
39828. **comment:** * Helpers for creating and querying pc2line debug data, which * converts a bytecode program counter to a source line number. * * The run-time pc2line data is bit-packed, and documented in: * * doc/function-objects.rst
label: documentation

39829. be_ctx->offset == length of encoded bitstream
39830. 1 1 1 <32 bits> * Encode in two parts to avoid bitencode 24-bit limitation
39831. valid pc range is [0, length[
39832. Although there are writable virtual properties (e.g. plain buffer * and buffer object number indices), they are handled before we come * here.
39833. should never happen
39834. [key getter this] -> [key retval]
39835. Note: no key on stack
39836. avoid attempt to compact the current object
39837. key is now reachable in the valstack
39838. NULL obj->p is OK
39839. **comment:** XXX: top of stack must contain value, which helper doesn't touch, * rework to use tv_val directly?
label: code-design

39840. all virtual properties are non-configurable and non-writable

39841. not found
39842. resume lookup from target
39843. **comment:** Note: reuse 'curr'
 label: code-design
39844. * Get old and new length
39845. * Note: although arguments object variable mappings are only established * for non-strict functions (and a call to a non-strict function created * the arguments object in question), an inner strict function may be doing * the actual property write. Hence the throw_flag applied here comes from * the property write call.
39846. **comment:** XXX: consolidated algorithm step 15.f -> redundant?
 label: code-design
39847. **comment:** XXX: very basic optimization -> duk_get_prop_stridx_top
 label: code-design
39848. * Check whether we need to abandon an array part (if it exists)
39849. no need to decref, as previous value is 'undefined'
39850. **comment:** * The 'in' operator requires an object as its right hand side, * throwing a TypeError unconditionally if this is not the case. * * However, lightfuncs need to behave like fully fledged objects * here to be maximally transparent, so we need to handle them * here.
 label: code-design
39851. USE_PROP_HASH_PART
39852. Get default hash part size for a certain entry part size.
39853. Must coerce key: if key is an object, it may coerce to e.g. 'length'.
39854. key is 'length', cannot match argument exotic behavior
39855. * First check whether property exists; if not, simple case. This covers * steps 1-4.
39856. String is an own (virtual) property of a lightfunc.
39857. Input value should be on stack top and will be coerced and * popped. Refuse to update an Array's 'length' to a value * outside the 32-bit range. Negative zero is accepted as zero.
39858. If neutered must return 0; offset is zeroed during * neutering.
39859. * Ecmascript compliant [[GetProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * May cause arbitrary side effects and invalidate (most) duk_tval * pointers.
39860. **comment:** XXX: inject test
 label: code-design
39861. setter and/or getter may be NULL
39862. Not possible because array object 'length' is present * from its creation and cannot be deleted, and is thus * caught as an existing property above.
39863. replaces top of stack with new value if necessary
39864. **comment:** prev value must be unused, no decref
 label: code-design
39865. * Update an existing property of the base object.
39866. late lookup, avoid side effects
39867. * Debug logging after adjustment.
39868. Notes: * - only numbered indices are relevant, so arr_idx fast reject is good * (this is valid unless there are more than 4**32-1 arguments). * - since variable lookup has no side effects, this can be skipped if * DUK_GETDESC_FLAG_PUSH_VALUE is not set.
39869. **comment:** XXX: option to pretend property doesn't exist if sanity limit is * hit might be useful.
 label: code-design
39870. update length (curr points to length, and we assume it's still valid)
39871. * Post resize assertions.
39872. flags
39873. Abandon array part, moving array entries into entries part. * This requires a props resize, which is a heavy operation. * We also compact the entries part while we're at it, although * this is not strictly required.
39874. IsGenericDescriptor(desc) == true; this means in practice that 'desc' * only has [[Enumerable]] or [[Configurable]] flag updates, which are * allowed at this point.
39875. No resize has occurred so temp_desc->e_idx is still OK
39876. Lookup 'key' from arguments internal 'map', perform a variable write if mapped. * Used in E5 Section 10.6 algorithm for [[DefineOwnProperty]] (used by [[Put]]). * Assumes stack top contains 'put' value (which is NOT popped).
39877. **comment:** XXX: this is a major target for size optimization
 label: code-design
39878. implicit attributes
39879. [in_val in_val]
39880. leave result on stack top
39881. if fails, e_size will be zero = not an issue, except performance-wise
39882. Note: no recursion issue, we can trust 'map' to behave
39883. non-object base, no offending virtual property
39884. caller ensures; rom objects are never bufferobjects now
39885. * Some change(s) need to be made. Steps 7-11.
39886. For internal use: get non-accessor entry value and attributes
39887. * Write to entry part
39888. * All done, switch properties ('p') allocation to new one.
39889. * Slow delete, but we don't care as we're already in a very slow path. * The delete always succeeds: key has no exotic behavior, property * is configurable, and no resize occurs.
39890. hash probe sequence
39891. must have been, since in array part
39892. array entries are all plain values
39893. **comment:** XXX: the helper currently assumes stack top contains new * 'length' value and the whole calling convention is not very * compatible with what we need.
 label: code-design
39894. Coerce to number before validating pointers etc so that the * number coercions in duk_hbufferobject_validated_write() are * guaranteed to be side effect free and not invalidate the * pointer checks we do here.
39895. Buffer writes are often integers.
39896. XXX: duk_to_int() ensures we'll get 8 lowest bits as * as input is within duk_int_t range (capped outside it).
39897. [obj key value desc]
39898. resume delete to target
39899. stage 2: delete configurable entries above target length
39900. Lightfunc virtual properties are non-configurable, so * reject if match any of them.
39901. success
39902. [] -> [res]
39903. keep key reachable for GC etc; guaranteed not to fail
39904. Special behavior for 'caller' property of (non-bound) function objects * and non-strict Arguments objects: if 'caller' -value- (!) is a strict * mode function, throw a TypeError (E5 Sections 15.3.5.4, 10.6). * Quite interestingly, a non-strict function with no formal arguments * will get an arguments object -without- special 'caller' behavior! * * The E5.1 spec is a bit ambiguous if this special behavior applies when * a bound function is the base value (not the 'caller' value): Section * 15.3.4.5 (describing bind()) states that [[Get]] for bound functions * matches that of Section 15.3.5.4 ([[Get]] for Function instances). * However, Section 13.3.5.4

has "NOTE: Function objects created using * Function.prototype.bind use the default [[Get]] internal method." * The current implementation assumes this means that bound functions * should not have the special [[Get]] behavior. * * The E5.1 spec is also a bit unclear if the TypeError throwing is * applied if the 'caller' value is a strict bound function. The * current implementation will throw even for both strict non-bound * and strict bound functions. * * See test-dev-strict-func-as-caller-prop-value.js for quite extensive * tests. * * This exotic behavior is disabled when the non-standard 'caller' property * is enabled, as it conflicts with the free use of 'caller'.

39905. !DUK_USE_NONSTD_FUNC_CALLER_PROPERTY

39906. assume array part is comprehensive (contains all array indexed elements * or none of them); hence no need to check the entries part here.

39907. index is above internal buffer length -> property is fully normal

39908. write_to_array_part:

39909. E5 Section 15.5.5.1

39910. ignore retval -> [key]

39911. Resolve Proxy targets until Proxy chain ends. No explicit check for * a Proxy loop: user code cannot create such a loop without tweaking * internal properties directly.

39912. * Because buffer values are often looped over, a number fast path * is important.

39913. * E5 Section 15.4.5.1, steps 3.k - 3.n. The order at the end combines * the error case 3.l.iii and the success case 3.m-3.n. * * Note: 'length' is always in entries part, so no array abandon issues for * 'writable' update.

39914. don't push value

39915. Convert a duk_tval number (caller checks) to a 32-bit index. Returns * DUK__NO_ARRAY_INDEX if the number is not whole or not a valid array * index.

39916. * Intern key via the valstack to ensure reachability behaves * properly. We must avoid longjmp's here so use non-checked * primitives. * * Note: duk_check_stack() potentially reallocs the valstack, * invalidating any duk_tval pointers to valstack. Callers * must be careful.

39917. **comment:** never shrinks; auto-adds DUK_VALSTACK_INTERNAL_EXTRA, which is generous

label: code-design

39918. increases key refcount

39919. Reject attempt to change virtual properties: not part of the * standard algorithm, applies currently to e.g. virtual index * properties of buffer objects (which are virtual but writable). * (Cannot "force" modification of a virtual property.)

39920. avoid attempts to add/remove object keys

39921. curr is accessor, desc is data

39922. Set up pointers to the new property area: this is hidden behind a macro * because it is memory layout specific.

39923. * Local defines

39924. E5 Section 15.4.5.1, step 4

39925. Lookup 'key' from arguments internal 'map', perform a variable lookup * if mapped, and leave the result on top of stack (and return non-zero). * Used in E5 Section 10.6 algorithms [[Get]] and [[GetOwnProperty]].

39926. Note: intentionally use approximations to shave a few instructions: * a_used = old_used (accurate: old_used + 1) * a_size = arr_idx (accurate: arr_idx + 1)

39927. [... old_result] -> [... result]

39928. Must avoid duk_pop() in exit path

39929. DUK_USE_FASTINT

39930. no decref needed for a number

39931. Leave 'getter' on stack

39932. [key] -> [undefined] (default value)

39933. **comment:** XXX: There is currently no support for writing buffer object * indexed elements here. Attempt to do so will succeed and * write a concrete property into the buffer object. This should * be fixed at some point but because buffers are a custom feature * anyway, this is relatively unimportant.

label: code-design

39934. **comment:** array part must not contain any non-unused properties, as they would * be configurable and writable.

label: code-design

39935. force_flag

39936. If not found, resume existence check from Function.prototype. * We can just substitute the value in this case; nothing will * need the original base value (as would be the case with e.g. * setters/getters).

39937. * Object internal value * * Returned value is guaranteed to be reachable / incref'd, caller does not need * to incref OR decref. No proxies or accessors are invoked, no prototype walk.

39938. curr value

39939. [key] (coerced)

39940. Note: no allocation pressure, no need to check refcounts etc

39941. This function is only called for objects with array exotic behavior. * The [[DefineOwnProperty]] algorithm for arrays requires that * 'length' can never have a value outside the unsigned 32-bit range, * attempt to write such a value is a RangeError. Here we can thus * assert for this. When Duktape internals go around the official * property write interface (doesn't happen often) this assumption is * easy to accidentally break, so such code must be written carefully. * See test-bi-array-push-maxlen.js.

39942. [key setter this val key] -> [key retval]

39943. * Found existing accessor property (own or inherited). * Call setter with 'this' set to orig, and value as the only argument. * Setter calls are OK even for ROM objects. * * Note: no exotic arguments object behavior, because [[Put]] never * calls [[DefineOwnProperty]] (E5 Section 8.12.5, step 5.b).

39944. guaranteed to find an empty slot

39945. Lookup 'key' from arguments internal 'map', and leave replacement value * on stack top if mapped (and return non-zero). * Used in E5 Section 10.6 algorithm for [[GetOwnProperty]] (used by [[Get]]).

39946. [key trap_result] -> []

39947. **comment:** XXX: investigate whether write protect can be handled above, if we * just update length here while ignoring its protected status

label: code-design

39948. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small.

label: code-design

39949. remove value

39950. * Object.isSealed() and Object.isFrozen() (E5 Sections 15.2.3.11, 15.2.3.13) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: all virtual (non-concrete) properties are currently non-configurable * and non-writable (and there are no accessor virtual properties), so they don't * need to be considered here now.

39951. * Fast path for defining array indexed values without interning the key. * This is used by e.g. code for Array prototype and traceback creation so * must avoid interning.

39952. * Object.getOwnPropertyDescriptor() (E5 Sections 15.2.3.3, 8.10.4) * * This is an actual function call.

39953. * Not found

39954. * Proxy helpers

39955. steps 3.h and 3.i

39956. * Found; post-processing (Function and arguments objects)

39957. [key val]

39958. valstack space that suffices for all local calls, including recursion * of other than Duktape calls (getters etc)

39959. the entries [new_e_next, new_e_size_adjusted[are left uninitialized on purpose (ok, not gc reachable)

39960. cannot be arguments exotic

39961. actually used, non-NULL keys

39962. avoid double coercion

39963. Note: assume array part is comprehensive, so that either * the write goes to the array part, or we've abandoned the * array above (and will not come here).

39964. Magically bound variable cannot be an accessor. However, * there may be an accessor property (or a plain property) in * place with magic behavior removed. This happens e.g. when * a magic property is redefined with defineProperty(). * Cannot assert for "not accessor" here.

39965. * Local prototypes

39966. **comment:** XXX: post-checks (such as no duplicate keys)

label: code-design

39967. **comment:** The handler is looked up with a normal property lookup; it may be an * accessor or the handler object itself may be a proxy object. If the * handler is a proxy, we need to extend the valstack as we make a * recursive proxy check without a function call in between (in fact * there is no limit to the potential recursion here). ** (For sanity, proxy creation rejects another proxy object as either * the handler or the target at the moment so recursive proxy cases * are not realized now.)

label: code-design

39968. Check array density and indicate whether or not the array part should be abandoned.

39969. [key value] or [key undefined]

39970. descriptor type specific checks

39971. avoid multiple computations of flags address; bypasses macros

39972. * Found existing own (non-inherited) plain property. * Do an access control check and update in place.

39973. * Ecmascript compliant [[GetOwnProperty]](P), for internal use only. ** If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero ** Notes: * * - Getting a property descriptor may cause an allocation (and hence * GC) to take place, hence reachability and refcount of all related * values matter. Reallocation of value stack, properties, etc may * invalidate many duk_tval pointers (concretely, those which reside * in memory areas subject to reallocation). However, heap object * pointers are never affected (heap objects have stable pointers). * * - The value of a plain property is always reachable and has a non-zero * reference count. * * - The value of a virtual property is not necessarily reachable from * elsewhere and may have a refcount of zero. Hence we push it onto * the valstack for the caller, which ensures it remains reachable * while it is needed. * * - There are no virtual accessor properties. Hence, all getters and * setters are always related to concretely stored properties, which * ensures that the get/set functions in the resulting descriptor are * reachable and have non-zero refcounts. Should there be virtual * accessor properties later, this would need to change.

39974. **comment:** * XXX: array indices are mostly typed as duk_uint32_t here; duk_uarridx_t * might be more appropriate.

label: code-design

39975. not enumerable

39976. * Allocate and initialize a new entry, resizing the properties allocation * if necessary. Returns entry index (e_idx) or throws an error if alloc fails. ** Sets the key of the entry (increasing the key's refcount), and updates * the hash part if it exists. Caller must set value and flags, and update * the entry value refcount. A decref for the previous value is not necessary.

39977. should not happen

39978. previous value is assumed to be garbage, so don't touch it

39979. * Abandon array failed, need to decref keys already inserted * into the beginning of new_e_k before unwinding valstack.

39980. standard behavior, step 3.f.i

39981. index is above internal string length -> property is fully normal

39982. * Rebuild the hash part always from scratch (guaranteed to finish). ** Any resize of hash part requires rehashing. In addition, by rehashing * get rid of any elements marked deleted (DUK_HASH_DELETED) which is critical * to ensuring the hash part never fills up.

39983. must not be extensible

39984. **comment:** fill new_h with u32 0xff = UNUSED

label: code-design

39985. * Misc helpers

39986. side effects

39987. NULL if tv_obj is primitive

39988. actual update happens once write has been completed without * error below.

39989. IsDataDescriptor(desc) == true

39990. Target object must be checked for a conflicting * non-configurable property.

39991. guaranteed when building arguments

39992. With this check in place fast paths won't need read-only * object checks. This is technically incorrect if there are * setters that cause no writes to ROM objects, but current * built-ins don't have such setters.

39993. * Copy array elements to new array part.

39994. property exists, either 'desc' is empty, or all values * match (SameValue)

39995. we currently assume virtual properties are not configurable (as none of them are)

39996. * Property lookup

39997. Stage 1: find highest preventing non-configurable entry (if any). * When forcing, ignore non-configurability.

39998. Count actually used entry part entries (non-NULL keys).

39999. Note: use 'curr' as a temp propdesc

40000. Fast check for extending array: check whether or not a slow density check is required.

40001. * GETPROP: Ecmascript property read.

40002. * Property not found in prototype chain.

40003. unsigned

40004. **comment:** * In a fast check we assume old_size equals old_used (i.e., existing * array is fully dense). ** Slow check if: * * (new_size - old_size) / old_size > limit * new_size - old_size > limit * old_size * new_size > (1 + limit) * old_size || limit' is 3 bits fixed point * new_size > (1 + (limit' / 8)) * old_size || * 8 * new_size > (8 + limit') * old_size || : 8 * new_size > (8 + limit') * (old_size / 8) * new_size > limit' * (old_size / 8) || limit' = 9 -> max 25% increase * arr_idx + 1 > limit' * (old_size / 8) ** This check doesn't work well for small values, so old_size is rounded * up for the check (and the '+ 1' of arr_idx can be ignored in practice): * * arr_idx > limit' * ((old_size + 7) / 8)

label: code-design

40005. * Property already exists. Steps 5-6 detect whether any changes need * to be made.

40006. Must have array part and no conflicting exotic behaviors. * Doesn't need to have array special behavior, e.g. Arguments * object has array part.

40007. Delete elements required by a smaller length, taking into account * potentially non-configurable elements. Returns non-zero if all * elements could be deleted, and zero if all or some elements could * not be deleted. Also writes final "target length" to 'out_result_len'. * This is the length value that should go into the 'length' property * (must be set by the caller). Never throws an error.

40008. * Compact an object. Minimizes allocation size for objects which are * not likely to be extended. This is useful for internal and non- * extensible objects, but can also be called for non-extensible objects. * May abandon the array part if it is computed to be too sparse. ** This call is relatively expensive, as it needs to scan both the * entries and the array part. * * The call may fail due to allocation error.

40009. XXX: fast path for arrays?

40010. force the property to 'undefined' to create a slot for it

40011. This may trigger mark-and-sweep with arbitrary side effects, * including an attempted resize of the object we're resizing, * executing a finalizer which may add or remove properties of * the object we're resizing etc.

40012. This is a pretty awkward control flow, but we need to recheck the * key coercion here.

40013. [obj key undefined]

40014. **comment:** * XXX: duk_uint_fast32_t should probably be used in many places here.

label: code-design

40015. [obj key desc value get set curr_value]

40016. valstack space allocated especially for proxy lookup which does a * recursive property lookup

40017. DUK_USE_HOBJECT_HASH_PART

40018. **comment:** XXX: pre-checks (such as no duplicate keys)

label: code-design

40019. coercion order matters

40020. XXX: The TypeError is currently not applied to bound * functions because the 'strict' flag is not copied by * bind(). This may or may not be correct, the specification * only refers to the value being a "strict mode Function * object" which is ambiguous.

40021. Undefine local defines
40022. [obj key value desc value]
40023. Array properties have exotic behavior but they are concrete, * so no special handling here. ** Arguments exotic behavior (E5 Section 10.6, [[GetOwnProperty]]) * is only relevant as a post-check implemented below; hence no * check here.
40024. since duk_abandon_array_checked() causes a resize, there should be no gaps in keys
40025. If the value happens to be 0xFFFFFFFF, it's not a valid array index * but will then match DUK_NO_ARRAY_INDEX.
40026. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. ** XXX: this is now an overkill for many fast paths. Rework this * to be faster (although switching to a valstack discipline might * be a better solution overall).
label: code-design
40027. default value
40028. write to entry part
40029. For internal use: get array part value
40030. treat like property not found
40031. [... put_value]
40032. **comment:** * Relocate property allocation, moving properties to the new allocation. ** Includes key compaction, rehashing, and can also optionally abandoning * the array part, 'migrating' array entries into the beginning of the * new entry part. Arguments are not validated here, so e.g. new_h_size * MUST be a valid prime. ** There is no support for in-place reallocation or just compacting keys * without resizing the property allocation. This is intentional to keep * code size minimal. ** The implementation is relatively straightforward, except for the array * abandonment process. Array abandonment requires that new string keys * are interned, which may trigger GC. All keys interned so far must be * reachable for GC at all times; valstack is used for that now. ** Also, a GC triggered during this reallocation process must not interfere * with the object being resized. This is currently controlled by using * heap->mark_and_sweep_base_flags to indicate that no finalizers will be * executed (as they can affect ANY object) and no objects are compacted * (it would suffice to protect this particular object only, though). ** Note: a non-checked variant would be nice but is a bit tricky to * implement for the array abandonment process. It's easy for * everything else. ** Note: because we need to potentially resize the valstack (as part * of abandoning the array part), any tval pointers to the valstack * will become invalid after this call.
label: code-design
40033. **comment:** Note: reuse 'curr' as a temp propdesc
label: code-design
40034. XXX: share final setting code for value and flags? difficult because * refcount code is different. Share entry allocation? But can't allocate * until array index checked.
40035. [key] -> []
40036. if abandon_array, new_a_size must be 0
40037. * Helpers to resize properties allocation on specific needs.
40038. for resizing of array part, use slow path
40039. -> [... trap handler]
40040. Note: proxy handling must happen before key is string coerced.
40041. guaranteed to finish, as hash is never full
40042. Push an arbitrary duk_tval to the stack, coerce it to string, and return * both a duk_hstring pointer and an array index (or DUK_NO_ARRAY_INDEX).
40043. * Migrate array to start of entries if requested. ** Note: from an enumeration perspective the order of entry keys matters. * Array keys should appear wherever they appeared before the array abandon * operation.
40044. Note: unconditional throw
40045. If a setter/getter is missing (undefined), the descriptor must * still have the property present with the value 'undefined'.
40046. Avoid side effects that might disturb curr.e_idx until * we're done editing the slot.
40047. **comment:** actually used
label: code-design
40048. **comment:** XXX: any chance of unifying this with the 'length' key handling?
label: code-design
40049. * Array part
40050. -0 is accepted here as index 0 because ToString(-0) == "0" which is * in canonical form and thus an array index.
40051. see below
40052. hash lookup
40053. **comment:** * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written. ** Note: must re-lookup because calls above (e.g. duk_alloc_entry_checked()) * may realloc and compact properties and hence change e_idx.
label: code-design
40054. * NormalizePropertyDescriptor() related helper. ** Internal helper which validates and normalizes a property descriptor * represented as an Ecmascript object (e.g. argument to defineProperty()). * The output of this conversion is a set of defprop_flags and possibly * some values pushed on the value stack; some subset of property value, * getter, setter. Caller must manage stack top carefully because the * number of values pushed depends on the input property descriptor. ** The original descriptor object must not be altered in the process.
40055. success: leave varname in stack
40056. Note: changing from writable to non-writable is OK
40057. don't touch property attributes or hash part
40058. stage 1 guarantees
40059. current assertion is quite strong: decref's and set to unused
40060. Length in elements: take into account shift, but * intentionally don't check the underlying buffer here.
40061. [key val] -> [key]
40062. * Argument exotic [[Delete]] behavior (E5 Section 10.6) is * a post-check, keeping arguments internal 'map' in sync with * any successful deletes (note that property does not need to * exist for delete to 'succeed'). ** Delete key from 'map'. Since 'map' only contains array index * keys, we can use arr_idx for a fast skip.
40063. [... key trap handler]
40064. * Get old and new length
40065. abandoning requires a props allocation resize and * 'rechecks' the valstack, invalidating any existing * valstack value pointers!
40066. * PUTPROP: Ecmascript property write. ** Unlike Ecmascript primitive which returns nothing, returns 1 to indicate * success and 0 to indicate failure (assuming throw is not set). ** This is an extremely tricky function. Some examples: * * * Currently a decref may trigger a GC, which may compact an object's * property allocation. Consequently, any entry indices (e_idx) will * be potentially invalidated by a decref. * * * Exotic behaviors (strings, arrays, arguments object) require, * among other things: * * - Preprocessing before and postprocessing after an actual property * write. For example, array index write requires pre-checking the * array 'length' property for access control, and may require an * array 'length' update after the actual write has succeeded (but * not if it fails). * * - Deletion of multiple entries, as a result of array 'length' write. * * * Input values are taken as pointers which may point to the valstack. * If valstack is resized because of the put (this may happen at least * when the array part is abandoned), the pointers can be invalidated. * (We currently make a copy of all of the input values to avoid issues.)
40067. * Found existing own or inherited plain property, but original * base is a primitive value.
40068. * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written.
40069. V
40070. ... target is extensible
40071. duk_get_min_grow_e() is always >= 1
40072. * Nice debug log.
40073. * DELPROP: Ecmascript property deletion.
40074. [... varname]
40075. * Ecmascript compliant [[Delete]](P, Throw).
40076. **comment:** XXX: for fastints, could use a variant which assumes a double duk_tval * (and doesn't need to check for fastint again).
label: code-design
40077. **comment:** XXX: the key in 'key in obj' is string coerced before we're called * (which is the required behavior in E5/E5.1/E6) so the key is a string * here already.

label: code-design

40078. note: original, uncoerced base

40079. $l = 0 \Rightarrow$ post-update for array 'length' (used when key is an array index)

40080. only need to guarantee 1 more slot, but allocation growth is in chunks

40081. IsAccessorDescriptor(desc) == true

40082. no wrap assuming h_bufobj->length is valid

40083. avoid issues with relocation

40084. [key] -> []

40085. contains value

40086. Receiver: Proxy object

40087. **comment:** XXX: inline into a prototype walking loop?**label:** code-design

40088. new entry: previous value is garbage; set to undefined to share write_value

40089. nargs

40090. * Found ** Arguments object has exotic post-processing, see E5 Section 10.6, * description of [[GetOwnProperty]] variant for arguments.

40091. happens when hash part dropped

40092. **comment:** Duktape 0.11.0 and prior tried to optimize the resize by not * counting the number of actually used keys prior to the resize. * This worked mostly well but also caused weird leak-like behavior * as in: test-bug-object-prop-alloc-unbounded.js. So, now we count * the keys explicitly to compute the new entry part size.**label:** code-design

40093. no need for decref/incref because value is a number

40094. [key undefined] -> [key]

40095. OK for NULL.

40096. shared exit path now

40097. attempt to change from accessor to data property

40098. Read value pushed on stack.

40099. curr and desc are data

40100. accessor with defined getter

40101. key coercion (unless already coerced above)

40102. covered by comparison

40103. * Must guarantee all actually used array entries will fit into * new entry part. Add one growth step to ensure we don't run out * of space right away.

40104. [key]

40105. * Object.prototype.hasOwnProperty() and Object.prototype.propertyIsEnumerable().

40106. -> [in_val]

40107. **comment:** Note: for an accessor without getter, falling through to * check for "caller" exotic behavior is unnecessary as * "undefined" will never activate the behavior. But it does * no harm, so we'll do it anyway.**label:** code-design

40108. steps 4.a and 4.b are tricky

40109. defaults, E5 Section 8.6.1, Table 7

40110. * For property layout 1, tweak e_size to ensure that the whole entry * part (key + val + flags) is a suitable multiple for alignment * (platform specific). * * Property layout 2 does not require this tweaking and is preferred * on low RAM platforms requiring alignment.

40111. flag for later write

40112. Get minimum array part growth for a certain size.

40113. arrays MUST have a 'length' property

40114. * Note: assuming new_a_size == 0, and that entry part contains * no conflicting keys, refcounts do not need to be adjusted for * the values, as they remain exactly the same. * * The keys, however, need to be interned, incref'd, and be * reachable for GC. Any intern attempt may trigger a GC and * claim any non-reachable strings, so every key must be reachable * at all times. * * A longjmp must not occur here, as the new_p allocation would * be freed without these keys being decref'd, hence the messy * decref handling if intern fails.

40115. * Helpers for managing property storage size

40116. * Arguments handling helpers (argument map mainly). * * An arguments object has exotic behavior for some numeric indices. * Accesses may translate to identifier operations which may have * arbitrary side effects (potentially invalidating any duk_tval * pointers).

40117. NULL if not an object

40118. Catches >0x100000000 and negative values.

40119. currently required

40120. **comment:** XXX: Extensibility check for target uses IsExtensible(). If we * implemented the isExtensible trap and didn't reject proxies as * proxy targets, it should be respected here.**label:** code-design

40121. Currently there are no deletable virtual properties, but * with force_flag we might attempt to delete one.

40122. * Start doing property attributes updates. Steps 12-13. * * Start by computing new attribute flags without writing yet. * Property type conversion is done above if necessary.

40123. **comment:** Note: array entries are always writable, so the writability check * above is pointless for them. The check could be avoided with some * refactoring but is probably not worth it.**label:** code-design40124. **comment:** XXX: for array instances we could take a shortcut here and assume * Array.prototype doesn't contain an array index property.**label:** code-design

40125. retval indicates delete failed

40126. E5 Section 15.4.5.1, steps 4.e.i - 4.e.ii

40127. **comment:** XXX: handling for array part missing now; this doesn't affect * compliance but causes array entry writes using defineProperty() * to always abandon array part.**label:** code-design

40128. Caller doesn't need to check exotic proxy behavior (but does so for * some fast paths).

40129. fall through

40130. * HASPROP: EcmaScript property existence check ("in" operator). * * Interestingly, the 'in' operator does not do any coercion of * the target object.

40131. **comment:** XXX: write protect after flag? -> any chance of handling it here?**label:** code-design

40132. If neutered must return 0; length is zeroed during * neutering.

40133. array case is handled comprehensively above

40134. Note: array part values are [[Writable]], [[Enumerable]], * and [[Configurable]] which matches the required attributes * here.

40135. [key setter this val] -> [key retval]

40136. stage 3: update length (done by caller), decide return code

40137. * Not found as concrete or virtual

40138. minimum new length is highest_arr_idx + 1

40139. property is configurable and

40140. consistency requires this

40141. * Coercion and fast path processing

40142. **comment:** XXX: Refactor key coercion so that it's only called once. It can't * be trivially lifted here because the object must be type checked * first.**label:** code-design

40143. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside object 'a_size'.

40144. The coercion order must match the ToPropertyDescriptor() algorithm * so that side effects in coercion happen in the correct order. * (This order also happens to be compatible with duk_def_prop(), * although it doesn't matter in practice.)

40145. relevant array index is non-configurable, blocks write

40146. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.

40147. **comment:** Note: we could return the hash index here too, but it's not * needed right now.
label: code-design

40148. P

40149. * Internal helper for defining an accessor property, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes. This is called * very rarely, so the implementation first sets a value to undefined * and then changes the entry to an accessor (this is to save code space).

40150. index/length check guarantees

40151. **comment:** XXX: At the moment Duktape accesses internal keys like _Finalizer using a * normal property set/get which would allow a proxy handler to interfere with * such behavior and to get access to internal key strings. This is not a problem * as such because internal key strings can be created in other ways too (e.g. * through buffers). The best fix is to change Duktape internal lookups to * skip proxy behavior. Until that, internal property accesses bypass the * proxy and are applied to the target (as if the handler did not exist). * This has some side effects, see test-bi-proxy-internal-keys.js.
label: code-design

40152. new value

40153. actually used, non-NULL entries

40154. Do decrefs only with safe pointers to avoid side effects * disturbing e_idx.

40155. shared checks for all descriptor types

40156. strict flag for putvar comes from our caller (currently: fixed)

40157. * Coercion and fast path processing.

40158. Not strictly necessary because if key == NULL, flag MUST be ignored.

40159. caller ensures

40160. traits are separate; in particular, arguments not an array

40161. 0 = don't push current value

40162. Note: new_e_next matches pushed temp key count, and nothing can * fail above between the push and this point.

40163. fill new entries with -unused- (required, gc reachable)

40164. duk__get_min_grow_a() is always >= 1

40165. [key getter this key] -> [key retval]

40166. * New length not lower than old length => no changes needed * (not even array allocation).

40167. **comment:** * Write to 'length' of an array is a very complex case * handled in a helper which updates both the array elements * and writes the new 'length'. The write may result in an * unconditional RangeError or a partial write (indicated * by a return code). * * Note: the helper has an unnecessary writability check * for 'length', we already know it is writable.
label: code-design

40168. important for callers

40169. * Arguments exotic behavior not possible for new properties: all * magically bound properties are initially present in the arguments * object, and if they are deleted, the binding is also removed from * parameter map.

40170. clear array part flag only after switching

40171. * Writability check

40172. tv1 points to value storage

40173. caller must have decref'd values above new_a_size (if that is necessary)

40174. * XXX: shrink array allocation or entries compaction here?

40175. not found in 'curr', next in prototype chain; impose max depth

40176. FOUND

40177. Buffer has virtual properties similar to string, but indexed values * are numbers, not 1-byte buffers/strings which would perform badly.

40178. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small; some overlap with string * handling.
label: code-design

40179. entry part must not contain any configurable properties, or * writable properties (if is_frozen).

40180. t >= 0 always true, unsigned

40181. Storing the entry top is cheaper here to ensure stack is correct at exit, * as there are several paths out.

40182. Lookup 'key' from arguments internal 'map', delete mapping if found. * Used in E5 Section 10.6 algorithm for [[Delete]]. Note that the * variable/argument itself (where the map points) is not deleted.

40183. Set property slot to an empty state. Careful not to invoke * any side effects while using desc.e_idx so that it doesn't * get invalidated by a finalizer mutating our object.

40184. target

40185. fall thru

40186. [key setter this val]

40187. **comment:** Note: we can reuse 'desc' here
label: code-design

40188. * Array exotic behaviors can be implemented at this point. The local variables * are essentially a 'value copy' of the input descriptor (Desc), which is modified * by the Array [[DefineOwnProperty]] (E5 Section 15.4.5.1).

40189. data property or accessor without getter

40190. **comment:** XXX: fastint
label: code-design

40191. required to guarantee success of rehashing, * intentionally use unadjusted new_e_size

40192. Regardless of whether property is found in entry or array part, * it may have arguments exotic behavior (array indices may reside * in entry part for abandoned / non-existent array parts).

40193. Careful with wrapping (left shifting idx would be unsafe).

40194. -> [... obj key trap handler]

40195. [...val] -> [...]

40196. currently used -> new size

40197. mask out flags not actually stored

40198. varenv remains reachable through 'obj'

40199. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.

40200. * New length is smaller than old length, need to delete properties above * the new length. * * If array part exists, this is straightforward: array entries cannot * be non-configurable so this is guaranteed to work. * * If array part does not exist, array-indexed values are scattered * in the entry part, and some may not be configurable (preventing length * from becoming lower than their index + 1). To handle the algorithm * in E5 Section 15.4.5.1, step 1 correctly, we scan the entire property * set twice.

40201. **comment:** XXX: optimize this filling behavior later
label: code-design

40202. **comment:** * Property count check. This is the only point where we ensure that * we don't get more (allocated) property space that we can handle. * There aren't hard limits as such, but some algorithms fail (e.g. * finding next higher prime, selecting hash part size) if we get too * close to the 4G property limit. * * Since this works based on allocation size (not actually used size), * the limit is a bit approximate but good enough in practice.
label: code-design

40203. All lightfunc own properties are non-writable and the lightfunc * is considered non-extensible. However, the write may be captured * by an inherited setter which means we can't stop the lookup here.

40204. Convert a duk_tval fastint (caller checks) to a 32-bit index.
40205. [key result] -> [result]
40206. * Standard algorithm succeeded without errors, check for exotic post-behaviors. ** Arguments exotic behavior in E5 Section 10.6 occurs after the standard * [[DefineOwnProperty]] has completed successfully. ** Array exotic behavior in E5 Section 15.4.5.1 is implemented partly * prior to the default [[DefineOwnProperty]], but: * - for an array index key (e.g. "10") the final 'length' update occurs here * - for 'length' key the element deletion and 'length' update occurs here
40207. map is reachable through obj
40208. 0 if no used entries
40209. * All defined array-indexed properties are in the array part * (we assume the array part is comprehensive), and all array * entries are writable, configurable, and enumerable. Thus, * nothing can prevent array entries from being deleted.
40210. resume check from proxy target
40211. **comment:** Count actually used array part entries and array minimum size. * NOTE: 'out_min_size' can be computed much faster by starting from the * end and breaking out early when finding first used entry, but this is * not needed now.
label: code-design
40212. arguments MUST have an initialized lexical environment reference
40213. if new_p == NULL, all of these pointers are NULL
40214. If index is not valid, idx will be DUK_NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside bufferobject length.
40215. automatic length update
40216. always grow the array, no sparse / abandon support here
40217. **comment:** remove old value
label: code-design
40218. **comment:** NULL always indicates alloc failure because * new_alloc_size > 0.
label: code-design
40219. [...] -> [...]
40220. always in entry part, no need to look up parents etc
40221. * Copy keys and values in the entry part (compacting them at the same time).
40222. E5 Section 15.4.5.1, step 3, steps a - i are implemented here, j - n at the end
40223. no need to compact since we already did that in duk_abandon_array_checked() * (regardless of whether an array part existed or not).
40224. XXX: C recursion limit if proxies are allowed as handler/target values
40225. **comment:** XXX: assertion that entries >= old_len are already unused
label: code-design
40226. Arrays never have other exotic behaviors.
40227. * Create a new property in the original object. ** Exotic properties need to be reconsidered here from a write * perspective (not just property attributes perspective). * However, the property does not exist in the object already, * so this limits the kind of exotic properties that apply.
40228. Initial value for highest_idx is -1 coerced to unsigned. This * is a bit odd, but (highest_idx + 1) will then wrap to 0 below * for out_min_size as intended.
40229. * Write to array part? ** Note: array abandoning requires a property resize which uses * 'rechecks' valstack for temporaries and may cause any existing * valstack pointers to be invalidated. To protect against this, * tv_obj, tv_key, and tv_val are copies of the original inputs.
40230. copy existing entries as is
40231. value from hook
40232. **comment:** * Shallow fast path checks for accessing array elements with numeric * indices. The goal is to try to avoid coercing an array index to an * (interned) string for the most common lookups, in particular, for * standard Array objects. ** Interning is avoided but only for a very narrow set of cases: * - Object has array part, index is within array allocation, and * value is not unused (= key exists) * - Object has no interfering exotic behavior (e.g. arguments or * string object exotic behaviors interfere, array exotic * behavior does not). ** Current shortcoming: if key does not exist (even if it is within * the array allocation range) a slow path lookup with interning is * always required. This can probably be fixed so that there is a * quick fast path for non-existent elements as well, at least for * standard Array objects.
label: code-design
40233. [... put_value varname]
40234. number
40235. Get minimum entry part growth for a certain size.
40236. if accessor without getter, return value is undefined
40237. undefined is accepted
40238. [obj key desc value get set curr_value varname]
40239. key must not already exist in entry part
40240. * Note: currently no fast path for array index writes. * They won't be possible anyway as strings are immutable.
40241. * Object.seal() and Object.freeze() (E5 Sections 15.2.3.8 and 15.2.3.9) ** Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. ** Note: virtual (non-concrete) properties which are non-configurable but * writable would pose some problems, but such properties do not currently * exist (all virtual properties are non-configurable and non-writable). * If they did exist, the non-configurability does NOT prevent them from * becoming non-writable. However, this change should be recorded somehow * so that it would turn up (e.g. when getting the property descriptor), * requiring some additional flags in the object.
40242. **comment:** * HASPROP variant used internally. ** This primitive must never throw an error, callers rely on this. * In particular, don't throw an error for prototype loops; instead, * pretend like the property doesn't exist if a prototype sanity limit * is reached. ** Does not implement proxy behavior: if applied to a proxy object, * returns key existence on the proxy object itself.
label: code-design
40243. **comment:** The fast path for array property put is not fully compliant: * If one places conflicting number-indexed properties into * Array.prototype (for example, a non-writable Array.prototype[7]) * the fast path will incorrectly ignore them. ** This fast path could be made compliant by falling through * to the slow path if the previous value was UNUSED. This would * also remove the need to check for extensibility. Right now a * non-extensible array is slower than an extensible one as far * as writes are concerned. ** The fast path behavior is documented in more detail here: * tests/ecmascript/test-misc-array-fast-write.js
label: code-design
40244. * Because buffer values may be looped over and read/written * from, an array index fast path is important.
40245. Note: only numbered indices are relevant, so arr_idx fast reject * is good (this is valid unless there are more than 4**32-1 arguments).
40246. entry allocation updates hash part and increases the key * refcount; may need a props allocation resize but doesn't * 'recheck' the valstack.
40247. * Actual Object.defineProperty() default algorithm.
40248. **comment:** Linear scan: more likely because most objects are small. * This is an important fast path. ** XXX: this might be worth inlining for property lookups.
label: code-design
40249. [...] value? getter? setter?]
40250. may become sparse...
40251. Get Proxy target object. If the argument is not a Proxy, return it as is. * If a Proxy is revoked, an error is thrown.
40252. **comment:** * Object.defineProperty() related helper (E5 Section 15.2.3.6) ** Inlines all [[DefineOwnProperty]] exotic behaviors. ** Note: Ecmascript compliant [[DefineOwnProperty]](P, Desc, Throw) is not * implemented directly, but Object.defineProperty() serves its purpose. * We don't need the [[DefineOwnProperty]] internally and we don't have a * property descriptor with 'missing values' so it's easier to avoid it * entirely. * * Note: this is only called for actual objects, not primitive values. * This must support virtual properties for full objects (e.g. Strings) * but not for plain values (e.g. strings). Lighfuncs, even though * primitive in a sense, are treated like objects and accepted as target * values.
label: code-design
40253. **comment:** * Array needs to grow, but we don't want it becoming too sparse. * If it were to become sparse, abandon array part, moving all * array entries into the entries part (for good). ** Since we don't keep track of actual density (used vs. size) of * the array part, we need to estimate somehow. The check is made * in two parts: * * - Check whether the resize need is small compared to the * current size (relatively); if so, resize without further * checking (essentially we assume that

the original part is * "dense" so that the result would be dense enough). * * - Otherwise, compute the resize using an actual density * measurement based on counting the used array entries.

label: code-design

40254. * Compute new alloc size and alloc new area. * * The new area is allocated as a dynamic buffer and placed into the * valstack for reachability. The actual buffer is then detached at * the end. * * Note: heap_mark_and_sweep_base_flags are altered here to ensure * no-one touches this object while we're resizing and rehashing it. * The flags must be reset on every exit path after it. Finalizers * and compaction is prevented currently for all objects while it * would be enough to restrict it only for the current object.

40255. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. * * XXX: this is an overkill for some paths, so optimize this later * (or maybe switch to a stack arguments model entirely).

label: code-design

40256. **comment:** re-lookup to update curr.flags * XXX: would be faster to update directly

label: code-design

40257. This should not happen because DUK_TAG_OBJECT case checks * for this already, but check just in case.

40258. * Hobject property set/get functionality. * * This is very central functionality for size, performance, and compliance. * It is also rather intricate; see hobject-algorithms.rst for discussion on * the algorithms and memory-management.rst for discussion on refcounts and * side effect issues. * * Notes: * * - It might be tempting to assert "refcount nonzero" for objects * being operated on, but that's not always correct: objects with * a zero refcount may be operated on by the refcount implementation * (finalization) for instance. Hence, no refcount assertions are made. * * - Many operations (memory allocation, identifier operations, etc) * may cause arbitrary side effects (e.g. through GC and finalization). * These side effects may invalidate duk_tval pointers which point to * areas subject to reallocation (like value stack). Heap objects * themselves have stable pointers. Holding heap object pointers or * duk_tval copies is not problematic with respect to side effects; * care must be taken when holding and using argument duk_tval pointers. * * - If a finalizer is executed, it may operate on the same object * we're currently dealing with. For instance, the finalizer might * delete a certain property which has already been looked up and * confirmed to exist. Ideally finalizers would be disabled if GC * happens during property access. At the moment property table realloc * disables finalizers, and all DECREFs may cause arbitrary changes so * handle DECREF carefully. * * - The order of operations for a DECREF matters. When DECREF is executed, * the entire object graph must be consistent; note that a refzero may * lead to a mark-and-sweep through a refcount finalizer.

40259. **comment:** * Entries part is a bit more complex

label: code-design

40260. remove key

40261. array 'length' is always a number, as we coerce it

40262. = IsAccessorDescriptor(Desc)

40263. Reject attempt to change a read-only object.

40264. idx_value may be < 0 (no value), set and get may be NULL

40265. Is whole and within 32 bit range. If the value happens to be 0xFFFFFFFF, * it's not a valid array index but will then match DUK_NO_ARRAY_INDEX.

40266. * Find an existing key from entry part either by linear scan or by * using the hash index (if it exists). * * Sets entry index (and possibly the hash index) to output variables, * which allows the caller to update the entry and hash entries in-place. * If entry is not found, both values are set to -1. If entry is found * but there is no hash part, h_idx is set to -1.

40267. -> [... handler trap]

40268. DUK_USE_ES6_PROXY

40269. not reachable

40270. marker values for hash part

40271. 'varname' is in stack in this else branch, leaving an unbalanced stack below, * but this doesn't matter now.

40272. no need for 'caller' post-check, because 'key' must be an array index

40273. remove in_val

40274. Grow entry part allocation for one additional entry.

40275. * Internal helper to define a property with specific flags, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes unless caller requests * that value only be updated if it doesn't already exists. * * Does not support: * - virtual properties (error if write attempted) * - getter/setter properties (error if write attempted) * - non-default (!= WEC) attributes for array entries (error if attempted) * - array abandoning: if array part exists, it is always extended * - array 'length' updating * * Stack: [... in_val] -> [] * * Used for e.g. built-in initialization and environment record * operations.

40276. for zero size, don't push anything on valstack

40277. [key result]

40278. remaining actual steps are carried out if standard DefineOwnProperty succeeds

40279. curr is accessor -> cannot be in array part

40280. E5 Section 15.5.5.2

40281. equality not actually possible

40282. This is not strictly necessary, but avoids compiler warnings; e.g. * gcc won't reliably detect that no uninitialized data is read below.

40283. Avoid zero copy with an invalid pointer. If obj->p is NULL, * the 'new_a' pointer will be invalid which is not allowed even * when copy size is zero.

40284. ignore result

40285. cannot be e.g. arguments exotic, since exotic 'traits' are mutually exclusive

40286. * Arguments objects have exotic [[DefineOwnProperty]] which updates * the internal 'map' of arguments for writes to currently mapped * arguments. More concretely, writes to mapped arguments generate * a write to a bound variable. * * The [[Put]] algorithm invokes [[DefineOwnProperty]] for existing * data properties and new properties, but not for existing accessors. * Hence, in E5 Section 10.6 ([[DefinedOwnProperty]] algorithm), we * have a Desc with 'Value' (and possibly other properties too), and * we end up in step 5.b.i.

40287. * Helper: handle Array object 'length' write which automatically * deletes properties, see E5 Section 15.4.5.1, step 3. This is * quite tricky to get right. * * Used by duk_hobject_putprop().

40288. **comment:** XXX: is valstack top best place for argument?

label: code-design

40289. XXX: sufficient to check 'strict', assert for 'is function'

40290. Leave 'value' on stack

40291. return value

40292. [...] val key] -> [...] key val]

40293. a dummy undefined value is pushed to make valstack * behavior uniform for caller

40294. * Found existing inherited plain property. * Do an access control check, and if OK, write * new property to 'orig'.

40295. result: hash_prime(floor(1.2 * e_size))

40296. stack contains value (if requested), 'out_desc' is set

40297. curr and desc are accessors

40298. Must be an object, otherwise TypeError (E5.1 Section 8.10.5, step 1).

40299. **comment:** Downgrade checks are not made everywhere, so 'length' is not always * a fastint (it is a number though). This can be removed once length * is always guaranteed to be a fastint.

label: code-design

40300. * Entries part

40301. second incref for the entry reference

40302. * Not found as a concrete property, check whether a String object * virtual property matches.

40303. out_desc is left untouched (possibly garbage), caller must use return * value to determine whether out_desc can be looked up

40304. **comment:** * Array abandon check; abandon if: * * new_used / new_size < limit * new_used < limit * new_size || limit is 3 bits fixed point * new_used < limit / 8 * new_size || *8 * 8*new_used < limit * new_size || :8 * new_used < limit * (new_size / 8) * * Here, new_used = a_used, new_size = a_size. * * Note: some callers use approximate values for a_used and/or a_size * (e.g. dropping a '+1' term). This doesn't affect the usefulness * of the check, but may confuse debugging.

label: code-design

40305. prev value can be garbage, no decref

40306. * Check whether the property already exists in the prototype chain. * Note that the actual write goes into the original base object * (except if an accessor property captures the write).

40307. step 3.e: replace 'Desc.[[Value]]'

40308. **comment:** NOTE: lightfuncs are coerced to full functions because * lightfuncs don't fit into a property value slot. This * has some side effects, see test-dev-lightfunc-accessor.js.
label: code-design

40309. **comment:** * Object.preventExtensions() and Object.isExtensible() (E5 Sections 15.2.3.10, 15.2.3.13) ** Not needed, implemented by macros DUK_HOBJECT_{HAS,CLEAR,SET}_EXTENSIBLE * and the Object built-in bindings.
label: code-design

40310. [str] -> [substr]

40311. value from target

40312. * New length lower than old length => delete elements, then * update length. ** Note: even though a bunch of elements have been deleted, the 'desc' is * still valid as properties haven't been resized (and entries compacted).

40313. Note: this order matters (final value before deleting map entry must be done)

40314. unconditional

40315. Object.defineProperty() calls [[DefineOwnProperty]] with Throw=true

40316. * Fast path for bufferobject getprop/putprop

40317. XXX: shrink array allocation or entries compaction here?

40318. Value is required to be a number in the fast path so there * are no side effects in write coercion.

40319. * Possible pending array length update, which must only be done * if the actual entry write succeeded.

40320. 0 = no update

40321. Grow array part for a new highest array index.

40322. **comment:** XXX: awkward helpers
label: code-design

40323. * Pre resize assertions.

40324. curr is data, desc is accessor

40325. stack prepended for func call: [... trap handler]

40326. remove hash entry (no decref)

40327. resume write to target

40328. For internal use: get non-accessor entry value

40329. * Internal helpers for managing object 'length'

40330. All the flags fit in 16 bits, so will fit into duk_bool_t.

40331. throw

40332. **comment:** XXX: macro checks for array index flag, which is unnecessary here
label: code-design

40333. [... value this_binding]

40334. Note: actual update happens once write has been completed * without error below. The write should always succeed * from a specification viewpoint, but we may e.g. run out * of memory. It's safer in this order.

40335. Note: 'curr' refers to 'length' propdesc

40336. **comment:** * Abandon array part because all properties must become non-configurable. * Note that this is now done regardless of whether this is always the case * (skips check, but performance problem if caller would do this many times * for the same object; not likely).
label: code-design

40337. varname is still reachable

40338. get varenv for varname (callee's declarative lexical environment)

40339. guaranteed to finish

40340. Leave 'setter' on stack

40341. If 16-bit hash is in use, stuff it into duk_heapheader_string flags.

40342. string is 'eval' or 'arguments'

40343. Slightly smaller code without explicit flag, but explicit flag * is very useful when 'clen' is dropped.

40344. computed live

40345. **comment:** XXX: could add flags for "is valid CESU-8" (EcmaScript compatible strings), * "is valid UTF-8", "is valid extended UTF-8" (internal strings are not, * regexp bytecode is), and "contains non-BMP characters". These are not * needed right now.
label: code-design

40346. * For an external string, the NUL-terminated string data is stored * externally. The user must guarantee that data behind this pointer * doesn't change while it's used.

40347. get array index related to string (or return DUK_HSTRING_NO_ARRAY_INDEX); * avoids helper call if string has no array index value.

40348. * Prototypes

40349. string hash

40350. The external string struct is defined even when the feature is inactive.

40351. string is a reserved word (strict)

40352. slower but more compact variant

40353. marker value; in E5 2^32-1 is not a valid array index (2^32-2 is highest valid)

40354. Impose a maximum string length for now. Restricted artificially to * ensure adding a heap header length won't overflow size_t. The limit * should be synchronized with DUK_HBUFFER_MAX_BYTELEN. ** E5.1 makes provisions to support strings longer than 4G characters. * This limit should be eliminated on 64-bit platforms (and increased * closer to maximum support on 32-bit platforms).

40355. string is a reserved word (non-strict)

40356. string is ASCII, clen == blen

40357. * String value of 'blen+1' bytes follows (+1 for NUL termination * convenience for C API). No alignment needs to be guaranteed * for strings, but fields above should guarantee alignment-by-4 * (but not alignment-by-8).

40358. **comment:** should never be called
label: code-design

40359. placed in duk_heapheader_string

40360. length in codepoints (must be E5 compatible)

40361. Note: we could try to stuff a partial hash (e.g. 16 bits) into the * shared heap header. Good hashing needs more hash bits though.

40362. * Heap string representation. ** Strings are byte sequences ordinarily stored in extended UTF-8 format, * allowing values larger than the official UTF-8 range (used internally) * and also allowing UTF-8 encoding of surrogate pairs (CESU-8 format). * Strings may also be invalid UTF-8 altogether which is the case e.g. with * strings used as internal property names and raw buffers converted to * strings. In such cases the 'clen' field contains an inaccurate value. * * EcmaScript requires support for 32-bit long strings. However, since each * 16-bit codepoint can take 3 bytes in CESU-8, this representation can only * support about 1.4G codepoint long strings in extreme cases. This is not * really a practical issue.

40363. string is a valid array index

40364. string is internal

40365. string data is external (duk_hstring_external)

40366. **comment:** Smaller heapheader than for other objects, because strings are held * in string intern table which requires no link pointers. Much of * the 32-bit flags field is unused by flags, so we can stuff a 16-bit * field in there.
label: code-design

40367. length in bytes (not counting NUL term)

40368. DUK_HSTRING_H_INCLUDED

40369. * Misc

40370. Caller must check character offset to be inside the string.
40371. unsigned
40372. This may throw an error though not for valid E5 strings.
40373. * Misc support functions
40374. !DUK_USE_HSTRING_CLEN
40375. **comment:** Most practical strings will go here.
 label: code-design
40376. **comment:** Convenience copies from heap/vm for faster access.
 label: code-design
40377. allocation size
40378. idx_bottom and idx_retval are only used for book-keeping of * Ecmascript-initiated calls, to allow returning to an Ecmascript * function properly. They are duk_size_t to match the convention * that value stack sizes are duk_size_t and local frame indices * are duk_idx_t.
40379. next to use, highest used is top - 1
40380. flags field: LLLLLLFT, L = label (24 bits), F = flags (4 bits), T = type (4 bits)
40381. current lexical environment (may be NULL if delayed)
40382. Current compiler state (if any), used for augmenting SyntaxErrors.
40383. No field needed when strings are in ROM.
40384. * Catcher defines
40385. end of valstack allocation (exclusive)
40386. function executes as a constructor (called via "new")
40387. cached: valstack_end - valstack (in entries, not bytes)
40388. internal extra elements assumed on function entry, * always added to user-defined 'extra' for e.g. the * duk_check_stack() call.
40389. **comment:** Builtin-objects; may or may not be shared with other threads, * threads existing in different "compartments" will have different * built-ins. Must be stored on a per-thread basis because there * is no intermediate structure for a thread group / compartment. * This takes quite a lot of space, currently 43x4 = 172 bytes on * 32-bit platforms. * * In some cases the builtins array could be ROM based, but it's * sometimes edited (e.g. for sandboxing) so it's better to keep * this array in RAM.
 label: code-design
40390. catch part will catch
40391. Note: DUK_VALSTACK_INITIAL_SIZE must be >= DUK_VALSTACK_API_ENTRY_MINIMUM * + DUK_VALSTACK_INTERNAL_EXTRA so that the initial stack conforms to spare * requirements.
40392. roughly 512 bytes
40393. DUK_USE_ROM_STRINGS
40394. thread has yielded
40395. roughly 256 bytes
40396. who resumed us (if any)
40397. callstack index of related activation
40398. * Flags for __FILE__ / __LINE__ registered into tracedata
40399. roughly 1 kiB
40400. **comment:** Pointer to bytecode executor's 'curr_pc' variable. Used to copy * the current PC back into the topmost activation when activation * state is about to change (or "syncing" is otherwise needed). This * is rather awkward but important for performance, see execution.rst.
 label: code-design
40401. activation prevents yield (native call or "new")
40402. function executes in strict mode
40403. Note: it's nice if size is 2^N (at least for 32-bit platforms).
40404. Sanity limits for stack sizes.
40405. borrowed: full duk_tval for function being executed; for lightfuncs
40406. indirect allocs
40407. * Assert context is valid: non-NULL pointer, fields look sane. * * This is used by public API call entrypoints to catch invalid 'ctx' pointers * as early as possible; invalid 'ctx' pointers cause very odd and difficult to * diagnose behavior so it's worth checking even when the check is not 100%.
40408. number of activation records in callstack preventing a yield
40409. activation has active breakpoint(s)
40410. Call stack. [0,callstack_top] is GC reachable.
40411. thread not currently running
40412. roughly 1.0 kiB -> but rounds up to DUK_VALSTACK_GROW_STEP in practice
40413. roughly 128 bytes
40414. (reference is valid as long activation exists)
40415. don't report __FILE__ / __LINE__ as fileName/lineNumber
40416. DUK_HTHREAD_H_INCLUDED
40417. borrowed reference to catch variable name (or NULL if none)
40418. roughly 0.5 kiB
40419. bottom of current frame
40420. Note: it's nice if size is 2^N (not 4x4 = 16 bytes on 32 bit)
40421. Return value when returning to this activation (points to caller * reg, not callee reg); index is absolute (only set if activation is * not topmost). * * Note: idx_bottom is always set, while idx_retval is only applicable * for activations below the topmost one. Currently idx_retval for * the topmost activation is considered garbage (and it not initialized * on entry or cleared on return; may contain previous or garbage * values).
40422. thread currently running (only one at a time)
40423. top of current frame (exclusive)
40424. number of elements guaranteed to be user accessible * (in addition to call arguments) on Duktape/C function entry.
40425. * Prototypes
40426. activation has tail called one or more times
40427. Current strictness flag: affects API calls.
40428. Backpointers.
40429. roughly 2 kiB
40430. values for the state field
40431. needed for stepping
40432. Yield/resume book-keeping.
40433. **comment:** XXX: Valstack, callstack, and catchstack are currently assumed * to have non-NULL pointers. Relaxing this would not lead to big * benefits (except perhaps for terminated threads).
 label: code-design
40434. finally part will catch
40435. idx_base and idx_base+1 get completion value and type
40436. Executor interrupt default interval when nothing else requires a * smaller value. The default interval must be small enough to allow * for reasonable execution timeout checking but large enough to keep * impact on execution performance low.
40437. Thread state.
40438. request to create catch binding
40439. nop
40440. start of valstack allocation
40441. next instruction to execute (points to 'func' bytecode, stable pointer), NULL for native calls

40442. * Struct defines
40443. Value stack: these are expressed as pointers for faster stack manipulation. * [valstack, valstack_top[is GC-reachable, [valstack_top, valstack_end[is * not GC-reachable but kept initialized as 'undefined'.
40444. * Heap thread object representation. ** duk_hthread is also the 'context' (duk_context) for exposed APIs * which mostly operate on the topmost frame of the value stack.
40445. activation is a direct eval call
40446. Shared object part
40447. Interrupt counter for triggering a slow path check for execution * timeout, debugger interaction such as breakpoints, etc. The value * is valid for the current running thread, and both the init and * counter values are copied whenever a thread switch occurs. It's * important for the counter to be conveniently accessible for the * bytecode executor inner loop for performance reasons.
40448. start value for current countdown
40449. type and control flags, label number
40450. borrowed: function being executed; for bound function calls, this is the final, real function, NULL for lightfuncs
40451. current variable environment (may be NULL if delayed)
40452. Catch stack. [0, catchstack_top[is GC reachable.
40453. * Stack constants
40454. **comment:** XXX: for a memory-code tradeoff, remove 'func' and make it's access either a function * or a macro. This would make the activation 32 bytes long on 32-bit platforms again.
label: code-design
40455. * Activation defines
40456. Previous value of 'func' caller, restored when unwound. Only in use * when 'func' is non-strict.
40457. thread resumed another thread (active but not running)
40458. roughly 64 bytes
40459. Bottom of valstack for this activation, used to reset * valstack_bottom on return; index is absolute. Note: * idx_top not needed because top is set to 'nregs' always * when returning to an Ecmascript activation.
40460. catch or with binding is currently active
40461. resume execution from pc_base or pc_base+1 (points to 'func' bytecode, stable pointer)
40462. countdown state
40463. * Thread defines
40464. thread has terminated
40465. **comment:** Current 'this' binding is the value just below idx_bottom. * Previously, 'this' binding was handled with an index to the * (calling) valstack. This works for everything except tail * calls, which must not "cumulate" valstack temps.
label: code-design
40466. * duk_hthread allocation and freeing.
40467. valstack
40468. callstack
40469. * Allocate initial stacks for a thread. Note that 'thr' must be reachable * as a garbage collection may be triggered by the allocation attempts. * Returns zero (without leaking memory) if init fails.
40470. For indirect allocs.
40471. catchstack
40472. **comment:** XXX: relocate
label: code-design
40473. DUK_HOBJECT_FLAG_EXOTIC_STRINGOBJ varies
40474. currently, even for Array.prototype
40475. normal valued properties
40476. 'prototype' property for all built-in objects (which have it) has attributes: * [[Writable]] = false, * [[Enumerable]] = false, * [[Configurable]] = false
40477. Architecture, OS, and compiler strings
40478. DUK_USE_LIGHTFUNC_BUILTINS
40479. Setup builtins from ROM objects. All heaps/threads will share * the same readonly objects.
40480. **comment:** Currently all built-in native functions are strict. * This doesn't matter for many functions, but e.g. * String.prototype.charAt (and other string functions) * rely on being strict so that their 'this' binding is * not automatically coerced.
label: code-design
40481. enable exotic behaviors last
40482. Built-in 'name' is not writable by default. Function 'name' * is writable to allow user code to set a '.name' on a native * function.
40483. provideThis=false
40484. initjs data is NUL terminated
40485. **comment:** XXX: try to optimize to 8 (would now be possible, <200 used)
label: code-design
40486. * First create all built-in bare objects on the empty valstack. ** Built-ins in the index range [0, DUK_NUM_BUILTINS-1] have value * stack indices matching their eventual thr->builtins[] index. ** Built-ins in the index range [DUK_NUM_BUILTINS, DUK_NUM_ALL_BUILTINS] * will exist on the value stack during init but won't be placed * into thr->builtins[]. These are objects referenced in some way * from thr->builtins[] roots but which don't need to be indexed by * Duktape through thr->builtins[] (e.g. user custom objects).
40487. [(builtin objects) name func]
40488. chain
40489. By default the global object is read-only which is often much * more of an issue than having read-only built-in objects (like * RegExp, Date, etc). Use a RAM-based copy of the global object * and the global environment object for convenience.
40490. Cast converts magic to 16-bit signed value
40491. Packed or unpacked tvl
40492. Currently all built-in native functions are strict. * duk_push_c_function() now sets strict flag, so * assert for it.
40493. **comment:** XXX: magic for getter/setter? use duk_def_prop()
label: code-design
40494. Object property allocation layout
40495. avoid empty label at the end of a compound statement
40496. Create a fresh object environment for the global scope. This is * needed so that the global scope points to the newly created RAM-based * global object.
40497. Inherit from ROM-based global object: less RAM usage, less transparent.
40498. **comment:** XXX: keep property attributes or tweak them here? * Properties will now be non-configurable even when they're * normally configurable for the global object.
label: code-design
40499. unsigned
40500. -> [... new_global new_globalenv new_global]
40501. 'constructor' property for all built-in objects (which have it) has attributes: * [[Writable]] = true, * [[Enumerable]] = false, * [[Configurable]] = true
40502. **comment:** XXX: refactor into internal helper, duk_clone_hobject()
label: code-design
40503. Encoding endianness must match target memory layout, * build scripts and genbuiltins.py must ensure this.
40504. def_value
40505. DUK_USE_USER_INITJS
40506. * The default property attributes are correct for all * function valued properties of built-in objects now.
40507. genbuiltins.py ensures

40508. DUK_HOBJECT_FLAG_CONSTRUCTABLE varies
40509. Endianness indicator
40510. **comment:** XXX: any way to avoid decoding magic bit; there are quite * many function properties and relatively few with magic values.
 label: code-design
40511. **comment:** XXX: this is a bit awkward because there is no exposed helper * in the API style, only this internal helper.
 label: code-design
40512. **comment:** XXX: function properties
 label: code-design
40513. * Since built-ins are not often extended, compact them.
40514. convenience
40515. * Pop built-ins from stack: they are now INCREF'd and * reachable from the builtins[] array or indirectly * through builtins[].
40516. **comment:** * Special post-tweaks, for cases not covered by the init data format. * * - Set Date.prototype.toGMTString to Date.prototype.toUTCString. * toGMTString is required to have the same Function object as * toUTCString in E5 Section B.2.6. Note that while Smjs respects * this, V8 does not (the Function objects are distinct). * * - Make DoubleError non-extensible. * * - Add info about most important effective compile options to Duktape. * * - Possibly remove some properties (values or methods) which are not * desirable with current feature options but are not currently * conditional in init data.
 label: code-design
40517. **comment:** XXX: set magic directly here? (it could share the c_nargs arg)
 label: code-design
40518. only Array.prototype matches
40519. [(builtin objects) name]
40520. DUK_USE_ROM_GLOBAL_CLONE || DUK_USE_ROM_GLOBAL_INHERIT
40521. **comment:** 0 = DUK_HOBJECT_CLASS_UNUSED
 label: code-design
40522. * Initialize built-in objects. Current thread must have a valstack * and initialization errors may longjmp, so a setjmp() catch point * must exist.
40523. exhaustive
40524. Alignment guarantee
40525. These functions have trouble working as lightfuncs. * Some of them have specific asserts and some may have * additional properties (e.g. 'require.id' may be written).
40526. DUK_USE_ROM_OBJECTS
40527. some assertions
40528. **comment:** XXX: compression
 label: code-design
40529. side effect free
40530. **comment:** integer mixed endian not really used now
 label: code-design
40531. * InitJS code - Ecmascript code evaluated from a built-in source * which provides e.g. backward compatibility. User can also provide * JS code to be evaluated at startup.
40532. always 1 arg
40533. * Create built-in objects by parsing an init bitstream generated * by genbuiltins.py.
40534. * Property attribute defaults are defined in E5 Section 15 (first * few pages); there is a default for all properties and a special * default for 'length' properties. Variation from the defaults is * signaled using a single flag bit in the bitstream.
40535. DUK_HOBJECT_FLAG_STRICT varies
40536. * Then decode the builtins init data (see genbuiltins.py) to * init objects
40537. DUK_HOBJECT_FLAG_EXOTIC_ARRAY varies
40538. must hold DUK_VARARGS
40539. Almost all global level Function objects are constructable * but not all: Function.prototype is a non-constructable, * callable Function.
40540. key, getter and setter, now reachable through object
40541. **comment:** XXX: compression (as an option)
 label: code-design
40542. **comment:** Clone the properties of the ROM-based global object to create a * fully RAM-based global object. Uses more memory than the inherit * model but more compliant.
 label: code-design
40543. No built-in functions are constructable except the top * level ones (Number, etc).
40544. Low memory options
40545. accessor flag not encoded explicitly
40546. probe
40547. no prototype or class yet
40548. DUK_USE_BUILTIN_INITJS
40549. **comment:** XXX: other properties of function instances; 'arguments', 'caller'.
 label: code-design
40550. always 0 args
40551. [(builtin objects)]
40552. must be signed
40553. DUK_HOBJECT_FLAG_NATIVEFUNCTION varies
40554. * Encoding constants, must match genbuiltins.py
40555. XXX: ARRAY_PART for Array prototype
40556. * For top-level objects, 'length' property has the following * default attributes: non-writable, non-enumerable, non-configurable * (E5 Section 15). * * However, 'length' property for Array.prototype has attributes * expected of an Array instance which are different: writable, * non-enumerable, non-configurable (E5 Section 15.4.5.2). * * This is currently determined implicitly based on class; there are * no attribute flags in the init data.
40557. native function properties
40558. all native functions have NEWENV
40559. push operation normalizes NaNs
40560. **comment:** XXX: This won't be shown in practice now * because this code is not run when builtins * are in ROM.
 label: code-design
40561. no prototype
40562. Copy the property table verbatim; this handles attributes etc. * For ROM objects it's not necessary (or possible) to update * refcounts so leave them as is.
40563. no need to decref
40564. **comment:** XXX: Shrink the stacks to minimize memory usage? May not * be worth the effort because terminated threads are usually * garbage collected quite soon.
 label: code-design
40565. **comment:** XXX: store 'bcode' pointer to activation for faster lookup?
 label: code-design
40566. Write bytecode executor's curr_pc back to topmost activation (if any).
40567. Here we could remove references to built-ins, but it may not be * worth the effort because built-ins are quite likely to be shared * with another (unterminated) thread, and terminated threads are also * usually garbage collected quite quickly. Also, doing DECREFs * could trigger finalization, which would run on the current thread * and have access to only some of the built-ins. Garbage collection * deals with this correctly already.
40568. Order of unwinding is important
40569. ptr_curr_pc != NULL only when bytecode executor is active.

40570. unwinds valstack, updating refcounts
40571. DUK_USE_DEBUGGER_SUPPORT
40572. * Thread support.
40573. side effects, possibly errors
40574. Note: this may cause a corner case situation where a finalizer * may see a currently reachable activation whose 'func' is NULL.
40575. avoid warning (unsigned)
40576. unsigned
40577. * Restore 'caller' property for non-strict callee functions.
40578. **comment:** * Note: must use indirect variant of DUK_REALLOC() because underlying * pointer may be changed by mark-and-sweep.
 label: code-design
40579. true, despite side effect resizes
40580. note: any entries above the callstack top are garbage and not zeroed
40581. cannot grow
40582. With lightfuncs, act 'func' may be NULL
40583. The act->prev_caller should only be set if the entry for 'caller' * exists (as it is only set in that case, and the property is not * configurable), but handle all the cases anyway.
40584. func is NULL for lightfunc
40585. must be, since env was created when catcher was created
40586. note: any entries above the catchstack top are garbage and not zeroed
40587. * Update preventcount
40588. avoid side effect issues
40589. current lex_env of the activation (created for catcher)
40590. Note that multiple catchstack entries may refer to the same * callstack entry.
40591. **comment:** * Manipulation of thread stacks (valstack, callstack, catchstack). * * Ideally unwinding of stacks should have no side effects, which would * then favor separate unwinding and shrink check primitives for each * stack type. A shrink check may realloc and thus have side effects. * * However, currently callstack unwinding itself has side effects, as it * needs to DECREF multiple objects, close environment records, etc. * Stacks must thus be unwound in the correct order by the caller. * * (XXX: This should be probably reworked so that there is a shared * unwind primitive which handles all stacks as requested, and knows * the proper order for unwinding.) * * Valstack entries above 'top' are always kept initialized to * "undefined unused". Callstack and catchstack entries above 'top' * are not zeroed and are left as garbage. * * Value stack handling is mostly a part of the API implementation.
 label: code-design
40592. * Unwind debugger state. If we unwind while stepping * (either step over or step into), pause execution.
40593. Note: any entries above the callstack top are garbage and not zeroed. * Also topmost activation idx_retval is garbage (not zeroed), and must * be ignored.
40594. * Reference count updates * * Note: careful manipulation of refcounts. The top is * not updated yet, so all the activations are reachable * for mark-and-sweep (which may be triggered by decref). * However, the pointers are NULL so this is not an issue.
40595. side effects
40596. no incref needed
40597. Just transfer the refcount from act->prev_caller to tv_caller, * so no need for a refcount update. This is the expected case.
40598. check that there is space for at least one new entry
40599. Because new_size != 0, if condition doesn't need to be * (p != NULL || new_size == 0).
40600. No need for a NULL/zero-size check because new_size > 0
40601. * We could clear the book-keeping variables for the topmost activation, * but don't do so now.
40602. prototype is lex_env before catcher created
40603. avoid side effects
40604. this is a bit approximate (errors out before max is reached); this is OK
40605. Pause for all step types: step into, step over, step out. * This is the only place explicitly handling a step out.
40606. * The loop below must avoid issues with potential callstack * reallocations. A resize (and other side effects) may happen * e.g. due to finalizer/errhandler calls caused by a refzero or * mark-and-sweep. Arbitrary finalizers may run, because when * an environment record is refzero'd, it may refer to arbitrary * values which also become refzero'd. * * So, the pointer 'p' is re-looked-up below whenever a side effect * might have changed it.
40607. **comment:** * Since there are no references in the catcher structure, * unwinding is quite simple. The only thing we need to * look out for is popping a possible lexical environment * established for an active catch clause.
 label: code-design
40608. **comment:** XXX: Here we have a nasty dependency: the need to manipulate * the callstack means that catchstack must always be unwound by * the caller before unwinding the callstack. This should be fixed * later.
 label: code-design
40609. shrink failure is not fatal
40610. There is no need to decref anything else than 'env': if 'env' * becomes unreachable, refzero will handle decref'ing its prototype.
40611. * Close environment record(s) if they exist. * * Only variable environments are closed. If lex_env != var_env, it * cannot currently contain any register bound declarations. * * Only environments created for a NEWENV function are closed. If an * environment is created for e.g. an eval call, it must not be closed.
40612. common case, already closed, so skip
40613. **comment:** * User declarations, e.g. prototypes for user functions used by Duktape * macros. Concretely, if DUK_USE_PANIC_HANDLER is used and the macro * value calls a user function, it needs to be declared for Duktape * compilation to avoid warnings.
 label: code-design
40614. * The 'duktape.h' header provides the public API, but also handles all * compiler and platform specific feature detection, Duktape feature * resolution, inclusion of system headers, etc. These have been merged * because the public API is also dependent on e.g. detecting appropriate * C types which is quite platform/compiler specific especially for a non-C99 * build. The public API is also dependent on the resolved feature set. * * Some actions taken by the merged header (such as including system headers) * are not appropriate for building a user application. The define * DUK_COMPILING_DUKTAPE allows the merged header to skip/include some * sections depending on what is being built.
40615. DUK_INTERNAL_H_INCLUDED
40616. autogenerated: strings and built-in object init data
40617. * Duktape includes (other than duk_features.h) * * The header files expect to be included in an order which satisfies header * dependencies correctly (the headers themselves don't include any other * includes). Forward declarations are used to break circular struct/typedef * dependencies.
40618. builtins need e.g. duk_val tag definitions
40619. * Top-level include file to be used for all (internal) source files. * * Source files should not include individual header files, as they * have not been designed to be individually included.
40620. * Wrapper for jmp_buf. * * This is used because jmp_buf is an array type for backward compatibility. * Wrapping jmp_buf in a struct makes pointer references, sizeof, etc, * behave more intuitively. * * http://en.wikipedia.org/wiki/Setjmp.h#Member_types
40621. **comment:** unused
 label: code-design
40622. DUK_JMPBUF_H_INCLUDED
40623. use SameValue instead of non-strict equality
40624. conversions, coercions, comparison, etc
40625. E5 Sections 11.8.3, 11.8.5; x <= y --> not (x > y) --> not (y < x)
40626. identifiers and environment handling
40627. Flags for duk_js_compare_helper().
40628. E5 Sections 11.8.1, 11.8.5; x < y
40629. bytecode execution
40630. E5 Sections 11.8.2, 11.8.5; x > y --> y < x
40631. **comment:** duk_handle_call_xxx: call ignores C recursion limit (for errhandler calls)

recursion * limit" flag is not applicable) * - instead of making the call, this helper just performs the thread * setup and returns; the bytecode executor then restarts execution * internally * - ecmascript functions are never 'vararg' functions (they access * varargs through the 'arguments' object) * * The callstack of the target contains an earlier Ecmascript call in case * of an Ecmascript-to-Ecmascript call (whose idx_retval is updated), or * is empty in case of an initial Duktape.Thread.resume(). * * The first thing to do here is to figure out whether an ecma-to-ecma * call is actually possible. It's not always the case if the target is * a bound function; the final function may be native. In that case, * return an error so caller can fall back to a normal call path.

label: code-design

40697. use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to the existing one)

40698. step 11

40699. duk_tval ptr for 'func' on stack (borrowed reference) or tv_func_copy

40700. Note: either pointer may be NULL (at entry), so don't assert

40701. **comment:** XXX: unify handling with native call.

label: code-design

40702. Lightfuncs are always native functions and have "newenv".

40703. **comment:** XXX: we should never shrink here; when we error out later, we'd * need to potentially grow the value stack in error unwind which could * cause another error.

label: code-design

40704. XXX: byte offset?

40705. first call

40706. may need unwind

40707. * Helper for updating callee 'caller' property.

40708. side effects

40709. bound chain resolved

40710. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * Lightfunc detection happens here too. Note that lightweight functions * can be wrapped by (non-lightweight) bound functions so we must resolve * the bound function chain first.

40711. If the debugger is active we need to force an interrupt so that * debugger breakpoints are rechecked. This is important for function * calls caused by side effects (e.g. when doing a DUK_OP_GETPROP), see * GH-303. Only needed for success path, error path always causes a * breakpoint recheck in the executor. It would be enough to set this * only when returning to an Ecmascript activation, but setting the flag * on every return should have no ill effect.

40712. **comment:** XXX: very slow - better to bulk allocate a gap, and copy * from args_array directly (we know it has a compact array * part, etc).

label: code-design

40713. No need to re-lookup 'act' at present: no side effects.

40714. formals for 'func' (may be NULL if func is a C function)

40715. Normal non-bound function.

40716. valstack index of start of args (arg1) (relative to entry valstack_bottom)

40717. should never happen for a strict callee

40718. Backup 'caller' property and update its value.

40719. * Valstack manipulation for results.

40720. no need to handle thread state book-keeping here

40721. out of spec, must be configurable

40722. This may only happen if built-ins are being "torn down". * This behavior is out of specification scope.

40723. same as during entry

40724. Note: careful with indices like '-x'; if 'x' is zero, it refers to bottom

40725. original input stack before nargs/nregs handling must be * intact for 'arguments' object

40726. third arg: absolute index (to entire valstack) of idx_bottom of new activation

40727. strict: non-deletable, non-writable

40728. -> [... func this arg1 ... argN _Args length]

40729. 1 = num actual 'return values'

40730. **comment:** XXX: Is this INCREF necessary? 'func' is always a borrowed * reference reachable through the value stack? If changed, stack * unwind code also needs to be fixed to match.

label: code-design

40731. current thread

40732. * Setup value stack: clamp to 'nargs', fill up to 'nregs' * * Value stack may either grow or shrink, depending on the * number of func registers and the number of actual arguments. * If nregs >= 0, func wants args clamped to 'nargs'; else it * wants all args (= 'num_stack_args').

40733. Currently the bytecode executor and executor interrupt * instruction counts are off because we don't execute the * interrupt handler when we're about to exit from the initial * user call into Duktape. * * If we were to execute the interrupt handler here, the counts * would match. You can enable this block manually to check * that this is the case.

40734. Duktape/C API guaranteed entries (on top of args)

40735. Success path.

40736. XXX: We can't resize the value stack to a size smaller than the * current top, so the order of the resize and adjusting the stack * top depends on the current vs. final size of the value stack. * The operations could be combined to avoid this, but the proper * fix is to only grow the value stack on a function call, and only * shrink it (without throwing if the shrink fails) on function * return.

40737. bound function chain has already been resolved

40738. base of known return values

40739. **comment:** * Call handling. * * Main functions are: * * - duk_handle_call_unprotected(): unprotected call to Ecmascript or * Duktape/C function * - duk_handle_call_protected(): protected call to Ecmascript or * Duktape/C function * - duk_handle_safe_call(): make a protected C call within current * activation * - duk_handle_ecma_call_setup(): Ecmascript-to-Ecmascript calls * (not always possible), including tail calls and coroutine resume * * See 'execution.rst'. * * Note: setjmp() and local variables have a nasty interaction, * see execution.rst; non-volatile locals modified after setjmp() * call are not guaranteed to keep their value.

label: code-design

40740. step 11.c is relevant only if non-strict (checked in 11.c.ii)

40741. * Manipulate valstack so that args are on the current bottom and the * previous caller's 'this' binding (which is the value preceding the * current bottom) is replaced with the new 'this' binding: * * [... this_old | (crud) func this_new arg1 ... argN] * --> [... this_new | arg1 ... argN] * * For tail calling to work properly, the valstack bottom must not grow * here; otherwise crud would accumulate on the valstack.

40742. Return value handling.

40743. **comment:** XXX: because we unwind stacks above, thr->heap->curr_thread is at * risk of pointing to an already freed thread. This was indeed the * case in test-bug-multithread-valgrind.c, until duk_handle_call() * was fixed to restore thr->heap->curr_thread before rethrowing an * uncaught error.

label: code-design

40744. **comment:** XXX: currently NULL allocations are not supported; remove if later allowed

label: code-design

40745. flags

40746. Unwind the topmost callstack entry before reusing it

40747. borrowed, no refcount

40748. same thread

40749. may be NULL

40750. pop 'name'

40751. Other longjmp types are handled by executor before propagating * the error here.

40752. * Bytecode executor call. ** Execute bytecode, handling any recursive function calls and * thread resumptions. Returns when execution would return from * the entry level activation. When the executor returns, a * single return value is left on the stack top. ** The only possible longjmp() is an error (DUK_LJ_TYPE_THROW), * other types are handled internally by the executor.

40753. **comment:** XXX: duk_get_length?
label: code-design

40754. side effects (currently none though)

40755. [... formals arguments map mappedNames]

40756. # argument registers target function wants (< 0 => "as is")

40757. For now all calls except Ecma-to-Ecma calls prevent a yield.

40758. duk_hthread_callstack_unwind() will decrease this on unwind

40759. [... formals]

40760. See: tests/ecmascript/test-spec-bound-constructor.js

40761. * duk_handle_call_protected() and duk_handle_call_unprotected(): * call into a Duktape/C or an Ecmascript function from any state. ** Input stack (thr): ** [func this arg1 ... argN] ** [retval] (DUK_EXEC_SUCCESS) * [errobj] (DUK_EXEC_ERROR (normal error), protected call) * * Even when executing a protected call an error may be thrown in rare cases * such as an insane num_stack_args argument. If there is no catchpoint for * such errors, the fatal error handler is called. ** The error handling path should be error free, even for out-of-memory * errors, to ensure safe sandboxing. (As of Duktape 1.4.0 this is not * yet the case, see XXX notes below.)

40762. * Init argument related properties

40763. Tolerate act_caller->func == NULL which happens in * some finalization cases; treat like unknown caller.

40764. **comment:** * duk_handle_safe_call(): make a "C protected call" within the * current activation. ** The allowed thread states for making a call are the same as for * duk_handle_call_xxx(). * * Error handling is similar to duk_handle_call_xxx(); errors may be thrown * (and result in a fatal error) for insane arguments.
label: code-design

40765. **comment:** XXX: remove DUK_CALL_FLAG_IGNORE_RECLIMIT flag: there's now the * reclimit bump?
label: code-design

40766. DUK_USE_ASSERTIONS

40767. **comment:** XXX: inefficient; block remove primitive
label: code-design

40768. Note: multiple threads may be simultaneously in the RUNNING * state, but not in the same "resume chain".

40769. Lightfuncs are always considered strict.

40770. Ensure space for final configuration (idx_rebase + num_stack_rets) * and intermediate configurations.

40771. **comment:** Note: nargs (and nregs) may be negative for a native, * function, which indicates that the function wants the * input stack "as is" (i.e. handles "vararg" arguments).
label: code-design

40772. 'act' already set above

40773. * Setup value stack: clamp to 'nargs', fill up to 'nregs'

40774. lightfuncs are treated like objects and not coerced

40775. [... func this | arg1 ... argN] ('this' must precede new bottom)

40776. * Make the C call

40777. **comment:** XXX: rework
label: code-design

40778. index

40779. act_caller->func may be NULL in some finalization cases, * just treat like we don't know the caller.

40780. XXX: Multiple tv_func lookups are now avoided by making a local * copy of tv_func. Another approach would be to compute an offset * for tv_func from valstack bottom and recomputing the tv_func * pointer quickly as valstack + offset instead of calling duk_get_tval().

40781. **comment:** XXX: this should be an assert
label: test

40782. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur. If we end up not making the * call we must restore the value.

40783. We don't need to sync back thr->ptr_curr_pc here because * the bytecode executor always has a setjmp catchpoint which * does that before errors propagate to here.

40784. [... | (crud) errobj]

40785. Start filling in the activation

40786. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
label: code-design

40787. assumed to bottom relative

40788. Previous value doesn't need refcount changes because its ownership * is transferred to prev_caller.

40789. Preliminaries, required by setjmp() handler. Must be careful not * to throw an unintended error here.

40790. * Manipulate value stack so that exactly 'num_stack_rets' return * values are at 'idx_rebase' in every case, assuming there are * 'rc' return values on top of stack. * * This is a bit tricky, because the called C function operates in * the same activation record and may have e.g. popped the stack * empty (below idx_rebase).

40791. Exotic behaviors are only enabled for arguments objects * which have a parameter map (see E5 Section 10.6 main * algorithm, step 12). ** In particular, a non-strict arguments object with no * mapped formals does *NOT* get exotic behavior, even * for e.g. "caller" property. This seems counterintuitive * but seems to be the case.

40792. [... func this <some bound args> arg1 ... argN _Args]

40793. really 'not applicable' anymore, should not be referenced after this

40794. idx of first argument on stack

40795. thr->ptr_curr_pc is set by bytecode executor early on entry

40796. Chop extra retvals away / extend with undefined.

40797. setjmp catchpoint setup

40798. [... errobj]

40799. Longjmp state is kept clean in success path

40800. **comment:** * Recursion limit check. ** Note: there is no need for an "ignore recursion limit" flag * for duk_handle_safe_call now.
label: code-design

40801. final configuration

40802. Automatic error throwing, retval check.

40803. no need to decref previous value

40804. Explicit check for fastint downgrade.

40805. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur.

40806. refer to callstack entries below current

40807. Note: not a valid stack index if num_stack_args == 0

40808. tailcall cannot be flagged to resume calls, and a * previous frame must exist

40809. * Shared helper for non-bound func lookup. ** Returns duk_hobject * to the final non-bound function (NULL for lightfunc).

40810. * Preliminary activation record and valstack manipulation. * The concrete actions depend on whether we're dealing * with a tail call (reuse an existing activation), a resume, * or a normal call. ** The basic actions, in varying order, are: * * - Check stack size for call handling * - Grow call stack if necessary (non-tail-calls) * - Update current activation (idx_retval) if necessary * (non-tail, non-resume calls) * - Move start of args (idx_args) to valstack bottom * (tail calls) * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.

40811. must be updated to work properly (e.g. creation of 'arguments')

40812. write back
40813. extend with undefined
40814. **comment:** * Helper for handling a "bound function" chain when a call is being made. ** Follows the bound function chain until a non-bound function is found. * Prepends the bound arguments to the value stack (at idx_func + 2), * updating 'num_stack_args' in the process. The 'this' binding is also * updated if necessary (at idx_func + 1). Note that for constructor calls * the 'this' binding is never updated by [[BoundThis]]. ** XXX: bound function chains could be collapsed at bound function creation * time so that each bound function would point directly to a non-bound * function. This would make call time handling much easier.
label: code-design
40815. * Helper for setting up var_env and lex_env of an activation, * assuming it does NOT have the DUK_HOBJECT_FLAG_NEWENV flag.
40816. should actually never happen, but check anyway
40817. Use a new environment but there's no 'arguments' object; * delayed environment initialization. This is the most * common case.
40818. Use loop to minimize code size of relookup after bound function case
40819. **comment:** XXX: should this happen in the callee's activation or after unwinding?
label: code-design
40820. don't want an intermediate exposed state with invalid pc
40821. catchstack limits
40822. # total registers target function wants on entry (< 0 => never for ecma calls)
40823. idx_argbase
40824. * Value stack resize and stack top adjustment helper. ** XXX: This should all be merged to duk_valstack_resize_raw().
40825. don't want an intermediate exposed state with func == NULL
40826. Error; error value is in heap->j.value1.
40827. * Store entry state.
40828. **comment:** Note: cannot portably debug print a function pointer, hence 'func' not printed!
label: code-design
40829. [...] retval]
40830. no incref
40831. may have side effects
40832. we're running inside the caller's activation, so no change in call/catch stack or valstack bottom
40833. if a tail call: * - an Ecmascript activation must be on top of the callstack * - there cannot be any active catchstack entries
40834. NULL for lightfunc
40835. Use a new environment and there's an 'arguments' object. * We need to initialize it right now.
40836. Lightweight function: never bound, so terminate.
40837. # argument registers target function wants (< 0 => never for ecma calls)
40838. compiler ensures this
40839. internal spare
40840. already NULLED (by unwind)
40841. * Misc shared helpers.
40842. out of spec, don't care
40843. [...] func this arg1 ... argN envobj]
40844. + spare
40845. keep current valstack_top
40846. XXX: optimize value stack operation
40847. bottom of new func
40848. can't happen when keeping current stack size
40849. Ensure there is internal valstack spare before we exit; this may * throw an alloc error. The same guaranteed size must be available * as before the call. This is not optimal now: we store the valstack * allocated size during entry; this value may be higher than the * minimal guarantee for an application.
40850. steps 11.c.ii.1 - 11.c.ii.4, but our internal book-keeping * differs from the reference model
40851. XXX: byte offset
40852. may be changed by call
40853. DUK_USE_NONSTD_FUNC_CALLER_PROPERTY
40854. topmost activation idx_retval is considered garbage, no need to init
40855. [args [crud] arguments]
40856. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. ** If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. ** If the final target function cannot be handled by an ecma-to-ecma * call, return to the caller with a return value indicating this case. * The bound chain is resolved and the caller can resume with a plain * function call.
40857. **comment:** XXX: some overlapping code; cleanup
label: code-design
40858. must be signed for loop structure
40859. [...] func this <bound args> arg1 ... argN]
40860. vararg
40861. no need to create environment record now; leave as NULL
40862. [...] arg1 ... argN]
40863. caller must check
40864. set exotic behavior only after we're done
40865. [...] name]
40866. [...] | (crud)]
40867. [...] arg1 ... argN envobj]
40868. nothing to incref
40869. Note: may be NULL if first call
40870. num args starting from idx_argbase
40871. check that the valstack has space for the final amount and any * intermediate space needed; this is unoptimal but should be safe
40872. If caller is global/eval code, 'caller' should be set to * 'null'. ** XXX: there is no exotic flag to infer this correctly now. * The NEWENV flag is used now which works as intended for * everything (global code, non-strict eval code, and functions) * except strict eval code. Bound functions are never an issue * because 'func' has been resolved to a non-bound function.
40873. valstack index of 'func' and retval (relative to entry valstack_bottom)
40874. **comment:** These base values are never used, but if the compiler doesn't know * that DUK_ERROR() won't return, these are needed to silence warnings. * On the other hand, scan-build will warn about the values not being * used, so add a DUK_UNREF.
label: code-design
40875. [...] | [...] | errobj (M * undefined)] where M = num_stack_rets - 1
40876. Restore entry thread executor curr_pc stack frame pointer.
40877. idx_args = idx_func + 2
40878. On entry, item at idx_func is a bound, non-lightweight function, * but we don't rely on that below.
40879. Success path handles
40880. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** This handling is now identical for C and Ecmascript functions. * C functions always have the 'NEWENV' flag set, so their * environment record initialization is delayed (which is good). ** Delayed creation (on demand) is handled in duk_js_var.c.
40881. XXX: bump preventcount by one for the duration of this call?

40882. stack discipline consistency check
40883. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. * * Delayed creation (on demand) is handled in duk_js_var.c.
40884. See: test-bug-tailcall-preventyield-assert.c.
40885. insert 'undefined' values at idx_rcbase to get the * return values to idx_rebase
40886. [... func this (crud) retval]
40887. XXX: assert? compiler is responsible for this never happening
40888. Helper for creating the arguments object and adding it to the env record * on top of the value stack. This helper has a very strict dependency on * the shape of the input stack.
40889. * Shift to new valstack_bottom.
40890. **comment:** XXX: unnecessary, handle in adjust
 label: code-design
40891. already updated
40892. to avoid relookups
40893. **comment:** * Update idx_retval of current activation. * * Although it might seem this is not necessary (bytecode executor * does this for Ecmascript-to-Ecmascript calls; other calls are * handled here), this turns out to be necessary for handling yield * and resume. For them, an Ecmascript-to-native call happens, and * the Ecmascript call's idx_retval must be set for things to work.
 label: code-design
40894. Note: either pointer may be NULL (at entry), so don't assert.
40895. * Create required objects: * - 'arguments' object: array-like, but not an array * - 'map' object: internal object, tied to 'arguments' * - 'mappedNames' object: temporary value used during construction
40896. [... func this arg1 ... argN]
40897. replace in stack
40898. **comment:** XXX: check .caller writability?
 label: code-design
40899. Since stack indices are not reliable, we can't do anything useful * here. Invoke the existing setjmp catcher, or if it doesn't exist, * call the fatal error handler.
40900. duk_tvall ptr for 'func' on stack (borrowed reference)
40901. tail call -> reuse current "frame"
40902. Unwind catchstack entries referring to the callstack entry we're reusing
40903. local copy to avoid relookups
40904. **comment:** * Error during call. The error value is at heap->jl.value1. * * The very first thing we do is restore the previous setjmp catcher. * This means that any error in error handling will propagate outwards * instead of causing a setjmp() re-entry above.
 label: code-design
40905. * Determine call type, then finalize activation, shift to * new value stack bottom, and call the target.
40906. * Arguments object creation. * * Creating arguments objects involves many small details, see E5 Section * 10.6 for the specific requirements. Much of the arguments object exotic * behavior is implemented in duk_hobject_props.c, and is enabled by the * object flag DUK_HOBJECT_FLAG_EXOTIC_ARGUMENTS.
40907. 'arguments'
40908. **comment:** XXX: remove the preventcount and make yield walk the callstack? * Or perhaps just use a single flag, not a counter, faster to just * set and restore?
 label: code-design
40909. at least errobj must be on stack
40910. **comment:** XXX: inefficient; block insert primitive
 label: code-design
40911. **comment:** XXX: space in valstack? see discussion in duk_handle_call_xxx().
 label: code-design
40912. **comment:** * Tailcall handling * * Although the callstack entry is reused, we need to explicitly unwind * the current activation (or simulate an unwind). In particular, the * current activation must be closed, otherwise something like * test-bug-reduce-judofyr.js results. Also catchstack needs be unwound * because there may be non-error-catching label entries in valid tail calls.
 label: code-design
40913. Unwind.
40914. Restore the previous setjmp catcher so that any error in * error handling will propagate outwards rather than re-enter * the same handler. However, the error handling path must be * designed to be error free so that sandboxing guarantees are * reliable, see e.g. <https://github.com/svaarala/duktape/issues/476>.
40915. * Setup a preliminary activation and figure out nargs/nregs. * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
40916. Error path.
40917. **comment:** XXX: could be eliminated with valstack adjust
 label: code-design
40918. * Ecmascript call
40919. clamp anything above nargs
40920. **comment:** XXX: this could be more compact by accessing the internal properties * directly as own properties (they cannot be inherited, and are not * externally visible).
 label: code-design
40921. Argument validation and func/args offset.
40922. # total registers target function wants on entry (< 0 => "as is")
40923. cannot be strict (never mapped variables)
40924. * C call recursion depth check, which provides a reasonable upper * bound on maximum C stack size (arbitrary C stack growth is only * possible by recursive handle_call / handle_safe_call calls).
40925. no compact
40926. XXX: callstack unwind may now throw an error when closing * scopes; this is a sandboxing issue, described in: * <https://github.com/svaarala/duktape/issues/476>
40927. * Valstack spare check
40928. * Determine the effective 'this' binding and coerce the current value * on the valstack to the effective one (in-place, at idx_this). * * The current this value in the valstack (at idx_this) represents either: * - the caller's requested 'this' binding; or * - a 'this' binding accumulated from the bound function chain * * The final 'this' binding for the target function may still be * different, and is determined as described in E5 Section 10.4.3. * * For global and eval code (E5 Sections 10.4.1 and 10.4.2), we assume * that the caller has provided the correct 'this' binding explicitly * when calling, i.e.: * * - global code: this=global object * - direct eval: this=copy from eval() caller's this binding * - other eval: this=global object * * Note: this function may cause a recursive function call with arbitrary * side effects, because ToObject() may be called.
40929. 'func' wants stack "as is"
40930. * Init arguments properties, map, etc.
40931. name
40932. [... arg1 ... argN envobj argobj]
40933. act->func
40934. These are just convenience "wiping" of state. Side effects should * not be an issue here: thr->heap and thr->heap->jl have a stable * pointer. Finalizer runs etc capture even out-of-memory errors so * nothing should throw here.
40935. idx_this = idx_func + 1
40936. or a non-catching entry
40937. **comment:** XXX: tv_func is not actually needed
 label: requirement
40938. XXX: this doesn't actually work properly for tail calls, so * tail calls are disabled when DUK_USE_NONSTD_FUNC_CALLER_PROPERTY * is in use.
40939. different thread

40940. **comment:** Function.prototype.bind() should never let this happen, * ugly error message is enough.
label: code-design

40941. **comment:** XXX: this needs to be reworked so that we never shrink the value * stack on function entry so that we never need to grow it here. * Needing to grow here is a sandboxing issue because we need to * allocate which may cause an error in the error handling path * and thus propagate an error out of a protected call.
label: code-design

40942. step 12

40943. [... name name]

40944. -> [... func this arg1 ... argN _Args]

40945. steps 13-14

40946. no prototype

40947. leave formals on stack for later use

40948. * Return to bytecode executor, which will resume execution from * the topmost activation.

40949. Note: 'func' is popped from valstack here, but it is * already reachable from the activation.

40950. [... func this arg1 ... argN] (not tail call) * [this | arg1 ... argN] (tail call) * * idx_args updated to match

40951. * Native call.

40952. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
label: code-design

40953. duk_has_prop() popped the second 'name'

40954. A debugger forced interrupt check is not needed here, as * problematic safe calls are not caused by side effects.

40955. **comment:** XXX: block removal API primitive
label: code-design

40956. The final object may be a normal function or a lightfunc. * We need to re-lookup tv_func because it may have changed * (also value stack may have been resized). Loop again to * do that; we're guaranteed not to come here again.

40957. **comment:** * Cleanups (all statement parsing flows through here). * * Pop label site and reset labels. Reset 'next temp' to value at * entry to reuse temps.
label: code-design

40958. * Rewind lexer. * * duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp * literal" mode with current strictness. * * curr_token line number info should be initialized for pass 2 before * generating prologue, to ensure prologue bytecode gets nice line numbers.

40959. only functions can have arguments

40960. **comment:** * Ecmascript compiler. * * Parses an input string and generates a function template result. * Compilation may happen in multiple contexts (global code, eval * code, function code). * * The parser uses a traditional top-down recursive parsing for the * statement level, and an operator precedence based top-down approach * for the expression level. The attempt is to minimize the C stack * depth. Bytecode is generated directly without an intermediate * representation (tree), at the cost of needing two passes over each * function. * * The top-down recursive parser functions are named "duk_parse_XXX". * * Recursion limits are in key functions to prevent arbitrary C recursion: * function body parsing, statement parsing, and expression parsing. * * See doc/compiler.rst for discussion on the design. * * A few typing notes: * * - duk_regconst_t: unsigned, no marker value for "none" * - duk_reg_t: signed, < 0 = none * - PC values: duk_int_t, negative values used as markers
label: code-design

40961. DUK_TOK_SNEQ

40962. DUK_TOK_TRUE

40963. * Variant 2

40964. UnaryExpression

40965. Buffer length is bounded to 0xffff automatically, avoid compile warning.

40966. **comment:** * Shared handling for logical AND and logical OR. * * args = (truthval << 8) + rbp * * Truthval determines when to skip right-hand-side. * For logical AND truthval=1, for logical OR truthval=0. * * See doc/compiler.rst for discussion on compiling logical * AND and OR expressions. The approach here is very simplistic, * generating extra jumps and multiple evaluations of truth values, * but generates code on-the-fly with only local back-patching. * * Both logical AND and OR are syntactically left-associated. * However, logical ANDs are compiled as right associative * expressions, i.e. "A && B && C" as "A && (B && C)", to allow * skip jumps to skip over the entire tail. Similarly for logical OR.
label: code-design

40967. No need to assert, buffer size maximum is 0xffff.

40968. Tailcalls are handled by back-patching the TAILCALL flag to the * already emitted instruction later (in return statement parser). * Since A and C have a special meaning here, they cannot be "shuffled".

40969. inserted jump

40970. curr_token follows 'function'

40971. Output shuffle needed after main operation

40972. DUK_TOK_TRY

40973. evaluate to final form (e.g. coerce GETPROP to code), throw away temp

40974. push before advancing to keep reachable

40975. DUK_TOK_GET

40976. init function state: inits valstack allocations

40977. for storing/restoring the varmap binding for catch variable

40978. convenience helpers

40979. * Build function fixed size 'data' buffer, which contains bytecode, * constants, and inner function references. * * During the building phase 'data' is reachable but incomplete. * Only incref's occur during building (no refzero or GC happens), * so the building process is atomic.

40980. omitted

40981. Check statement type based on the first token type. * * Note: expression parsing helpers expect 'curr_tok' to * contain the first token of the expression upon entry.

40982. -> pushes fixed buffer

40983. Not safe to use 'reg_varbind' as assignment expression * value, so go through a temp.

40984. Ensure compact use of temps.

40985. default case exists: go there if no case matches

40986. **comment:** Temporary structure used to pass a stack allocated region through * duk_safe_call().
label: code-design

40987. eat 'do'

40988. * Do-while statement is mostly trivial, but there is special * handling for automatic semicolon handling (triggered by the * DUK_ALLOW_AUTO_SEMI_ALWAYS) flag related to a bug filed at: * * https://bugs.ecmascript.org/show_bug.cgi?id=8 * * See doc/compiler.rst for details.

40989. -> [... name index]

40990. max # of values initialized in one MPUTARR set

40991. * Reset function state and perform register allocation, which creates * 'varmap' for second pass. Function prologue for variable declarations, * binding value initializations etc is emitted as a by-product. * * Strict mode restrictions for duplicate and invalid argument * names are checked here now that we know whether the function * is actually strict. See: test-dev-strict-mode-boundary.js. * * Inner functions are compiled during pass 1 and are not reset.

40992. Strict equality is NOT enough, because we cannot use the same * constant for e.g. +0 and -0.

40993. e.g. DUK_OP_PREINCV

40994. reuse 'res' as 'left'

40995. Initial bytecode size allocation.

40996. XXX: patch initial size afterwards?

40997. yes, must set array length explicitly

40998. in-place setup

40999. Does not assume that jump_pc contains a DUK_OP_JUMP previously; this is intentional * to allow e.g. an INVALID opcode be overwritten with a JUMP (label management uses this).

41000. **comment:** Delete semantics are a bit tricky. The description in E5 specification * is kind of confusing, because it distinguishes between resolvability of * a reference (which is only known at runtime) seemingly at compile time * (= SyntaxError throwing).
label: code-design

41001. binding power must be high enough to NOT allow comma expressions directly
41002. flags
41003. _Formals: omitted if function is guaranteed not to need a (non-strict) arguments object
41004. Note: these variables must reside in 'curr_func' instead of the global * context: when parsing function expressions, expression parsing is nested.
41005. implicit_return_value
41006. * Initialize function state for a zero-argument function
41007. * Special name handling
41008. eat 'for'
41009. right associative
41010. init 'curr_token'
41011. allow a constant to be returned
41012. DUK_TOK_LBRACKET
41013. XXX: check num_args
41014. DUK_TOK_LPAREN
41015. directly uses slow accesses
41016. **comment:** It'd be nice to do something like this - but it doesn't * work for closures created inside the catch clause.
label: code-design

41017. duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp literal" mode with current strictness
41018. **comment:** XXX: Shuffling support could be implemented here so that LDINT+LDINTX * would only shuffle once (instead of twice). The current code works * though, and has a smaller compiler footprint.
label: code-design

41019. * Post-increment/decrement will return the original value as its * result value. However, even that value will be coerced using * ToNumber() which is quite awkward. Specific bytecode opcodes * are used to handle these semantics. * * Note that post increment/decrement has a "no LineTerminator here" * restriction. This is handled by duk_expr_lbp(), which forcibly terminates * the previous expression if a LineTerminator occurs before '++'/'--'.
41020. DUK_TOK_LT
41021. If duk_ivalue_toplain_raw() allocates a temp, forget it and * restore next temp state.
41022. catch depth from current func
41023. eat colon
41024. **comment:** XXX: since the enumerator may be a memory expensive object, * perhaps clear it explicitly here? If so, break jump must * go through this clearing operation.
label: code-design

41025. [... pattern flags]
41026. * Init compiler and lexer contexts
41027. DUK_TOK_NEQ
41028. * Code emission helpers * * Some emission helpers understand the range of target and source reg/const * values and automatically emit shuffling code if necessary. This is the * case when the slot in question (A, B, C) is used in the standard way and * for opcodes the emission helpers explicitly understand (like DUK_OP_CALL). * * The standard way is that: * - slot A is a target register * - slot B is a source register/constant * - slot C is a source register/constant * * If a slot is used in a non-standard way the caller must indicate this * somehow. If a slot is used as a target instead of a source (or vice * versa), this can be indicated with a flag to trigger proper shuffling * (e.g. DUK_EMIT_FLAG_B_IS_TARGET). If the value in the slot is not * register/const related at all, the caller must ensure that the raw value * fits into the corresponding slot so as to not trigger shuffling. The * caller must set a "no shuffle" flag to ensure compilation fails if * shuffling were to be triggered because of an internal error. * * For slots B and C the raw slot size is 9 bits but one bit is reserved for * the reg/const indicator. To use the full 9-bit range for a raw value, * shuffling must be disabled with the DUK_EMIT_FLAG_NO_SHUFFLE_{B,C} flag. * Shuffling is only done for A, B, and C slots, not the larger BC or ABC slots. * * There is call handling specific understanding in the A-B-C emitter to * convert call setup and call instructions into indirect ones if necessary.
41029. if num_values > 0, MPUTARR emitted by outer loop after break
41030. **comment:** XXX: would be nice to omit this jump when the jump is not * reachable, at least in the obvious cases (such as the case * ending with a 'break'. * * Perhaps duk_parse_stmt() could provide some info on whether * the statement is a "dead end"? * * If implemented, just set pc_prevstmt to -1 when not needed.
label: code-design

41031. **comment:** Two temporaries are preallocated here for variants 3 and 4 which need * registers which are never clobbered by expressions in the loop * (concretely: for the enumerator object and the next enumerated value). * Variants 1 and 2 "release" these temps.
label: code-design

41032. DUK_USE_NONSTD_FUNC_STMT
41033. DUK_TOK_STRING
41034. stmt has non-empty value
41035. for array and object literals
41036. When LHS is not register bound, always go through a * temporary. No optimization for top level assignment.
41037. non-strict eval: env is caller's env or global env (direct vs. indirect call) * global code: env is global env
41038. **comment:** XXX: use the exactly same arithmetic function here as in executor
label: code-design

41039. -1 == not set, -2 == pending (next statement list)
41040. Explicit check for fastint downgrade.
41041. DUK_TOK_RETURN
41042. **comment:** _Varmap: omitted if function is guaranteed not to do slow path identifier * accesses or if it would turn out to be empty of actual register mappings * after a cleanup. When debugging is enabled, we always need the varmap to * be able to lookup variables at any point.
label: code-design

41043. We could rely on max temp/const checks: if they don't exceed BC * limit, nothing here can either (just asserts would be enough). * Currently we check for the limits, which provides additional * protection against creating invalid bytecode due to compiler * bugs.

41044. **comment:** XXX: Currently function source code is not stored, as it is not * required by the standard. Source code should not be stored by * default (user should enable it explicitly), and the source should * probably be compressed with a trivial text compressor; average * compression of 20-30% is quite easy to achieve even with a trivial * compressor (RLE + backwards lookup). * * Debugging needs source code to be useful: sometimes input code is * not found in files as it may be generated and then eval()'d, given * by dynamic C code, etc. * * Other issues: * * - Need tokenizer indices for start and end to substring * - Always normalize function declaration part? * - If we keep _Formals, only need to store body
label: code-design

41045. require initializer for var/const
41046. DUK_TOK_FALSE
41047. fall-through control flow patchup; note that pc_prevstmt may be * < 0 (i.e. no case clauses), in which case this is a no-op.
41048. * Detect iteration statements; if encountered, establish an * empty label.
41049. **comment:** Should never happen but avoid infinite loop just in case.
label: code-design

41050. indirect opcode follows direct
41051. statement parsing
41052. * Function declarations
41053. **comment:** * Parse variant 1 or 2. The first part expression (which differs * in the variants) has already been parsed and its code emitted. * * reg_temps + 0: unused * reg_temps + 1: unused
label: code-design

41054. Valstack should suffice here, required on function valstack init
41055. reg_res should be smallest possible
41056. pushes function template
41057. **comment:** Pick a destination register. If either base value or key * happens to be a temp value, reuse it as the destination. ** XXX: The temp must be a "mutable" one, i.e. such that no * other expression is using it anymore. Here this should be * the case because the value of a property access expression * is neither the base nor the key, but the lookup result.
label: code-design
41058. Note: although there is no 'undefined' literal, undefined * values can occur during compilation as a result of e.g. * the 'void' operator.
41059. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()!
41060. Identifier, i.e. don't allow reserved words
41061. **comment:** * Emit a final RETURN. ** It would be nice to avoid emitting an unnecessary "return" opcode * if the current PC is not reachable. However, this cannot be reliably * detected; even if the previous instruction is an unconditional jump, * there may be a previous jump which jumps to current PC (which is the * case for iteration and conditional statements, for instance).
label: code-design
41062. **comment:** Parse an inner function, adding the function template to the current function's * function table. Return a function number to be used by the outer function. ** Avoiding O(depth^2) inner function parsing is handled here. On the first pass, * compile and register the function normally into the 'funcs' array, also recording * a lexer point (offset/line) to the closing brace of the function. On the second * pass, skip the function and return the same 'fnum' as on the first pass by using * a running counter. ** An unfortunate side effect of this is that when parsing the inner function, almost * nothing is known of the outer function, i.e. the inner function's scope. We don't * need that information at the moment, but it would allow some optimizations if it * were used.
label: code-design
41063. Slot B
41064. **comment:** * Emit initializers in sets of maximum max_init_values. * Corner cases such as single value initializers do not have * special handling now. ** Elided elements must not be emitted as 'undefined' values, * because such values would be enumerable (which is incorrect). * Also note that trailing elisions must be reflected in the * length of the final array but cause no elements to be actually * inserted.
label: code-design
41065. XXX: to util
41066. Specific assumptions for opcode numbering.
41067. _Pc2line
41068. EQUALITY EXPRESSION
41069. register declarations in first pass
41070. DUK_TOK_EQUALSIGN
41071. **comment:** No catch variable, e.g. a try-finally; replace LDCONST with * NOP to avoid a bogus LDCONST.
label: code-design
41072. useful for patching jumps later
41073. name
41074. -> [... func name]
41075. inherit initial strictness from parent
41076. **comment:** XXX: this illustrates that a C catchpoint implemented using duk_safe_call() * is a bit heavy at the moment. The wrapper compiles to ~180 bytes on x64. * Alternatives would be nice.
label: code-design
41077. DUK_TOK_WHILE
41078. Use special opcodes to load short strings
41079. temp variable to pass constants and flags to shared code
41080. truncated in case pass 3 needed
41081. reg
41082. * Second (and possibly third) pass. ** Generate actual code. In most cases the need for shuffle * registers is detected during pass 1, but in some corner cases * we'll only detect it during pass 2 and a third pass is then * needed (see GH-115).
41083. DUK_TOK_MUL_EQ
41084. **comment:** maintain highest 'used' temporary, needed to figure out nregs of function
label: code-design
41085. Slot C is used in a non-standard fashion (range of regs), * emitter code has special handling for it (must not set the * "no shuffle" flag).
41086. Catch attempts to use out-of-range reg/const. Without this * check Duktape 0.12.0 could generate invalid code which caused * an assert failure on execution. This error is triggered e.g. * for functions with a lot of constants and a try-catch statement. * Shuffling or opcode semantics change is needed to fix the issue. * See: test-bug-trycatch-many-constants.js.
41087. DUK_TOK_DO
41088. **comment:** XXX: match flag is awkward, rework
label: code-design
41089. continue matched this label -- we can only continue if this is the empty * label, for which duplication is allowed, and thus there is hope of * finding a match deeper in the label stack.
41090. Evaluate RHS only when LHS is safe.
41091. DUK_TOK_VOID
41092. DUK_USE_PC2LINE
41093. Special handling for call setup instructions. The target * is expressed indirectly, but there is no output shuffling.
41094. no catchers
41095. A number can be loaded either through a constant, using * LDINT, or using LDINT+LDINTX. LDINT is always a size win, * LDINT+LDINTX is not if the constant is used multiple times. * Currently always prefer LDINT+LDINTX over a double constant.
41096. DUK_USE_NONSTD_FUNC_SOURCE_PROPERTY
41097. * Variant 4
41098. * Preliminaries
41099. is_extra
41100. Slot A
41101. terminates expression; e.g. post-increment/-decrement
41102. * Parser duk__advance() token eating functions
41103. inline string concatenation
41104. no operators
41105. * Variable declarations. ** Unlike function declarations, variable declaration values don't get * assigned on entry. If a binding of the same name already exists, just * ignore it silently.
41106. Strict: never allow function declarations outside top level.
41107. **comment:** * XXX: for now, indicate that an expensive catch binding * declarative environment is always needed. If we don't * need it, we don't need the const_varname either.
label: code-design
41108. **comment:** cleanup varmap from any null entries, compact it, etc; returns number * of final entries after cleanup.
label: code-design
41109. * comp_ctx->curr_func is now ready to be converted into an actual * function template.
41110. BITWISE EXPRESSIONS
41111. Used for minimal 'const': initializer required.
41112. DUK_TOK_CONST
41113. DUK_TOK_RPAREN

41114. Slot C
41115. true
41116. comma after a value, expected
41117. general temp register
41118. **comment:** XXX: common reg allocation need is to reuse a sub-expression's temp reg, * but only if it really is a temp. Nothing fancy here now.
label: code-design
41119. DUK_TOK_SUPER
41120. **comment:** * Declaration binding instantiation conceptually happens when calling a * function; for us it essentially means that function prologue. The * conceptual process is described in E5 Section 10.5. ** We need to keep track of all encountered identifiers to (1) create an * identifier-to-register map ("varmap"); and (2) detect duplicate * declarations. Identifiers which are not bound to registers still need * to be tracked for detecting duplicates. Currently such identifiers * are put into the varmap with a 'null' value, which is later cleaned up. ** To support functions with a large number of variable and function * declarations, registers are not allocated beyond a certain limit; * after that limit, variables and functions need slow path access. * Arguments are currently always register bound, which imposes a hard * (and relatively small) argument count limit. ** Some bindings in E5 are not configurable (= deletable) and almost all * are mutable (writable). Exceptions are: ** - The 'arguments' binding, established only if no shadowing argument * or function declaration exists. We handle 'arguments' creation * and binding through an explicit slow path environment record. ** - The "name" binding for a named function expression. This is also * handled through an explicit slow path environment record.
label: code-design
41121. DUK_TOK_NULL
41122. step 4.a
41123. Check whether statement list ends.
41124. Setup state. Initial ivalue is 'undefined'.
41125. DUK_TOK_STATIC
41126. first coerce to a plain value
41127. **comment:** XXX: coerce to regs? it might be better for enumeration use, where the * same PROP ivalue is used multiple times. Or perhaps coerce PROP further * there?
label: code-design
41128. res' contains expression value
41129. invalidates tv1, tv2
41130. rc_varname and reg_varbind are ignored here
41131. **comment:** Extraop arithmetic opcodes must have destination same as * first source. If second source matches destination we need * a temporary register to avoid clobbering the second source. ** XXX: change calling code to avoid this situation in most cases.
label: code-design
41132. reg(ignored)
41133. bytecode limits
41134. DUK_TOK_SEQ
41135. XXX: clarify on when and where DUK_CONST_MARKER is allowed
41136. bump up "allocated" reg count, just in case
41137. overwrite any previous binding of the same name; the effect is * that last argument of a certain name wins.
41138. A bunch of helpers (for size optimization) that combine duk_expr()/duk_exprtop() * and result conversions. ** Each helper needs at least 2-3 calls to make it worth while to wrap.
41139. DUK_TOK_INSTANCEOF
41140. * Parse an individual source element (top level statement) or a statement. ** Handles labeled statements automatically (peeling away labels before * parsing an expression that follows the label(s)). ** Upon entry, 'curr_tok' contains the first token of the statement (parsed * in "allow regexp literal" mode). Upon exit, 'curr_tok' contains the first * token following the statement (if the statement has a terminator, this is * the token after the terminator).
41141. DUK_TOK_IF
41142. evaluate to plain value, no forced register (temp/bound reg both ok)
41143. JUMP L1 omitted
41144. e.g. DUK_OP_POSTINCR
41145. **comment:** XXX: many operations actually want toforcedtemp() -- brand new temp?
label: code-design
41146. * Debug dumping
41147. * The switch statement is pretty messy to compile. * See the helper for details.
41148. DUK_TOK_ENUM
41149. when key is NULL, value is garbage so no need to set
41150. knock back "next temp" to this whenever possible
41151. **comment:** * Parser control values for tokens. The token table is ordered by the * DUK_TOK_XXX defines. ** The binding powers are for lbp() use (i.e. for use in led() context). * Binding powers are positive for typing convenience, and bits at the * top should be reserved for flags. Binding power step must be higher * than 1 so that binding power "lbp - 1" can be used for right associative * operators. Currently a step of 2 is used (which frees one more bit for * flags).
label: code-design
41152. * Detected a directive
41153. entry_top + 3
41154. ASSIGNMENT EXPRESSION
41155. if a site already exists, nop: max one label site per statement
41156. first value: comma must not precede the value
41157. DUK_TOK_CATCH
41158. rethrow
41159. key encountered as a plain property
41160. e.g. DUK_OP_POSTINCV
41161. * Label handling ** Labels are initially added with flags prohibiting both break and continue. * When the statement type is finally uncovered (after potentially multiple * labels), all the labels are updated to allow/prohibit break and continue.
41162. sufficient for keeping temp reg numbers in check
41163. Note: when parsing a formal list in non-strict context, e.g. * "implements" is parsed as an identifier. When the function is * later detected to be strict, the argument list must be rechecked * against a larger set of reserved words (that of strict mode). * This is handled by duk_parse_func_body(). Here we recognize * whatever tokens are considered reserved in current strictness * (which is not always enough).
41164. **comment:** Update function min/max line from current token. Needed to improve * function line range information for debugging, so that e.g. opening * curly brace is covered by line range even when no opcodes are emitted * for the line containing the brace.
label: code-design
41165. pass2 allocation handles this
41166. DUK_TOK_IN
41167. jump to false
41168. require a short (8-bit) reg/const which fits into bytecode B/C slot
41169. **comment:** Sanity workaround for handling functions with a large number of * constants at least somewhat reasonably. Otherwise checking whether * we already have the constant would grow very slow (as it is O(N^2)).
label: code-design
41170. **comment:** XXX: duk_set_length
label: code-design
41171. DUK_TOK_BOR_EQ

41172. **comment:** XXX: Add a proper condition. If formals list is omitted, recheck * handling for 'length' in duk_js_push_closure(); it currently relies * on _Formals being set. Removal may need to be conditional to debugging * being enabled/disabled too.
label: code-design

41173. NB: must accept reserved words as property name

41174. +1, right after inserted jump

41175. * After arguments, allocate special registers (like shuffling temps)

41176. * Variant 1

41177. DUK_TOK_DEFAULT

41178. temp reset is not necessary after duk__parse_stmt(), which already does it

41179. same handling for identifiers and strings

41180. e.g. DUK_OP_PREINCP

41181. decl name

41182. LOGICAL EXPRESSIONS

41183. jump is inserted here

41184. Resolve 'res' directly into the LHS binding, and use * that as the expression value if safe. If not safe, * resolve to a temp/const and copy to LHS.

41185. * Switch is pretty complicated because of several conflicting concerns: * * - Want to generate code without an intermediate representation, * i.e., in one go * * - Case selectors are expressions, not values, and may thus e.g. throw * exceptions (which causes evaluation order concerns) * * - Evaluation semantics of case selectors and default clause need to be * carefully implemented to provide correct behavior even with case value * side effects * * - Fall through case and default clauses; avoiding dead JUMPs if case * ends with an unconditional jump (a break or a continue) * * - The same case value may occur multiple times, but evaluation rules * only process the first match before switching to a "propagation" mode * where case values are no longer evaluated * * See E5 Section 12.11. Also see doc/compiler.rst for compilation * discussion.

41186. Ensure argument name is not a reserved word in current * (final) strictness. Formal argument parsing may not * catch reserved names if strictness changes during * parsing. * * We only need to do this in strict mode because non-strict * keyword are always detected in formal argument parsing.

41187. const flag for C

41188. unary minus

41189. number of pairs in current MPUTOBJ set

41190. Tear down state.

41191. **comment:** XXX: where to release temp regs in intermediate expressions? * e.g. 1+2+3 -> don't inflate temp register count when parsing this. * that particular expression temp regs can be forced here.
label: code-design

41192. expression parsing helpers

41193. [...] key_obj key]

41194. **comment:** XXX: actually single step levels would work just fine, clean up
label: code-design

41195. one token

41196. DUK_TOK_BNOT

41197. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

41198. * Peephole optimizer for finished bytecode. * * Does not remove opcodes; currently only straightens out unconditional * jump chains which are generated by several control structures.

41199. **comment:** alloc temp just in case, to update max temp
label: code-design

41200. is_setget

41201. DUK_TOK_COMMA

41202. Shuffle decision not changed.

41203. DUK__ALLOW_AUTO_SEMI_ALWAYS workaround

41204. temp_start+0 = key, temp_start+1 = closure

41205. forced_reg

41206. DUK_TOK_FINALY

41207. 'data' (and everything in it) is reachable through h_res now

41208. arg2 would be clobbered so reassign it to a temp.

41209. DUK_TOK_REGEXP

41210. Note: this is correct even for default clause statements: * they participate in 'fall-through' behavior even if the * default clause is in the middle.

41211. DUK_TOK_PACKAGE

41212. * Peephole optimize JUMP chains.

41213. temps

41214. to body

41215. no match, next case

41216. * Directive prologue tracking.

41217. DUK_TOK_IDENTIFIER

41218. * Misc helpers

41219. finished jump

41220. POSTFIX EXPRESSION

41221. base

41222. DUK_TOK_PUBLIC

41223. variable name reg/const, if variable not register-bound

41224. Eating a left curly; regexp mode is allowed by left curly * based on duk__token_lbp[] automatically.

41225. Get the value represented by an duk_ispec to a register or constant. * The caller can control the result by indicating whether or not: * * (1) a constant is allowed (sometimes the caller needs the result to * be in a register) * * (2) a temporary register is required (usually when caller requires * the register to be safely mutable; normally either a bound * register or a temporary register are both OK) * * (3) a forced register target needs to be used * * Bytecode may be emitted to generate the necessary value. The return * value is either a register or a constant.

41226. NEXTENUM jump slot: executed when enum finished

41227. single string token

41228. Insert an empty jump in the middle of code emitted earlier. This is * currently needed for compiling for-in.

41229. DUK_TOK_ARSHIFT

41230. eat the right paren

41231. **comment:** XXX: possibly incorrect handling of empty expression
label: code-design

41232. **comment:** XXX: what about statements which leave a half-cooked value in 'res' * but have no stmt value? Any such statements?
label: code-design

41233. MAXVAL is inclusive

41234. "release" preallocated temps since we won't need them

41235. temp reg value for start of loop

41236. is_decl

41237. unsigned

41238. Expression, ']' terminates

41239. Don't allow a constant for the object (even for a number etc), as * it goes into the 'A' field of the opcode.

41240. no need to coerce
41241. func limits
41242. 'res' is used for "left", and 'tmp' for "right"
41243. [... template]
41244. end switch (tok)
41245. XXX: shared handling for 'duk__expr_lhs'?
41246. keep func->h_funcs; inner functions are not reparsed to avoid O(depth^2) parsing
41247. num_values and temp_start reset at top of outer loop
41248. Fix for https://github.com/svaarala/duktape/issues/155: * If 'default' is first clause (detected by pc_prevcase < 0) * we need to ensure we stay in the matching chain.
41249. allow_source_elem
41250. duplicate/invalid key checks; returns 1 if syntax error
41251. eat 'in'
41252. Name will be filled from function expression, not by caller. * This case is used by Function constructor and duk_compile() * API with the DUK_COMPILE_FUNCTION option.
41253. * Detected label
41254. * Case clause. * * Note: cannot use reg_case as a temp register (for SEQ target) * because it may be a constant.
41255. DUK_TOK_SUB
41256. Default implicit return value.
41257. elision - flush
41258. temp reg for key literal
41259. * Parse a case or default clause.
41260. **comment:** Try to optimize X <op>= Y for reg-bound * variables. Detect side-effect free RHS * narrowly by seeing whether it emits code. * If not, rewind the code emitter and overwrite * the unnecessary temp reg load.
label: code-design
41261. DUK_TOK_THIS
41262. Unlike break/continue, throw statement does not allow an empty value.
41263. DUK_TOK_VAR
41264. * Expression parsing: duk__expr_nud(), duk__expr_led(), duk__expr_lbp(), and helpers. * * - duk__expr_nud(): ("null denotation"): process prev_token as a "start" of an expression (e.g. literal) * - duk__expr_led(): ("left denotation"): process prev_token in the "middle" of an expression (e.g. operator) * - duk__expr_lbp(): ("left-binding power"): return left-binding power of curr_token
41265. max # of key-value pairs initialized in one MPUTOBJ set
41266. expect_token
41267. Note: if the reg_temp load generated shuffling * instructions, we may need to rewind more than * one instruction, so use explicit PC computation.
41268. **comment:** Function declaration for global/eval code is emitted even * for duplicates, because of E5 Section 10.5, step 5.e of * E5.1 (special behavior for variable bound to global object). * * DECLVAR will not re-declare a variable as such, but will * update the binding value.
label: code-design
41269. Stack top contains plain value
41270. make current token the previous; need to fiddle with valstack "backing store"
41271. comp_ctx->lex.input and comp_ctx->lex.input_length filled by caller
41272. >>
41273. trailing elisions?
41274. unary plus of a number is identity
41275. no comma
41276. advance to get one step of lookup
41277. right paren eaten
41278. MPUTARR emitted by outer loop
41279. DUK_TOK_NUMBER
41280. **comment:** * XXX: attempt to get the call result to "next temp" whenever * possible to avoid unnecessary register shuffles. * * XXX: CSPROP (and CSREG) can overwrite the call target register, and save one temp, * if the call target is a temporary register and at the top of the temp reg "stack".
label: code-design
41281. jump for 'finally' case or end (if no finally)
41282. * Identifier handling
41283. DUK_TOK_SET
41284. Code is not accepted before the first case/default clause
41285. Note: 'q_instr' is still used below
41286. start array index of current MPUTARR set
41287. expect_eof
41288. XXX: request a "last statement is terminal" from duk__parse_stmt() and duk__parse_stmts(); * we could avoid the last RETURN if we could ensure there is no way to get here * (directly or via a jump)
41289. [... key_obj key key flags]
41290. rbp_flags
41291. use 'left' as a temp
41292. don't allow const
41293. Note: special DUK_EMIT_FLAG_B_IS_TARGETSOURCE * used to indicate that B is both a source and a * target register. When shuffled, it needs to be * both input and output shuffled.
41294. '++' or '--' in a post-increment/decrement position, * and a LineTerminator occurs between the operator and * the preceding expression. Force the previous expr * to terminate, in effect treating e.g. "a,b\n++" as * "a,b;++" (= SyntaxError).
41295. DUK_TOK_BAND
41296. if highest bit of a register number is set, it refers to a constant instead
41297. no need to reset temps, as we're finished emitting code
41298. -> [...]
41299. 'typeof' must handle unresolvable references without throwing * a ReferenceError (E5 Section 11.4.3). Register mapped values * will never be unresolvable so special handling is only required * when an identifier is a "slow path" one.
41300. Expression_opt
41301. * The catch variable must be updated to reflect the new allocated * register for the duration of the catch clause. We need to store * and restore the original value for the varmap entry (if any).
41302. * Parse variant 3 or 4. * * For variant 3 (e.g. "for (A in C) D;") the code for A (except the * final property/variable write) has already been emitted. The first * instruction of that code is at pc_v34_lhs; a JUMP needs to be inserted * there to satisfy control flow needs. * * For variant 4, if the variable declaration had an initializer * (e.g. "for (var A = B in C) D;") the code for the assignment * (B) has already been emitted. * * Variables set before entering here: * * pc_v34_lhs: insert a "JUMP L2" here (see doc/compiler.rst example). * reg_temps + 0: iteration target value (written to LHS) * reg_temps + 1: enumerator object
41303. still in prologue
41304. DUK_TOK_DIV_EQ
41305. * Arguments check
41306. **comment:** The line number tracking is a bit inconsistent right now, which * affects debugger accuracy. Mostly call sites emit opcodes when * they have parsed a token (say a terminating semicolon) and called * duk__advance(). In this case the line number of the previous * token is the most accurate one (except in prologue where * prev_token.start_line is 0). This is probably not 100% correct * right now.
label: code-design
41307. bc

41308. parse new token
41309. Parse enumeration target and initialize enumerator. For 'null' and 'undefined', * INITENUM will creates a 'null' enumerator which works like an empty enumerator * (E5 Section 12.6.4, step 3). Note that INITENUM requires the value to be in a * register (constant not allowed).
41310. numargs
41311. **comment:** XXX: some code might benefit from DUK__SETTEMP_IFTEMP(ctx,x)
 label: code-design
41312. **comment:** Very minimal inlining to handle common idioms '!0' and '!1', * and also boolean arguments like '!false' and '!true'.
 label: code-design
41313. DUK_TOK_ELSE
41314. nret
41315. See MPUTOBJ comments above.
41316. preincrement and predecrement
41317. Need a short reg/const, does not have to be a mutable temp.
41318. Expression, terminates at a ')'
41319. convert duk_compiler_func into a function template, leaving the result * on top of stack.
41320. Special handling for CALL/NEW/MPUTOBJ/MPUTARR shuffling. * For each, slot B identifies the first register of a range * of registers, so normal shuffling won't work. Instead, * an indirect version of the opcode is used.
41321. DUK_TOK_FOR
41322. DUK_TOK_DEBUGGER
41323. patched later
41324. Duplicate (shadowing) labels are not allowed, except for the empty * labels (which are used as default labels for switch and iteration * statements). * * We could also allow shadowing of non-empty pending labels without any * other issues than breaking the required label shadowing requirements * of the E5 specification, see Section 12.12.
41325. **comment:** XXX: depend on available temps?
 label: code-design
41326. 'reg_varbind' is the operation result and can also * become the expression value for top level assignments * such as: "var x; x += y;"
41327. last array index explicitly initialized, +1
41328. Output shuffling: only one output register is realistically possible. * * (Zero would normally be an OK marker value: if the target register * was zero, it would never be shuffled. But with DUK_USE_SHUFFLE_TORTURE * this is no longer true, so use -1 as a marker instead.)
41329. **comment:** * Three possible element formats: * 1) PropertyName : AssignmentExpression * 2) get PropertyName () { FunctionBody } * 3) set PropertyName (PropertyNameSetParameterList) { FunctionBody } * * PropertyName can be IdentifierName (includes reserved words), a string * literal, or a number literal. Note that IdentifierName allows 'get' and * 'set' too, so we need to look ahead to the next token to distinguish: * * { get : 1 } * * and * * { get foo() { return 1 } } * { get get { return 1 } } // 'get' as getter propertyname * * Finally, a trailing comma is allowed. * * Key name is coerced to string at compile time (and ends up as a * string constant) even for numeric keys (e.g. "{1:'foo'}"). * These could be emitted using e.g. LDINT, but that seems hardly * worth the effort and would increase code size.
 label: code-design
41330. MPUTOBJ emitted by outer loop
41331. MultiplicativeExpression
41332. prev_token.start_offset points to the closing brace here; when skipping * we're going to reparse the closing brace to ensure semicolon insertion * etc work as expected.
41333. more initializers
41334. need side effects, not value
41335. FunctionDeclaration: not strictly a statement but handled as such. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().
41336. * Check function name validity now that we know strictness. * This only applies to function declarations and expressions, * not setter/getter name. * * See: test-dev-strict-mode-boundary.js
41337. allow automatic semicolon even without lineterm (compatibility)
41338. binding power "levels" (see doc/compiler.rst)
41339. * Cleanup: restore original function, restore valstack state.
41340. DUK_TOK_QUESTION
41341. **comment:** XXX: macro smaller than call?
 label: code-design
41342. Note: negative pc values are ignored when patching jumps, so no explicit checks needed
41343. result reg
41344. Coerce an duk_ivalue to a 'plain' value by generating the necessary * arithmetic operations, property access, or variable access bytecode. * The duk_ivalue argument ('x') is converted into a plain value as a * side effect.
41345. eat comma
41346. **comment:** XXX: source code property
 label: code-design
41347. this is needed for regexp mode
41348. * Compile input string into an executable function template without * arguments. * * The string is parsed as the "Program" production of EcmaScript E5. * Compilation context can be either global code or eval code (see E5 * Sections 14 and 15.1.2.1). * * Input stack: [... filename] * Output stack: [... func_template]
41349. [... varname]
41350. preinitialize lexer state partially
41351. jump to next case clause
41352. const
41353. Code emission flags, passed in the 'opcode' field. Opcode + flags * fit into 16 bits for now, so use duk_small_uint.t.
41354. eat 'if'
41355. key encountered as a getter
41356. **comment:** XXX: macros
 label: code-design
41357. must have catch and/or finally
41358. DUK_TOK_GT
41359. 'new' MemberExpression
41360. DUK_TOK_BXOR
41361. * Store lexer position for a later rewind
41362. regCatch+0 and regCatch+1 are reserved for TRYCATCH
41363. Function name is an Identifier (not IdentifierName), but we get * the raw name (not recognizing keywords) here and perform the name * checks only after pass 1.
41364. DUK_TOK_DECREMENT
41365. **comment:** The target for this LDCONST may need output shuffling, but we assume * that 'pc_ldconst' will be the LDCONST that we can patch later. This * should be the case because there's no input shuffling. (If there's * no catch clause, this LDCONST will be replaced with a NOP.)
 label: code-design
41366. default case control flow patchup; note that if pc_prevcase < 0 * (i.e. no case clauses), control enters default case automatically.
41367. At this point 'res' holds the potential expression value. * It can be basically any ivalue here, including a reg-bound * identifier (if code above deems it safe) or a unary/binary * operation. Operations must be resolved to a side effect free * plain value, and the side effects must happen exactly once.
41368. require a (mutable) temporary as a result (or a const if allowed)
41369. DUK_OP_NONE marks a 'plain' assignment
41370. keep in onvalstack, use borrowed ref below

41371. * Parse code after the clause. Possible terminators are * 'case', 'default', and ')'. ** Note that there may be no code at all, not even an empty statement, * between case clauses. This must be handled just like an empty statement * (omitting seemingly pointless JUMPs), to avoid situations like * test-bug-case-fallthrough.js.
41372. res.x1 -> res.x2
41373. number of values in current MPUTARR set
41374. <<
41375. to exit
41376. **comment:** has inner functions which may slow access (XXX: this can be optimized by looking at the inner functions)
 label: code-design
41377. Parse formals.
41378. **comment:** XXX: need to determine LHS type, and check that it is LHS compatible
 label: code-design
41379. getter/setter
41380. false
41381. register bound variables are non-configurable -> always false
41382. nargs
41383. XXX: init or assert catch depth etc -- all values
41384. **comment:** Flags for intermediate value coercions. A flag for using a forced reg * is not needed, the forced_reg argument suffices and generates better * code (it is checked as it is used).
 label: code-design
41385. copy bytecode instructions one at a time
41386. LHS is already side effect free
41387. * Variant 3
41388. DUK_TOK_IMPORT
41389. XXX: range assert
41390. points to LABEL; pc+1 = jump site for break; pc+2 = jump site for continue
41391. * On second pass, skip the function.
41392. **comment:** XXX: hack, remove when const lookup is not O(n)
 label: code-design
41393. token closes expression, e.g. ')', '['
41394. fileName
41395. chain jumps for case * evaluation and checking
41396. needed for line number tracking (becomes prev_token.start_line)
41397. * 'key_obj' tracks keys encountered so far by associating an * integer with flags with already encountered keys. The checks * below implement E5 Section 11.1.5, step 4 for production: * * PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment
41398. function call
41399. DUK_TOK_MUL
41400. **comment:** XXX: opcode specific assertions on when consts are allowed
 label: code-design
41401. const limits
41402. don't coerce yet to a plain value (variant 3 needs special handling)
41403. The issues below can be solved with better flags
41404. [...] res]
41405. If forced reg, use it as destination. Otherwise try to * use either coerced ispec if it is a temporary. ** When using extraops, avoid reusing arg2 as dest because that * would lead to an LDREG shuffle below. We still can't guarantee * dest != arg2 because we may have a forced_reg.
41406. continue jump
41407. LeftHandSideExpression does not allow empty expression
41408. Don't allow negative zero as it will cause trouble with * LDINT+LDINTX. But positive zero is OK.
41409. First evaluate LHS fully to ensure all side effects are out.
41410. Object literal set/get functions have a name (property * name) but must not have a lexical name binding, see * test-bug-getset-func-name.js.
41411. allow source elements
41412. **comment:** XXX: This now coerces an identifier into a GETVAR to a temp, which * causes an extra LDREG in call setup. It's sufficient to coerce to a * unary ivalue?
 label: code-design
41413. eat 'function'
41414. one token before
41415. keep in valstack
41416. Limit checks for bytecode byte size and line number.
41417. Parse a single statement. ** Creates a label site (with an empty label) automatically for iteration * statements. Also "peels off" any label statements for explicit labels.
41418. normal key/value
41419. Note: escaped characters differentiate directives
41420. Setting "no shuffle A" is covered by the assert, but it's needed * with DUK_USE_SHUFFLE_TORTURE.
41421. reject 'in' token (used for for-in)
41422. Note: 0xff != DUK_BC_B_MAX
41423. Function expression. Note that any statement beginning with 'function' * is handled by the statement parser as a function declaration, or a * non-standard function expression/statement (or a SyntaxError). We only * handle actual function expressions (occurring inside an expression) here. ** O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().
41424. special RegExp literal handling after IdentifierName
41425. **comment:** XXX: awkward, especially the bunch of separate output values -> output struct?
 label: code-design
41426. statement is guaranteed to be terminal (control doesn't flow to next statement)
41427. fall-through jump to next code of next case (backpatched)
41428. Use a fast break/continue when possible. A fast break/continue is * just a jump to the LABEL break/continue jump slot, which then jumps * to an appropriate place (for break, going through ENDLABEL correctly). * The peephole optimizer will optimize the jump to a direct one.
41429. * The length comparisons are present to handle * strings like "use strict\u0000foo" as required.
41430. directive prologue status at entry
41431. * Function declaration, function expression, or (non-standard) * function statement. ** The E5 specification only allows function declarations at * the top level (in "source elements"). An ExpressionStatement * is explicitly not allowed to begin with a "function" keyword * (E5 Section 12.4). Hence any non-error semantics for such * non-top-level statements are non-standard. Duktape semantics * for function statements are modelled after V8, see * test-dev-func-decl-outside-top.js.
41432. Parse argument list. Arguments are written to temps starting from * "next temp". Returns number of arguments parsed. Expects left paren * to be already eaten, and eats the right paren before returning.
41433. * Variant 1 or 3
41434. temp object for tracking / detecting duplicate keys
41435. Opcode slot C is used in a non-standard way, so shuffling * is not allowed.
41436. DUK_TOK_SEMICOLON
41437. DUK_TOK_RSHIFT
41438. * Size-optimized pc->line mapping.

41439. Note: if target_pc1 == i, we'll optimize a jump to itself. * This does not need to be checked for explicitly; the case * is rare and max iter breaks us out.
41440. DUK_TOK_EQ
41441. code emission
41442. strict mode restrictions (E5 Section 12.2.1)
41443. result in csreg
41444. **comment:** Technically return value is not needed because INVLHS will * unconditionally throw a ReferenceError. Coercion is necessary * for proper semantics (consider ToNumber() called for an object). * Use DUK_EXTRAOP_UNP with a dummy register to get ToNumber().
 label: code-design
41445. decl type
41446. **comment:** The entries can either be register numbers or 'null' values. * Thus, no need to DECREF them and get side effects. DECREF'ing * the keys (strings) can cause memory to be freed but no side * effects as strings don't have finalizers. This is why we can * rely on the object properties not changing from underneath us.
 label: code-design
41447. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.
41448. PropertyName -> IdentifierName | StringLiteral | NumericLiteral
41449. top-down expression parser
41450. Lookup an identifier name in the current varmap, indicating whether the * identifier is register-bound and if not, allocating a constant for the * identifier name. Returns 1 if register-bound, 0 otherwise. Caller can * also check (out_reg_varbind >= 0) to check whether or not identifier is * register bound. The caller must NOT use out_rc_varname at all unless * return code is 0 or out_reg_varbind is < 0; this is because out_rc_varname * is unsigned and doesn't have a "unused" / none value.
41451. E5 Section steps 7-8
41452. **comment:** XXX: unoptimal use of temps, resetting
 label: code-design
41453. Any 'res' will do.
41454. is_decl
41455. varmap is already in comp_ctx->curr_func.varmap_idx
41456. DUK_TOK_SWITCH
41457. eat 'try'
41458. b
41459. Note: 0xff != DUK_BC_C_MAX
41460. _Source
41461. Lenient: allow function declarations outside top level in * non-strict mode but reject them in strict mode.
41462. start variant 3/4 left-hand-side code (L1 in doc/compiler.rst example)
41463. COMMA
41464. Parse statements until a closing token (EOF or '}') is found.
41465. XXX: if assertions enabled, walk through all valid PCs * and check line mapping.
41466. The token in the switch has already been eaten here
41467. DUK_TOK_LAND
41468. currently all labels accept a break, so no explicit check for it now
41469. DUK_TOK_NEW
41470. force_no_namebind
41471. DUK_TOK_RBRACKET
41472. * Setup call: target and 'this' binding. Three cases: * * 1. Identifier base (e.g. "foo()") * 2. Property base (e.g. "foo.bar()") * 3. Register base (e.g. "foo()"); i.e. when a return value is a function
41473. statement does not terminate directive prologue
41474. SHIFT EXPRESSION
41475. **comment:** XXX: awkward and bloated asm -- use faster internal accesses
 label: code-design
41476. approximation, close enough
41477. DUK_TOK_LOR
41478. **comment:** XXX: specific getter
 label: code-design
41479. **comment:** Then evaluate RHS fully (its value becomes the expression value too). * Technically we'd need the side effect safety check here too, but because * we always throw using INVLHS the result doesn't matter.
 label: code-design
41480. First we need to insert a jump in the middle of previously * emitted code to get the control flow right. No jumps can * cross the position where the jump is inserted. See doc/compiler.rst * for discussion on the intricacies of control flow and side effects * for variants 3 and 4.
41481. * Emit initializers in sets of maximum max_init_pairs keys. * Setter/getter is handled separately and terminates the * current set of initializer values. Corner cases such as * single value initializers do not have special handling now.
41482. intentional overlap with earlier memzero
41483. Match labels starting from latest label because there can be duplicate empty * labels in the label set.
41484. Need to set curr_token.t because lexing regexp mode depends on current * token type. Zero value causes "allow regexp" mode.
41485. chain jumps for 'fall-through' * after a case matches.
41486. assumed to also be PC of "LABEL"
41487. * Main switch for statement / source element type.
41488. * Function name (if any) * * We don't check for prohibited names here, because we don't * yet know whether the function will be strict. Function body * parsing handles this retroactively. * * For function expressions and declarations function name must * be an Identifier (excludes reserved words). For setter/getter * it is a PropertyName which allows reserved words and also * strings and numbers (e.g. "{ get 1() { ... } }").
41489. DUK_TOK_DELETE
41490. DUK_TOK_BOR
41491. DUK_TOK_CASE
41492. DUK_TOK_ADD_EQ
41493. **comment:** XXX: optimize temp reg use
 label: code-design
41494. * Compact the function template.
41495. >>>
41496. PRIMARY EXPRESSIONS
41497. left.x1 -> res.x1
41498. num_args
41499. DUK_TOK_ADD
41500. always terminates led()
41501. allocate to reg only (not const)
41502. skip jump conditionally
41503. prevent duk__expr_led() by using a binding power less than anything valid
41504. default clause matches next statement list (if any)
41505. filter out flags from exptop rbp_flags here to save space
41506. eat 'return'

41507. * Parse a function body or a function-like expression, depending * on flags.
41508. **comment:** must be able to emit code, alloc consts, etc.
 label: code-design
41509. **comment:** We want the argument expression value to go to "next temp" * without additional moves. That should almost always be the * case, but we double check after expression parsing. * * This is not the cleanest possible approach.
 label: code-design
41510. slot B is both a target and a source (used by extraops like DUK_EXTRAOP_INSTOF)
41511. FUNCTION EXPRESSIONS
41512. MULTIPLICATIVE EXPRESSION
41513. may be a string constant
41514. Labels can be used for iteration statements but also for other statements, * in particular a label can be used for a block statement. All cases of a * named label accept a 'break' so that flag is set here. Iteration statements * also allow 'continue', so that flag is updated when we figure out the * statement type.
41515. entry_top + 1
41516. **comment:** NEXTENUM needs a jump slot right after the main instruction. * When the JUMP is taken, output spilling is not needed so this * workaround is possible. The jump slot PC is exceptionally * plumbed through comp_ctx to minimize call sites.
 label: code-design
41517. UNARY EXPRESSIONS
41518. Parse a function-like expression, assuming that 'comp_ctx->curr_func' is * correctly set up. Assumes that curr_token is just after 'function' (or * 'set'/'get' etc).
41519. function: always register bound
41520. a statement following a label cannot be a source element * (a function declaration).
41521. Note: Identifier rejects reserved words
41522. [... filename (temps) func]
41523. DUK_TOK_EOF
41524. temp reg
41525. * Second pass parsing.
41526. spare
41527. tmp -> res
41528. DUK_TOK_LCURLY
41529. DUK_TOK_BREAK
41530. **comment:** XXX: for simple cases like x[y] an unnecessary LDREG is * emitted for the base value; could avoid it if we knew that * the key expression is safe (e.g. just a single literal).
 label: code-design
41531. DUK_TOK_MOD_EQ
41532. E5 Section 11.13.1 (and others) step 4 never matches for prop writes -> no check
41533. DUK_TOK_WITH
41534. These limits are based on bytecode limits. Max temps is limited * by duk_hcompiledfunction nargs/nregs fields being 16 bits.
41535. Note that duk_exprtop() here can clobber any reg above current temp_next, * so any loop variables (e.g. enumerator) must be "preallocated".
41536. **comment:** XXX: need a toplain_ignore() which will only coerce a value to a temp * register if it might have a side effect. Side-effect free values do not * need to be coerced.
 label: code-design
41537. * Parse a statement list. * * Handles automatic semicolon insertion and implicit return value. * * Upon entry, 'curr_tok' should contain the first token of the first * statement (parsed in the "allow regexp literal" mode). Upon exit, * 'curr_tok' contains the token following the statement list terminator * (EOF or closing brace).
41538. * Init object properties * Properties should be added in decreasing order of access frequency. * (Not very critical for function templates.)
41539. DUK_TOK_PERIOD
41540. * ctx->prev_token token to process with duk_expr_led() * ctx->curr_token updated by caller
41541. 'res' must be a plain ival, and not register-bound variable.
41542. not present
41543. ADDITIVE EXPRESSION
41544. function helpers
41545. default case does not exist, or no statements present * after default case: finish case evaluation
41546. expected ival
41547. use temp_next for tracking register allocations
41548. Note: must coerce to a (writable) temp register, so that e.g. "Ix" where x * is a reg-mapped variable works correctly (does not mutate the variable register).
41549. **comment:** may indirectly slow access through a direct eval
 label: code-design
41550. Update all labels with matching label_id.
41551. <
41552. The directive prologue flag is cleared by default so that it is * unset for any recursive statement parsing. It is only "revived" * if a directive is detected. (We could also make directives only * allowed if 'allow_source_elem' was true.)
41553. variable binding register if register-bound (otherwise < 0)
41554. key
41555. **comment:** XXX: to be implemented?
 label: code-design
41556. Property access expressions are critical for correct LHS ordering, * see comments in duk_expr()! * * A conservative approach would be to use duk_ivalue_totempconst() * for 'left'. However, allowing a reg-bound variable seems safe here * and is nice because "foo.bar" is a common expression. If the ivalue * is used in an expression a GETPROP will occur before any changes to * the base value can occur. If the ivalue is used as an assignment * RHS, the assignment code will ensure the base value is safe from * RHS mutation.
41557. * Shared handling of binary operations * * args = (is_extraop << 16) + (opcode << 8) + rbp
41558. just in case
41559. entry_top + 4
41560. key encountered as a setter
41561. continue jump not patched, an INVALID opcode remains there
41562. **comment:** XXX: increase ctx->expr_tokens here for every consumed token * (this would be a nice statistic)?
 label: code-design
41563. **comment:** unused
 label: code-design
41564. current (next) array index
41565. * Push result object and init its flags
41566. step 4.d
41567. duk_hobject_set_length_zero(thr, func->h_funcs);
41568. register binding lookup is based on varmap (even in first pass)
41569. empty statement with an explicit semicolon
41570. one operator (= assign)
41571. * Expression parsing. * * Upon entry to 'expr' and its variants, 'curr_tok' is assumed to be the * first token of the expression. Upon exit, 'curr_tok' will be the first * token not part of the expression (e.g. semicolon terminating an expression * statement).
41572. see above
41573. If an implicit return value is needed by caller, it must be * initialized to 'undefined' because we don't know whether any * non-empty (where "empty" is a continuation type, and different * from an empty statement) statements will be executed. * * However, since 1st pass is a throwaway one, no need to emit * it here.

41574. [... name reg/null] -> [...]
41575. no need to init reg, it will be undefined on entry
41576. must restore reliably before returning
41577. DUK_TOK_THROW
41578. Just flip the single bit.
41579. **comment:** XXX: add support for variables to not be register bound always, to * handle cases with a very large number of variables?
 label: code-design
41580. entry_top + 2
41581. break jump
41582. Property access expressions ('a[b]') are critical to correct * LHS evaluation ordering, see test-dev-assign-eval-order*.js. * We must make sure that the LHS target slot (base object and * key) don't change during RHS evaluation. The only concrete * problem is a register reference to a variable-bound register * (i.e., non-temp). Require temp regs for both key and base. * * Don't allow a constant for the object (even for a number * etc), as it goes into the 'A' field of the opcode.
41583. [... key_obj key key]
41584. allow empty expression
41585. expect_eof
41586. * For/for-in main variants are: * * 1. for (ExpressionNoIn_opt; Expression_opt; Expression_opt) Statement * 2. for (var VariableDeclarationNoIn; Expression_opt; Expression_opt) Statement * 3. for (LeftHandSideExpression in Expression) Statement * 4. for (var VariableDeclarationNoIn in Expression) Statement * * Parsing these without arbitrary lookahead or backtracking is relatively * tricky but we manage to do so for now. * * See doc/compiler.rst for a detailed discussion of control flow * issues, evaluation order issues, etc.
41587. label handling
41588. DUK_TOK_SUB_EQ
41589. Expression
41590. allow negative PCs, behave as a no-op
41591. Important main primitive.
41592. ivalue/ispec helpers
41593. DUK_TOK_BXOR_EQ
41594. code_idx = entry_top + 0
41595. this optimization is important to handle negative literals * (which are not directly provided by the lexical grammar)
41596. name
41597. c
41598. E5 Section 11.13.1 (and others for other assignments), step 4.
41599. DUK_TOK_CONTINUE
41600. eat 'throw'
41601. DUK_TOK_CLASS
41602. * First pass. * * Gather variable/function declarations needed for second pass. * Code generated is dummy and discarded.
41603. Potential direct eval call detected, flag the CALL * so that a run-time "direct eval" check is made and * special behavior may be triggered. Note that this * does not prevent 'eval' from being register bound.
41604. **comment:** XXX: spare handling, slow now
 label: code-design
41605. comma check
41606. pop varname
41607. * Parse function body
41608. * Use current token to decide whether a RegExp can follow. * * We can use either 't' or 't_nores'; the latter would not * recognize keywords. Some keywords can be followed by a * RegExp (e.g. "return"), so using 't' is better. This is * not trivial, see doc/compiler.rst.
41609. the outer loop will recheck and exit
41610. allow caller to give a const number with the DUK_CONST_MARKER
41611. varname
41612. jump to end
41613. * ctx->prev_token token to process with duk__expr_nud() * ctx->curr_token updated by caller * * Note: the token in the switch below has already been eaten.
41614. basereg
41615. a
41616. * Else, must be one of: * - ExpressionStatement, possibly a directive (String) * - LabelledStatement (Identifier followed by ':') * * Expressions beginning with 'function' keyword are covered by a case * above (such expressions are not allowed in standard E5 anyway). * Also expressions starting with '{' are interpreted as block * statements. See E5 Section 12.4. * * Directive detection is tricky; see E5 Section 14.1 on directive * prologue. A directive is an expression statement with a single * string literal and an explicit or automatic semicolon. Escape * characters are significant and no parens etc are allowed: * * 'use strict'; // valid 'use strict' directive * 'use\0020strict'; // valid directive, not a 'use strict' directive * ('use strict'); // not a valid directive * * The expression is determined to consist of a single string literal * based on duk__expr_nud() and duk__expr_led() call counts. The string literal * of a 'use strict' directive is determined to lack any escapes based * num_escapes count from the lexer. Note that other directives may be * allowed to contain escapes, so a directive with escapes does not * terminate a directive prologue. * * We rely on the fact that the expression parser will not emit any * code for a single token expression. However, it will generate an * intermediate value which we will then successfully ignore. * * A similar approach is used for labels.
41617. * Default clause.
41618. Note: 'fast' breaks will jump to pc_label_site + 1, which will * then jump here. The double jump will be eliminated by a * peephole pass, resulting in an optimal jump here. The label * site jumps will remain in bytecode and will waste code size.
41619. Used by duk__emit*() calls so that we don't shuffle the loadints that * are needed to handle indirect opcodes.
41620. DUK_USE_REGEXP_SUPPORT
41621. CONDITIONAL EXPRESSION
41622. break/continue without label
41623. the DUK_TOK_RCURLY is eaten by duk__parse_stmts()
41624. jump for 'catch' case
41625. **comment:** XXX: append primitive
 label: code-design
41626. [... name reg_bind]
41627. e.g. DUK_OP_PREINCR
41628. eat 'while'
41629. try part
41630. XXX: here we need to know if 'left' is left-hand-side compatible. * That information is no longer available from current expr parsing * state; it would need to be carried into the 'left' ivalue or by * some other means.
41631. * Parsing an expression starting with 'new' is tricky because * there are multiple possible productions deriving from * LeftHandSideExpression which begin with 'new'. * * We currently resort to one-token lookahead to distinguish the * cases. Hopefully this is correct. The binding power must be * such that parsing ends at an LPAREN (CallExpression) but not at * a PERIOD or LBRACKET (MemberExpression). * * See doc/compiler.rst for discussion on the parsing approach, * and testcases/test-dev-new.js for a bunch of documented tests.
41632. exptrop is the top level variant which resets nud/led counts
41633. next loop requires a comma
41634. If there's no catch block, rc_varname will be 0 and duk__patch_trycatch() * will replace the LDCONST with a NOP. For any actual constant (including * constant 0) the DUK_CONST_MARKER flag will be set in rc_varname.
41635. * Init remaining result fields * * 'nregs' controls how large a register frame is allocated. * * 'nargs' controls how many formal arguments are written to registers: * r0, ... r(nargs-1). The remaining registers are initialized to * undefined.
41636. get const for value at valstack top

41637. eat 'break' or 'continue'
41638. DUK_TOK_TYPEOF
41639. * Parse inner function
41640. **comment:** unused
 label: code-design
41641. No BC shuffling now.
41642. number
41643. syntactically left-associative but parsed as right-associative
41644. reserve a jumps slot after instr before target spilling, used for NEXTENUM
41645. * Wrap up
41646. at least one opcode emitted
41647. Encode into a single opcode (18 bits can encode 1-2 bytes + length indicator)
41648. relookup after possible realloc
41649. The 'left' value must not be a register bound variable * because it may be mutated during the rest of the expression * and E5.1 Section 11.2.1 specifies the order of evaluation * so that the base value is evaluated first. * See: test-bug-nested-prop-mutate.js.
41650. regexp literal must not follow this token
41651. -> [...] func]
41652. automatic semi will be inserted
41653. Special shuffling for INITGET/INITSET, where slot C * identifies a register pair and cannot be shuffled * normally. Use an indirect variant instead.
41654. a label site has been emitted by duk_parse_stmt() automatically * (it will also emit the ENDLABEL).
41655. slot B is a target (default: source)
41656. without finally, the second jump slot is used to jump to end of stmt
41657. forget temp
41658. [...] filename &comp_stk]
41659. break/continue with label (label cannot be a reserved word, production is 'Identifier')
41660. Have a catch variable.
41661. * Intermediate value helpers
41662. * Shared assignment expression handling ** args = (opcode << 8) + rbp ** If 'opcode' is DUK_OP_NONE, plain assignment without arithmetic. * Syntactically valid left-hand-side forms which are not accepted as * left-hand-side values (e.g. as in "f() = 1") must NOT cause a * SyntaxError, but rather a run-time ReferenceError. ** When evaluating X <op>= Y, the LHS (X) is conceptually evaluated * to a temporary first. The RHS is then evaluated. Finally, the * <op> is applied to the initial value of RHS (not the value after * RHS evaluation), and written to X. Doing so concretely generates * inefficient code so we'd like to avoid the temporary when possible. * See: <https://github.com/svaarala/duktape/pull/992>. ** The expression value (final LHS value, written to RHS) is * conceptually copied into a fresh temporary so that it won't * change even if the LHS/RHS values change in outer expressions. * For example, it'd be generally incorrect for the expression value * to be the RHS register binding, unless there's a guarantee that it * won't change during further expression evaluation. Using the * temporary concretely produces inefficient bytecode, so we try to * avoid the extra temporary for some known-to-be-safe cases. * Currently the only safe case we detect is a "top level assignment", * for example "x = y + z;", where the assignment expression value is * ignored. * See: test-dev-assign-expr.js and test-bug-assign-mutate-gh381.js.
41663. TRYCATCH, cannot emit now (not enough info)
41664. reset function state (prepare for pass 2)
41665. **comment:** Lookup active label information. Break/continue distinction is necessary to handle switch * statement related labels correctly: a switch will only catch a 'break', not a 'continue'. ** An explicit label cannot appear multiple times in the active set, but empty labels (unlabelled * iteration and switch statements) can. A break will match the closest unlabelled or labelled * statement. A continue will match the closest unlabelled or labelled iteration statement. It is * a syntax error if a continue matches a labelled switch statement; because an explicit label cannot * be duplicated, the continue cannot match any valid label outside the switch. ** A side effect of these rules is that a LABEL statement related to a switch should never actually * catch a continue abrupt completion at run-time. Hence an INVALID opcode can be placed in the * continue slot of the switch's LABEL statement.
 label: code-design
41666. Value stack slot limits: these are quite approximate right now, and * because they overlap in control flow, some could be eliminated.
41667. **comment:** NOTE: "get" and "set" are not officially ReservedWords and the lexer * currently treats them always like ordinary identifiers (DUK_TOK_GET * and DUK_TOK_SET are unused). They need to be detected based on the * identifier string content.
 label: code-design
41668. **comment:** * Note: currently register bindings must be fixed for the entire * function. So, even though the catch variable is in a register * we know, we must use an explicit environment record and slow path * accesses to read/write the catch binding to make closures created * within the catch clause work correctly. This restriction should * be fixable (at least in common cases) later. ** See: test-bug-catch-binding-2.js. ** XXX: improve to get fast path access to most catch clauses.
 label: code-design
41669. DUK_TOK_EXPORT
41670. entry_top + 0
41671. * Variant 2 or 4
41672. [...] escaped_source bytecode]
41673. return 'res' (of right part) as our result
41674. jump from true part to end
41675. **comment:** XXX: spilling
 label: code-design
41676. eat identifier
41677. x1 must be a string
41678. break matches always
41679. dangerous: must only lower (temp_max not updated)
41680. * Inside one or more 'with' statements fall back to slow path always. * (See e.g. test-stmt-with.js.)
41681. format is bit packed
41682. DUK_TOK_LBRACKET already eaten, current token is right after that
41683. parse args starting from "next temp", reg_target + 1
41684. is_setget
41685. enable manually for dumping
41686. shadowed, ignore
41687. DUK_TOK_COLON
41688. DUK_TOK_IMPLEMENT
41689. jump to end or else part
41690. for side effects, result ignored
41691. for duk_error_augment.c
41692. AssignmentExpression
41693. DUK_TOK_MOD
41694. **comment:** XXX: this is pointless here because pass 1 is throw-away
 label: code-design
41695. * On first pass, perform actual parsing. Remember valstack top on entry * to restore it later, and switch to using a new function in comp_ctxt.
41696. non-Reference deletion is always 'true', even in strict mode
41697. no statement value (unlike function expression)
41698. jump to next loop, using reg_v34_iter as iterated value
41699. DUK_USE_TAILCALL

41700. For X <op>= Y we need to evaluate the pre-op * value of X before evaluating the RHS: the RHS * can change X, but when we do <op> we must use * the pre-op value.

41701. * Convert duk_compiler_func to a function template

41702. final result is already in 'res'

41703. -> loop body

41704. catch jump

41705. num_pairs and temp_start reset at top of outer loop

41706. **comment:** XXX: return indication of "terminalness" (e.g. a 'throw' is terminal)
label: code-design

41707. * Wrapping duk_safe_call() will mangle the stack, just return stack top

41708. pc_label

41709. slot A is a source (default: target)

41710. assume 'var' has been eaten

41711. old value is number: no refcount

41712. empty expressions can be detected conveniently with nud/led counts

41713. DUK_TOK_ALSHIFT_EQ

41714. label limits

41715. A 'return' statement is only allowed inside an actual function body, * not as part of eval or global code.

41716. DUK_TOK_GE

41717. DUK_TOK_LET

41718. Only a reg fits into 'A' so coerce 'res' into a register * for PUTVAR. ** XXX: here the current A/B/C split is suboptimal: we could * just use 9 bits for reg_res (and support constants) and 17 * instead of 18 bits for the varname const index.

41719. else an array initializer element

41720. tracks maximum initialized index + 1

41721. * Convert duk_compiler_func to a function template and add it * to the parent function table.

41722. **comment:** XXX: very similar to DUK_IVAL_ARITH - merge?
label: code-design

41723. **comment:** * Formal argument list * * We don't check for prohibited names or for duplicate argument * names here, because we don't yet know whether the function will * be strict. Function body parsing handles this retroactively.
label: code-design

41724. object literal key tracking flags

41725. reset 'allow_in' for parenthesized expression

41726. Coerce an duk_alue to a register or constant; result register may * be a temp or a bound register. ** The duk_alue argument ('x') is converted into a regconst as a * side effect.

41727. MEMBER/NEW/CALL EXPRESSIONS

41728. **comment:** XXX: add flag to indicate whether caller cares about return value; this * affects e.g. handling of assignment expressions. This change needs API * changes elsewhere too.
label: code-design

41729. Input shuffling happens before the actual operation, while output * shuffling happens afterwards. Output shuffling decisions are still * made at the same time to reduce branch clutter; output shuffle decisions * are recorded into X_out variables.

41730. parse args starting from "next temp"

41731. **comment:** XXX: support unary arithmetic ivales (useful?)
label: code-design

41732. arguments object would be accessible; note that shadowing * bindings are arguments or function declarations, neither * of which are deletable, so this is safe.

41733. lexing

41734. Main operation

41735. duk_reconst_t is unsigned, so use 0 as dummy value (ignored by caller)

41736. * For global or eval code this is straightforward. For functions * created with the Function constructor we only get the source for * the body and must manufacture the "function ..." part. ** For instance, for constructed functions (v8): * * > a = new Function("foo", "bar", "print(foo)"); * [Function] * > a.toString() * 'function anonymous(foo,bar) {\nprint(foo)\n}' * * Similarly for e.g. getters (v8): * * > x = { get a(foo,bar) { print(foo); } } * { a: [Getter] } * > Object.getOwnPropertyDescriptor(x, 'a').get.toString() * 'function a(foo,bar) { print(foo); }'

41737. initial index

41738. 'new' MemberExpression Arguments

41739. stmt has explicit/implicit semicolon terminator

41740. op_flags

41741. **comment:** * See the following documentation for discussion: * * doc/execution.rst: control flow details * * Try, catch, and finally "parts" are Blocks, not Statements, so * they must always be delimited by curly braces. This is unlike e.g. * the if statement, which accepts any Statement. This eliminates any * questions of matching parts of nested try statements. The Block * parsing is implemented inline here (instead of calling out). ** Finally part has a 'let scoped' variable, which requires a few kinks * here.
label: code-design

41742. XXX: integrate support for this into led() instead? * Similar issue as post-increment/post-decrement.

41743. DUK_TOK_FUNCTION

41744. array writes autoincrement length

41745. * Prototypes

41746. main expression parser function

41747. * 'arguments' binding is special; if a shadowing argument or * function declaration exists, an arguments object will * definitely not be needed, regardless of whether the identifier * 'arguments' is referenced inside the function body.

41748. statements go here (if any) on next loop

41749. **comment:** XXX: make this lenience dependent on flags or strictness?
label: code-design

41750. silence scan-build warning

41751. varname is popped by above code

41752. always allow 'in', coerce to 'tr' just in case

41753. A top-level assignment is e.g. "x = y". For these it's safe * to use the RHS as-is as the expression value, even if the RHS * is a reg-bound identifier. The RHS ('res') is right associative * so it has consumed all other assignment level operations; the * only relevant lower binding power construct is comma operator * which will ignore the expression value provided here. Usually * the top level assignment expression value is ignored, but it * is relevant for e.g. eval code.

41754. Tail call check: if last opcode emitted was CALL(I), and * the context allows it, change the CALL(I) to a tail call. * This doesn't guarantee that a tail call will be allowed at * runtime, so the RETURN must still be emitted. (Duktape * 0.10.0 avoided this and simulated a RETURN if a tail call * couldn't be used at runtime; but this didn't work * correctly with a thread yield/resume, see * test-bug-tailcall-thread-yield-resume.js for discussion.) * * In addition to the last opcode being CALL, we also need to * be sure that 'rc_val' is the result register of the CALL(I). * For instance, for the expression 'return 0, (function () * { return 1; })', 2' the last opcode emitted is CALL (no * bytecode is emitted for '2') but 'rc_val' indicates * constant '2'. Similarly if '2' is replaced by a register * bound variable, no opcodes are emitted but tail call would * be incorrect. ** This is tricky and easy to get wrong. It would be best to * track enough expression metadata to check that 'rc_val' came * from that last CALL instruction. We don't have that metadata * now, so we check that 'rc_val' is a temporary register result * (not a constant or a register bound variable). There should * be no way currently for 'rc_val' to be a temporary for an * expression following the CALL instruction without emitting * some opcodes following the CALL. This proxy check is used * below. ** See: test-bug-comma-expr-gh131.js. ** The non-standard 'caller' property disables tail calls * because they pose some special cases which haven't been * fixed yet.

41755. DUK_TOK_YIELD

41756. DUK_TOK_LNOT

41757. slot C is a target (default: source)
41758. Match labels starting from latest; once label_id no longer matches, we can * safely exit without checking the rest of the labels (only the topmost labels * are ever updated).
41759. reset bytecode buffer but keep current size; pass 2 will * require same amount or more.
41760. The 'else' ambiguity is resolved by 'else' binding to the innermost * construct, so greedy matching is correct here.
41761. **comment:** not allowed in strict mode, regardless of whether resolves; * in non-strict mode DELVAR handles both non-resolving and * resolving cases (the specification description is a bit confusing).
label: code-design
41762. end switch
41763. DUK_TOK_LE
41764. Use 'res' as the expression value (it's side effect * free and may be a plain value, a register, or a * constant) and write it to the LHS binding too.
41765. **comment:** XXX: 'res' setup can be moved to function body level; in fact, two 'res' * intermediate values suffice for parsing of each function. Nesting is needed * for nested functions (which may occur inside expressions).
label: code-design
41766. Parse a single variable declaration (e.g. "i" or "i=10"). A leading 'var' * has already been eaten. These is no return value in 'res', it is used only * as a temporary. * * When called from 'for-in' statement parser, the initializer expression must * not allow the 'in' token. The caller supply additional expression parsing * flags (like DUK_EXPR_FLAG_REJECT_IN) in 'expr_flags'. * * Finally, out_rc_varname and out_reg_varbind are updated to reflect where * the identifier is bound: * * If register bound: out_reg_varbind >= 0, out_rc_varname == 0 (ignore) * If not register bound: out_reg_varbind < 0, out_rc_varname >= 0 * * These allow the caller to use the variable for further assignment, e.g. * as is done in 'for-in' parsing.
41767. XXX: similar coercion issue as in DUK_TOK_PERIOD
41768. borrowed reference
41769. * Statement terminator check, including automatic semicolon * handling. After this step, 'curr_tok' should be the first * token after a possible statement terminator.
41770. bp to use when parsing a top level Expression
41771. Matches both +0 and -0 on purpose.
41772. bp is assumed to be even
41773. The code for writing reg_temps + 0 to the left hand side has already * been emitted.
41774. * Statement value handling. * * Global code and eval code has an implicit return value * which comes from the last statement with a value * (technically a non- "empty" continuation, which is * different from an empty statement). * * Since we don't know whether a later statement will * override the value of the current statement, we need * to coerce the statement value to a register allocated * for implicit return values. In other cases we need * to coerce the statement value to a plain value to get * any side effects out (consider e.g. "foo.bar;").
41775. transfer flags
41776. XXX: valstack handling is awkward. Add a valstack helper which * avoids dup():ing; valstack_copy(src, dst)?
41777. eat 'var'
41778. DUK_TOK_INTERFACE
41779. DUK_TOK_DIV
41780. * Any catch bindings ("catch (e)") also affect identifier binding. * * Currently, the varmap is modified for the duration of the catch * clause to ensure any identifier accesses with the catch variable * name will use slow path.
41781. iteration statements allow continue
41782. reg/const for switch value
41783. unary plus
41784. may be undefined
41785. reg/const for case value
41786. * Function formal arguments, always bound to registers * (there's no support for shuffling them now).
41787. **comment:** NEXTENUM needs a jump slot right after the main opcode. * We need the code emitter to reserve the slot: if there's * target shuffling, the target shuffle opcodes must happen * after the jump slot (for NEXTENUM the shuffle opcodes are * not needed if the enum is finished).
label: code-design
41788. DUK_TOK_PRIVATE
41789. Encode into a double constant (53 bits can encode 6*8 = 48 bits + 3-bit length
41790. then to a register
41791. expect EOF instead of }
41792. DUK_TOK_ARSHIFT_EQ
41793. **comment:** * For/for-in statement is complicated to parse because * determining the statement type (three-part for vs. a * for-in) requires potential backtracking. * * See the helper for the messy stuff.
label: code-design
41794. no code needs to be emitted, the regs already have values
41795. DUK_TOK_RCURLY
41796. fills window
41797. DUK_TOK_BAND_EQ
41798. jump is inserted here (variant 3)
41799. * Program code (global and eval code) has an implicit return value * from the last statement value (e.g. eval("1; 2+3;") returns 3). * This is not the case with functions. If implicit statement return * value is requested, all statements are coerced to a register * allocated here, and used in the implicit return statement below.
41800. **comment:** * Assignments are right associative, allows e.g. * a = 5; * a += b = 9; // same as a += (b = 9) * -> expression value 14, a = 14, b = 9 * * Right associativity is reflected in the BP for recursion, * "-1" ensures assignment operations are allowed. * * XXX: just use DUK_BP_COMMA (i.e. no need for 2-step bp levels)?
label: code-design
41801. don't allow continue
41802. valstack will be unbalanced, which is OK
41803. == DUK_MAX_TEMPS is OK
41804. DUK_TOK_EXTENDS
41805. [... varmap]
41806. inline arithmetic check for constant values
41807. identifier handling
41808. * Helpers for duk_compiler_func.
41809. DUK_TOK_RSHIFT_EQ
41810. step 4.b
41811. explicit semi follows
41812. shadowed; update value
41813. No support for lvalues returned from new or function call expressions. * However, these must NOT cause compile-time SyntaxErrors, but run-time * ReferenceErrors. Both left and right sides of the assignment must be * evaluated before throwing a ReferenceError. For instance: * * f() = g(); * * must result in f() being evaluated, then g() being evaluated, and * finally, a ReferenceError being thrown. See E5 Section 11.13.1.
41814. advance, whatever the current token is; parse next token in regexp context
41815. DUK_TOK_PROTECTED
41816. curr_token = get/set name
41817. target
41818. XXX: default priority for infix operators is duk_expr_lbp(tok) -> get it here?
41819. * Parse a function-like expression: * * - function expression * - function declaration * - function statement (non-standard) * - setter/getter * * Adds the function to comp_ctx->curr_func function table and returns the * function number. * * On entry, curr_token points to: * * - the token after 'function' for function expression/declaration/statement * - the token after 'set' or 'get' for setter/getter

41820. expr_flags
41821. Note: expect that caller has already eaten the left paren
41822. advance, expecting current token to be a specific token; parse next token in regexp context
41823. already in correct reg
41824. RELATIONAL EXPRESSION
41825. * Source filename (or equivalent), for identifying thrown errors.
41826. e.g. DUK_OP_POSTINCP
41827. preallocated temporaries (2) for variants 3 and 4
41828. emit instruction; could return PC but that's not needed in the majority * of cases.
41829. DUK_TOK_ALSHIFT
41830. **comment:** * Parse a function-body-like expression (FunctionBody or Program * in E5 grammar) using a two-pass parse. The productions appear * in the following contexts: * * - function expression * - function statement * - function declaration * - getter in object literal * - setter in object literal * - global code * - eval code *
- Function constructor body * * This function only parses the statement list of the body; the argument * list and possible function name must be initialized by the caller. * For instance, for Function constructor, the argument names are originally * on the value stack. The parsing of statements ends either at an EOF or * a closing brace; this is controlled by an input flag. * * Note that there are many differences affecting parsing and even code * generation: * * - Global and eval code have an implicit return value generated * by the last statement; function code does not * * - Global code, eval code, and Function constructor body end in * an EOF, other bodies in a closing brace ('}') * * Upon entry, 'curr_tok' is ignored and the function will pull in the * first token on its own. Upon exit, 'curr_tok' is the terminating * token (EOF or closing brace).
label: code-design
41831. trailing comma followed by rcurly
41832. eat 'with'
41833. comp_ctx->lex has been pre-initialized by caller: it has been * zeroed and input/input_length has been set.
41834. When torture not enabled, can just use the same helper because * 'reg' won't get spilled.
41835. step 4.c
41836. [...] key_obj key val]
41837. combine left->x1 and res->x1 (right->x1, really) -> (left->x1 OP res->x1)
41838. DUK_TOK_INCREMENT
41839. const flag for B
41840. keep func->h_argnames; it is fixed for all passes
41841. misc
41842. source is a function expression (used for Function constructor)
41843. **comment:** XXX: typing (duk_hcompiledfunction has duk_uint32_t)
label: code-design
41844. parsing in "directive prologue", recognize directives
41845. value stack indices for tracking objects
41846. DUK_ISPEC_XXX
41847. **comment:** code emission temporary
label: code-design
41848. maximum bytecode length in instructions
41849. bufwriter for code
41850. C array of duk_labelinfo
41851. Compiling state of one function, eventually converted to duk_hcompiledfunction
41852. with stack depth (affects identifier lookups)
41853. variable map for pass 2 (identifier -> register number or null (unmapped))
41854. curr_token slot1 (matches 'lex' slot1_idx)
41855. * Compiler intermediate values * * Intermediate values describe either plain values (e.g. strings or * numbers) or binary operations which have not yet been coerced into * either a left-hand-side or right-hand-side role (e.g. object property).
41856. type to represent a reg/const reference during compilation
41857. function is strict
41858. * Ecmascript compiler.
41859. array of declarations: [name1, val1, name2, val2, ...] * valN = (typeN) | (fnum << 8), where fnum is inner func number (0 for vars) * record function and variable declarations in pass 1
41860. bytecode opcode (or extraop) for binary ops
41861. current and previous token for parsing
41862. **comment:** function name (borrowed reference), ends up in _name
label: code-design
41863. lexing (tokenization) state (contains two valstack slot indices)
41864. property access
41865. array of active label names
41866. is eval code
41867. 1e8: protects against deeply nested inner functions
41868. source is eval code (not global)
41869. **comment:** function must not be tail called
label: code-design
41870. **comment:** code_idx: not needed
label: requirement
41871. no value
41872. inner function numbering
41873. always set; points to a reserved valstack slot
41874. expression is left-hand-side compatible
41875. These pointers are at the start of the struct so that they pack * nicely. Mixing pointers and integer values is bad on some * platforms (e.g. if int is 32 bits and pointers are 64 bits).
41876. highest value of temp_reg (temp_max - 1 is highest used reg)
41877. **comment:** XXX: can be optimized for smaller footprint esp. on 32-bit environments
label: code-design
41878. * Compiler state
41879. argument/function declaration shadows 'arguments'
41880. is a function declaration (as opposed to function expression)
41881. variable access
41882. function needs shuffle registers
41883. borrowed label name
41884. is global code
41885. strict outer context
41886. filename being compiled (ends up in functions' _filename' property)
41887. * PLAIN: x1 * ARITH: x1 <op> x2 * PROP: x1.x2 * VAR: x1 (name)
41888. function may call direct eval
41889. catch depth at point of definition
41890. binary arithmetic using extraops; DUK_EXTRAOP_INSTOF etc

41891. **comment:** XXX: optimize to 16 bytes
label: code-design

41892. binary arithmetic; DUK_OP_ADD, DUK_OP_EQ, other binary ops

41893. function refers to 'arguments' identifier

41894. **comment:** first register that is a temporary (below: variables)
label: code-design

41895. catch stack depth

41896. maximum loopcount for peephole optimization

41897. value resides in 'valstack_idx'

41898. value resides in register or constant

41899. Fast jumps (which avoid longjmp) jump directly to the jump sites * which are always known even while the iteration/switch statement * is still being parsed. A final peephole pass "straightens out" * the jumps.

41900. label id allocation (running counter)

41901. 1 GB

41902. current function being compiled (embedded instead of pointer for more compact access)

41903. stats for current expression being parsed

41904. * Prototypes

41905. DUK_IVAL_XXX

41906. **comment:** function makes one or more slow path accesses
label: code-design

41907. is an actual function (not global/eval code)

41908. array of function templates: [func1, offset1, line1, func2, offset2, line2] * offset/line points to closing brace to allow skipping on pass 2

41909. is a setter/getter

41910. ecmascript compiler limits

41911. curr_token slot2 (matches 'lex' slot2_idx)

41912. borrowed reference

41913. type to represent a straight register reference, with <0 indicating none

41914. **comment:** bit mask which indicates that a regconst is a constant instead of a register
label: code-design

41915. pc of label statement: * pc+1: break jump site * pc+2: continue jump site

41916. next temporary register to allocate

41917. parsing in "scanning" phase (first pass)

41918. shuffle registers if large number of regs/consts

41919. array

41920. array of formal argument names (-> _Formals)

41921. * Bytecode instruction representation during compilation ** Contains the actual instruction and (optionally) debug info.

41922. prev_token slot2

41923. DUK_JS_COMPILER_H_INCLUDED

41924. status booleans

41925. register for writing value of 'non-empty' statements (global or eval code), -1 is marker

41926. current paren level allows 'in' token

41927. numeric label_id (-1 reserved as marker)

41928. prev_token slot1

41929. reject RegExp literal on next advance() call; needed for handling IdentifierName productions

41930. statement id allocation (running counter)

41931. recursion limit

41932. parenthesis count, 0 = top level

41933. number of formal arguments

41934. register, constant, or value

41935. temp reg handling

41936. h_code: held in bw_code

41937. * Fast paths

41938. **comment:** 'd3' is never NaN, so no need to normalize
label: code-design

41939. * Executor interrupt handling ** The handler is called whenever the interrupt countdown reaches zero * (or below). The handler must perform whatever checks are activated, * e.g. check for cumulative step count to impose an execution step * limit or check for breakpoints or other debugger interaction. ** When the actions are done, the handler must reinit the interrupt * init and counter values. The 'init' value must indicate how many * bytecode instructions are executed before the next interrupt. The * counter must interface with the bytecode executor loop. Concretely, * the new init value is normally one higher than the new counter value. * For instance, to execute exactly one bytecode instruction the init * value is set to 1 and the counter to 0. If an error is thrown by the * interrupt handler, the counters are set to the same value (e.g. both * to 0 to cause an interrupt when the next bytecode instruction is about * to be executed after error handling). ** Maintaining the init/counter value properly is important for accurate * behavior. For instance, executor step limit needs a cumulative step * count which is simply computed as a sum of 'init' values. This must * work accurately even when single stepping.

41940. return value from Duktape.Thread.resume()

41941. Set catcher regs: idx_base+0 = value, idx_base+1 = lj_type.

41942. Result is always fastint compatible.

41943. Handle a BREAK/CONTINUE opcode. Avoid using longjmp() for BREAK/CONTINUE * handling because it has a measurable performance impact in ordinary * environments and an extreme impact in Emscripten (GH-342).

41944. -> [... val this]

41945. Preinc/predec for object properties.

41946. Assume that thr->valstack_bottom has been set-up before getting here.

41947. Sync and NULL early.

41948. * E5 Section 11.4.9

41949. **comment:** Hot variables for interpretation. Critical for performance, * but must add sparingly to minimize register shuffling.
label: code-design

41950. an Ecmascript function

41951. **comment:** XXX: improve check; check against nregs, not against top
label: code-design

41952. -> [... regexp_instance]

41953. Keep throwing an error whenever we get here. The unusual values * are set this way because no instruction is ever executed, we just * throw an error until all try/catch/finally and other catchpoints * have been exhausted. Duktape/C code gets control at each protected * call but whenever it enters back into Duktape the RangeError gets * raised. User exec timeout check must consistently indicate a timeout * until we've fully bubbled out of Duktape.

41954. x >= y --> not (x < y)

41955. A -> flags * B -> base register for call (base -> func, base+1 -> this, base+2 -> arg1 ... base+2+N-1 -> argN) * (for DUK_OP_CALLI, 'b' is indirect) * C -> nargs

41956. Unary plus is used to force a fastint check, so must include * downgrade check.

41957. Duktape.Thread.yield() should prevent

41958. **comment:** should never happen, but be robust
label: code-design

41959. [... enum] -> [...]

41960. A -> register of target object * B -> first register of key/value pair list * C -> number of key/value pairs
41961. **comment:** Note: 'act' is dangerous here because it may get invalidated at many * points, so we re-lookup it multiple times.
 label: code-design
41962. XXX: E5.1 Section 11.1.4 coerces the final length through * ToUint32() which is odd but happens now as a side effect of * 'arr_idx' type.
41963. side effects -> don't use tv_x, tv_y after
41964. flags
41965. Relookup and initialize dispatch loop variables. Debugger check.
41966. never here
41967. add_auto_proto
41968. don't re-enter e.g. during Eval
41969. -> [... obj key value]
41970. duk_new() will call the constructor using duk_handle_call(). * A constructor call prevents a yield from inside the constructor, * even if the constructor is an ECMAScript function.
41971. may be < thr->catchstack initially
41972. Longjmp handling has restored jmpbuf_ptr.
41973. valstack top, should match before interpreting each op (no leftovers)
41974. Entry level info.
41975. Input values are signed 48-bit so we can detect overflow * reliably from high bits or just a comparison.
41976. XXX: assert 'c' is an enumerator
41977. -> [val]
41978. **comment:** XXX: could unwind catchstack here, so that call handling * didn't need to do that?
 label: code-design
41979. **comment:** XXX: refactor out?
 label: code-design
41980. [...] env]
41981. B -> register for writing enumerator object * C -> value to be enumerated (register)
41982. cleared before entering catch part
41983. -> [... obj]
41984. no need to unwind catchstack
41985. **comment:** * Arithmetic operations other than '+' have number-only semantics * and are implemented here. The separate switch-case here means a * "double dispatch" of the arithmetic opcode, but saves code space. * * E5 Sections 11.5, 11.5.1, 11.5.2, 11.5.3, 11.6, 11.6.1, 11.6.2, 11.6.3.
 label: code-design
41986. When debugger is enabled, we need to recheck the activation * status after returning. This is now handled by call handling * and heap->dbg_force_restart.
41987. thr->heap->lj.value1 is already the value to throw
41988. if boolean matches A, skip next inst
41989. Preinc/predic for var-by-name, slow path.
41990. A -> target register * B -> bytecode (also contains flags) * C -> escaped source
41991. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
41992. bytecode has a stable pointer
41993. cleared before entering finally
41994. **comment:** Executor interrupt counter check, used to implement breakpoints, * debugging interface, execution timeouts, etc. The counter is heap * specific but is maintained in the current thread to make the check * as fast as possible. The counter is copied back to the heap struct * whenever a thread switch occurs by the DUK_HEAP_SWITCH_THREAD() macro.
 label: code-design
41995. Signed shift, named "arithmetic" (asl) because the result * is signed, e.g. 4294967295 << 1 -> -2. Note that result * must be masked.
41996. Overflow of the execution counter is fine and doesn't break * anything here.
41997. Restart bytecode execution, possibly with a changed thread.
41998. Don't allow a zero divisor. Fast path check by * "verifying" with multiplication. Also avoid zero * dividend to avoid negative zero issues (0 / -1 = -0 * for instance).
41999. no need for refcount update
42000. reg count
42001. **comment:** ToBoolean() does not require any operations with side effects so * we can do it efficiently. For footprint it would be better to use * duk_js_tobool()

and then push+replace to the result slot.
 label: code-design

42002. * Slow path: potentially requires function calls for coercion
42003. -> [Object defineProperty undefined obj key desc]
42004. DUK_USE_FASTINT
42005. ToNumber() for a fastint is a no-op.
42006. B -> object register * C -> C+0 contains key, C+1 closure (value)
42007. x < y
42008. call_flags
42009. **comment:** XXX: unnecessary copying of values? Just set 'top' to * b + c, and let the return handling fix up the stack frame?
 label: code-design
42010. Force interrupt right away if we're paused or in "checked mode". * Step out is handled by callstack unwind.
42011. * Local declarations.
42012. * INITSET/INITGET are only used to initialize object literal keys. * The compiler ensures that there cannot be a previous data property * of the same name. It also ensures that setter and getter can only * be initialized once (or not at all).
42013. If we processed any debug messages breakpoints may have * changed; restart execution to re-check active breakpoints.
42014. Handle a RETURN opcode. Avoid using longjmp() for return handling because * it has a measurable performance impact in ordinary environments and an extreme * impact in Emscripten (GH-342). Return value is on value stack top.
42015. Ecma-to-ecma call possible, may or may not be a tail call. * Avoid C recursion by being clever.
42016. Note: MUST NOT wipe_and_return here, as heap->lj must remain intact
42017. x > y --> y < x
42018. **comment:** XXX: this is now a very unoptimal implementation -- this can be * made very simple by direct manipulation of the object internals, * given the guarantees above.
 label: code-design
42019. curr_pc synced by duk_handle_ecma_call_setup()
42020. resume: [... initial_func] (currently actually: [initial_func])
42021. **comment:** XXX: use duk_is_valid_index() instead?
 label: code-design
42022. -> [... obj value]
42023. always normalized
42024. DUK_USE_INTERRUPT_COUNTER
42025. **comment:** writable, not configurable
 label: code-design
42026. written by a previous RESUME handling
42027. **comment:** LDINTX is not necessarily in FASTINT range, so * no fast path for now. * * XXX: perhaps restrict LDINTX to fastint range, wider * range very rarely needed.

label: code-design

42028. duk_js_call.c is required to restore the stack reserve * so we only need to reset the top.

42029. **comment:** this is not strictly necessary, but helps debugging**label:** code-design

42030. this should never be possible, because the switch-case is * comprehensive

42031. Pre/post inc/dec for register variables, important for loops.

42032. * Avoid nested calls. Concretely this happens during debugging, e.g. * when we eval() an expression. * * Also don't interrupt if we're currently doing debug processing * (which can be initiated outside the bytecode executor) as this * may cause the debugger to be called recursively. Check required * for correct operation of throw intercept and other "exotic" halting * scenarios.

42033. w/o refcounts

42034. fast path

42035. **comment:** XXX: this could be a DUK__CONSTP instead**label:** code-design

42036. stable; precalculated for faster lookups

42037. **comment:** 'val' is never NaN, so no need to normalize**label:** code-design

42038. may be reg or const

42039. +2 = catcher value, catcher lj_type

42040. **comment:** XXX: because we're dealing with 'own' properties of a fresh array, * the array initializer should just ensure that the array has a large * enough array part and write the values directly into array part, * and finally set 'length' manually in the end (as already happens now).**label:** code-design

42041. * Longjmp and other control flow transfer for the bytecode executor. * * The longjmp handler can handle all longjmp types: error, yield, and * resume (pseudotypes are never actually thrown). * * Error policy for longjmp: should not ordinarily throw errors; if errors * occur (e.g. due to out-of-memory) they bubble outwards rather than being * handled recursively.

42042. **comment:** Reg/const access macros: these are very footprint and performance sensitive * so modify with care.**label:** code-design

42043. [... obj] -> [...]

42044. Already declared, update value.

42045. If out of catchstack, cat = thr->catchstack - 1; * new_cat_top will be 0 in that case.

42046. duk_xdef_prop() will define an own property without any array * special behaviors. We'll need to set the array length explicitly * in the end. For arrays with elisions, the compiler will emit an * explicit SETALEN which will update the length.

42047. **comment:** Switch to caller's setjmp() catcher so that if an error occurs * during error handling, it is always propagated outwards instead * of causing an infinite loop in our own handler.**label:** code-design42048. **comment:** XXX: remove heap->dbg_exec_counter, use heap->inst_count_interrupt instead?**label:** code-design

42049. functions always have a NEWENV flag, i.e. they get a * new variable declaration environment, so only lex_env * matters here.

42050. should not happen

42051. A -> flags * BC -> reg_catch; base register for two registers used both during * trycatch setup and when catch is triggered * * If DUK_BC_TRYCATCH_FLAG_CATCH_BINDING set: * reg_catch + 0: catch binding variable name (string). * Automatic declarative environment is established for * the duration of the 'catch' clause. * * If DUK_BC_TRYCATCH_FLAG_WITH_BINDING set: * reg_catch + 0: with 'target value', which is coerced to * an object and then used as a bindind object for an * environment record. The binding is initialized here, for * the 'try' clause. * * Note that a TRYCATCH generated for a 'with' statement has no * catch or finally parts.

42052. [... constructor arg1 ... argN] -> [retval]

42053. Check for breakpoints only on line transition. * Breakpoint is triggered when we enter the target * line from a different line, and the previous line * was within the same function. * * This condition is tricky: the condition used to be * that transition to -or across- the breakpoint line * triggered the breakpoint. This seems intuitively * better because it handles breakpoints on lines with * no emitted opcodes; but this leads to the issue * described in: <https://github.com/svaaraala/duktape/issues/263>.

42054. B -> target register * C -> constant index of identifier name

42055. * Execution timeout check

42056. side effects

42057. E5 Section 11.7.2, steps 7 and 8

42058. updates thread state, minimizes its allocations

42059. **comment:** It might seem that replacing 'thr->heap' with just 'heap' below * might be a good idea, but it increases code size slightly * (probably due to unnecessary spilling) at least on x64.**label:** code-design

42060. use_prev_pc

42061. Must restart in all cases because we NULLed thr->ptr_curr_pc.

42062. checked by Duktape.Thread.resume()

42063. unsigned

42064. duk.bi_duk_object_yield() and duk.bi_duk_object_resume() ensure all of these are met

42065. E5 Section 11.2.3, step 6.a.i

42066. A -> target register (A, A+1) for call setup * (for DUK_OP_CSREGI, 'a' is indirect) * B -> register containing target function (not type checked here)

42067. Must ensure result is 64-bit (no overflow); a * simple and sufficient fast path is to allow only * 32-bit inputs. Avoid zero inputs to avoid * negative zero issues (-1 * 0 = -0, for instance).

42068. lj.value1 is already set

42069. Dispatch loop.

42070. fall through if overflow etc

42071. this may have side effects, so re-lookup act

42072. Assume interrupt init/counter are properly initialized here.

42073. Continue checked execution if there are breakpoints or we're stepping. * Also use checked execution if paused flag is active - it shouldn't be * because the debug message loop shouldn't terminate if it was. Step out * is handled by callstack unwind and doesn't need checked execution. * Note that debugger may have detached due to error or explicit request * above, so we must recheck attach status.

42074. Set a high interrupt counter; the original executor * interrupt invocation will rewrite before exiting.

42075. The compiler should never emit DUK_OP_REGEXEXP if there is no * regexp support.

42076. **comment:** XXX: TRYCATCH handling should be reworked to avoid creating * an explicit scope unless it is actually needed (e.g. function * instances or eval is executed inside the catch block). This * rework is not trivial because the compiler doesn't have an * intermediate representation. When the rework is done, the * opcode format can also be made more straightforward.**label:** code-design

42077. As a first approximation, buffer values are coerced to strings * for addition. This means that adding two buffers currently * results in a string.

42078. must be ecmascript

42079. **comment:** XXX: does not work if thr->catchstack is NULL**label:** code-design

42080. The smallest fastint is no longer 48-bit when * negated. Positive zero becomes negative zero * (cannot be represented) when negated.

42081. allocate catcher and populate it (should be atomic)

42082. E5 Section 11.7.1, steps 7 and 8

42083. No need to check for size of bp_active list, * it's always larger than maximum number of * breakpoints.

42084. YIELDED: Ecmascript activation + Duktape.Thread.yield() activation

42085. * Handling DUK_OP_ADD this way is more compact (experimentally) * than a separate case with separate argument decoding.

42086. INACTIVE: no activation, single function value on valstack

42087. **comment:** * Restart execution by reloading thread state. * * Note that 'thr' and any thread configuration may have changed, * so all local variables are suspect and we need to reinitialize. * * The number of local variables should be kept to a minimum: if * the variables are spilled, they will need to be loaded from * memory anyway. * * Any 'goto restart_execution;' code path in opcode dispatch must * ensure 'curr_pc' is synced back to act->curr_pc before the goto * takes place. * * The interpreter must be very careful with memory pointers, as * many pointers are not guaranteed to be 'stable' and may be * reallocated and relocated on-the-fly quite easily (e.g. by a * memory allocation or a property access). * * The following are assumed to have stable pointers: * - the current thread * - the current function * - the bytecode, constant table, inner function table of the * current function (as they are a part of the function allocation) * * The following are assumed to have semi-stable pointers: * - the current activation entry: stable as long as callstack * is not changed (reallocated by growing or shrinking), or * by any garbage collection invocation (through finalizers) * - Note in particular that ANY DECREF can invalidate the * activation pointer, so for the most part a fresh lookup * is required * * The following are not assumed to have stable pointers at all: * - the value stack (registers) of the current thread * - the catch stack of the current thread * * See execution.rst for discussion.

label: code-design

42088. * Ecmascript bytecode executor. * * Resume execution for the current thread from its current activation. * Returns when execution would return from the entry level activation, * leaving a single return value on top of the stack. Function calls * and thread resumptions are handled internally. If an error occurs, * a longjmp() with type DUK_LJ_TYPE_THROW is called on the entry level * setjmp() jmpbuf. * * Ecmascript function calls and coroutine resumptions are handled * internally (by the outer executor function) without recursive C calls. * Other function calls are handled using duk_handle_call(), increasing * C recursion depth. * * Abrupt completions (= long control transfers) are handled either * directly by reconfiguring relevant stacks and restarting execution, * or via a longjmp. Longjmp-free handling is preferable for performance * (especially Emscripten performance), and is used for: break, continue, * and return. * * For more detailed notes, see doc/execution.rst. * * Also see doc/code-issues.rst for discussion of setjmp(), longjmp(), * and volatile.

42089. Compiler is responsible for selecting property flags (configurability, * writability, etc).

42090. enum_flags

42091. E5 Sections 11.9.1, 11.9.3

42092. * Note: we currently assume that the setjmp() catchpoint is * not re-entrant (longjmp() cannot be called more than once * for a single setjmp()). * * See doc/code-issues.rst for notes on variable assignment * before and after setjmp().

42093. **comment:** * (Re)try handling the longjmp. * * A longjmp handler may convert the longjmp to a different type and * "virtually" rethrow by goto'ing to 'check_longjmp'. Before the goto, * the following must be updated: * - the heap 'lj' state * - 'thr' must reflect the "throwing" thread

label: code-design

42094. Force pause if we were doing "step into" in another activation.

42095. Note: target registers a and a+1 may overlap with DUK_REGP(b) * and DUK_REGCONSTP(c). Careful here.

42096. already set

42097. [... val] --> [... enum]

42098. these are not necessarily 0 or 1 (may be other non-zero), that's ok

42099. +1 = finally

42100. Negative value checked so that a "time jump" works * reasonably. * * Same interval is now used for status sending and * peeking.

42101. Note: 'this' may be bound to any value, not just an object

42102. * Currently only allowed only if yielding thread has only * Ecmascript activations (except for the Duktape.Thread.yield() * call at the callstack top) and none of them constructor * calls. * * This excludes the 'entry' thread which will always have * a preventcount > 0.

42103. + spare

42104. * Process messages and send status if necessary. * * If we're paused, we'll block for new messages. If we're not * paused, we'll process anything we can peek but won't block * for more. Detach (and re-attach) handling is all localized * to duk_debug_process_messages() too. * * Debugger writes outside the message loop may cause debugger * detach1 phase to run, after which dbg_read_cb == NULL and * dbg_detaching != 0. The message loop will finish the detach * by running detach2 phase, so enter the message loop also when * detaching.

42105. must "goto restart_execution", e.g. breakpoints changed

42106. Share yield longjmp handler.

42107. **comment:** * Three possible outcomes: * * A try or finally catcher is found => resume there. * (or) * * The error propagates to the bytecode executor entry * level (and we're in the entry thread) => rethrow * with a new longjmp(), after restoring the previous * catchpoint. * * The error is not caught in the current thread, so * the thread finishes with an error. This works like * a yielded error, except that the thread is finished * and can no longer be resumed. (There is always a * resumer in this case.) * * Note: until we hit the entry level, there can only be * Ecmascript activations.

label: code-design

42108. no need to unwind callstack

42109. unwind to 'yield' caller

42110. 'this binding' is just under bottom

42111. stable

42112. Enter message processing loop for sending Status notifs and * to finish a pending detach.

42113. Duktape.Thread.resume()

42114. +0 = break, +1 = continue

42115. Outer executor with setjmp/longjmp handling.

42116. if debugging disabled

42117. E5 Section 10.4.3

42118. does not modify tv_x

42119. [... func this arg1 ... argN]

42120. Cause an interrupt after executing one instruction.

42121. side effects, perform in-place

42122. 'act' is no longer accessed, scanbuild fix

42123. Ecmascript activation + Duktape.Thread.yield() activation

42124. **comment:** Presence of 'fun' is config based, there's a marginal performance * difference and the best option is architecture dependent.

label: code-design

42125. reg

42126. no compact

42127. restricted to regs

42128. exit bytecode executor by rethrowing an error to caller

42129. **comment:** XXX: side effect handling is quite awkward here

label: code-design

42130. Note: thr->catchstack_top may be 0, so that cat < thr->catchstack * initially. This is OK and intended.

42131. Call setup checks callability.

42132. exit bytecode execution with return value

42133. A -> target reg * B -> object reg/const (may be const e.g. in "'foo'[1]') * C -> key reg/const

42134. There is no eval() special handling here: eval() is never * automatically converted to a lightfunc.

42135. Write curr_pc back for the debugger.

42136. Called for handling both a longjmp() with type DUK_LJ_TYPE_YIELD and * when a RETURN opcode terminates a thread and yields to the resumer.

42137. * Other cases, use C recursion. * * If a tail call was requested we ignore it and execute a normal call. * Since Duktape 0.11.0 the compiler emits a RETURN opcode even after * a tail call to avoid test-bug-tailcall-thread-yield-resume.js. * * Direct eval call: (1) call target (before following bound function * chain) is the built-in eval() function, and (2) call was made with * the identifier 'eval'.

42138. **comment:** Current PC, accessed by other functions through thr->ptr_to_curr_pc. * Critical for performance. It would be safest to make this volatile, * but that eliminates performance benefits; aliasing guarantees * should be enough though.

label: code-design

42139. no refcount changes

42140. 'null' enumerator case -> behave as with an empty enumerator
42141. **comment:** Compared to duk_handle_call(): * - protected call: never * - ignore recursion limit: never
 label: code-design
42142. A -> unused (reserved for flags, for consistency with DUK_OP_CALL) * B -> target register and start reg: constructor, arg1, ..., argN * (for DUK_OP_NEWI, 'b' is indirect) * C -> num args (N)
42143. **comment:** XXX: putvar takes a duk_tval pointer, which is awkward and * should be reworked.
 label: code-design
42144. bottom of current func
42145. duk__handle_return() is guaranteed never to throw, except * for potential out-of-memory situations which will then * propagate out of the executor longjmp handler.
42146. **comment:** XXX: could use at least one fewer loop counters
 label: code-design
42147. A -> flags * B -> return value reg/const * C -> currently unused
42148. Note: left shift, should mask
42149. Ecmascript function
42150. curr_pc synced back above
42151. relookup (side effects)
42152. GH-303
42153. return value from Duktape.Thread.yield()
42154. * Ecmascript modulus ('%') does not match IEEE 754 "remainder" * operation (implemented by remainder() in C99) but does seem * to match ANSI C fmod(). * * Compare E5 Section 11.5.3 and "man fmod".
42155. The counter value is one less than the init value: init value should * indicate how many instructions are executed before interrupt. To * execute 1 instruction (after interrupt handler return), counter must * be 0.
42156. unsigned shift
42157. Note: errors are augmented when they are created, not * when they are thrown. So, don't augment here, it would * break re-throwing for instance.
42158. We request a tail call, but in some corner cases * call handling can decide that a tail call is * actually not possible. * See: test-bug-tailcall-preventyield-assert.c.
42159. **comment:** Not necessary to unwind catchstack: return handling will * do it. The finally flag of 'cat' is no longer set. The * catch flag may be set, but it's not checked by return handling.
 label: code-design
42160. **comment:** Inner executor, performance critical.
 label: code-design
42161. **comment:** This could also be thrown internally (set the error, goto check_longjmp), * but it's better for internal errors to bubble outwards so that we won't * infinite loop in this catchpoint.
 label: code-design
42162. * Assuming a register binds to a variable declared within this * function (a declarative binding), the 'this' for the call * setup is always 'undefined'. E5 Section 10.2.1.1.6.
42163. CATCH flag may be enabled or disabled here; it may be enabled if * the statement has a catch block but the try block does not throw * an error.
42164. * If entering a 'catch' block which requires an automatic * catch variable binding, create the lexical environment. * * The binding is mutable (= writable) but not deletable. * Step 4 for the catch production in E5 Section 12.14; * no value is given for CreateMutableBinding 'D' argument, * which implies the binding is not deletable.
42165. XXX: Set 32-bit result (but must then handle signed and * unsigned results separately).
42166. x
42167. lj.value1 already set
42168. A -> object reg * B -> key reg/const * C -> value reg/const * * Note: intentional difference to register arrangement * of e.g. GETPROP; 'A' must contain a register-only value.
42169. at least one activation, ours
42170. XXX: assert idx_base
42171. not caught by anything before entry level; rethrow and let the * final catcher unwind everything
42172. Quite heavy assert: check valstack policy. Improper * shuffle instructions can write beyond valstack_top/end * so this check catches them in the act.
42173. Note: currently the catch binding is handled without a register * binding because we don't support dynamic register bindings (they * must be fixed for an entire function). So, there is no need to * record regbases etc.
42174. * Breakpoint and step state checks
42175. [... env target target]
42176. Don't allow a zero divisor. Restrict both v1 and * v2 to positive values to avoid compiler specific * behavior.
42177. overflow, fall through
42178. restored if ecma-to-ecma setup fails
42179. type
42180. reset longjmp
42181. **comment:** pseudotypes, not used in actual longjumps
 label: code-design
42182. borrowed reference; although 'tv1' comes from a register, * its value was loaded using LDCONST so the constant will * also exist and be reachable.
42183. A -> target reg * BC -> inner function index
42184. resumee
42185. **comment:** This macro works when a regconst field is 9 bits, [0,0x1ff]. Adding * DUK_LIKELY/DUK_UNLIKELY increases code footprint and doesn't seem to * improve performance on x64 (and actually harms performance in some tests).
 label: code-design
42186. Fast path for the case where the register * is a number (e.g. loop counter).
42187. * Ecmascript bytecode executor.
42188. **comment:** * Four possible outcomes: * * 1. A 'finally' in the same function catches the 'return'. * It may continue to propagate when 'finally' is finished, * or it may be neutralized by 'finally' (both handled by * ENDFIN). * * 2. The return happens at the entry level of the bytecode * executor, so return from the executor (in C stack). * * 3. There is a calling (Ecmascript) activation in the call * stack => return to it, in the same executor instance. * * 4. There is no calling activation, and the thread is * terminated. There is always a resumer in this case, * which gets the return value similarly to a 'yield' * (except that the current thread can no longer be * resumed).
 label: code-design
42189. **comment:** XXX: signalling the need to shrink check (only if unwound)
 label: code-design
42190. **comment:** 'this' binding is not needed here
 label: requirement
42191. DUK_USE_EXEC_TIMEOUT_CHECK
42192. Two lowest bits of opcode are used to distinguish * variants. Bit 0 = inc(0)/dec(1), bit 1 = pre(0)/post(1).
42193. entry level reached
42194. get_value
42195. Note: target registers a and a+1 may overlap with DUK__REGP(b). * Careful here.
42196. **comment:** * Enumeration semantics come from for-in statement, E5 Section 12.6.4. * If called with 'null' or 'undefined', this opcode returns 'null' as * the enumerator, which is special cased in NEXTENUM. This simplifies * the compiler part
 label: code-design
42197. signed shift

42198. **comment:** * To determine whether to use an optimized Ecmascript-to-Ecmascript * call, we need to know whether the final, non-bound function is an * Ecmascript function. ** This is now implemented so that we start to do an ecma-to-ecma call * setup which will resolve the bound chain as the first thing. If the * final function is not eligible, the return value indicates that the * ecma-to-ecma call is not possible. The setup will overwrite the call * target at DUK_REGP(idx) with the final, non-bound function (which * may be a lightfunc), and fudge arguments if necessary. ** XXX: If an ecma-to-ecma call is not possible, this initial call * setup will do bound function chain resolution but won't do the * "effective this binding" resolution which is quite confusing. * Perhaps add a helper for doing bound function and effective this * binding resolution - and call that explicitly? Ecma-to-ecma call * setup and normal function handling can then assume this prestep has * been done by the caller.
label: code-design

42199. ensures callstack_top - 1 >= 0
42200. Don't need to sync curr_pc here; duk_new() will do that * when it augments the created error.
42201. No need to reinit setjmp() catchpoint, as call handling * will store and restore our state.
42202. variable value (function, we hope, not checked here)
42203. x <= y --> not (x > y) --> not (y < x)
42204. mandatory if du.d is a NaN
42205. Typing: use duk_small_(u)int_fast_t when decoding small * opcode fields (op, A, B, C) and duk_(u)int_fast_t when * decoding larger fields (e.g. BC which is 18 bits). Use * unsigned variant by default, signed when the value is used * in signed arithmetic. Using variable names such as 'a', 'b', * 'c', 'bc', etc makes it easier to spot typing mismatches.
42206. **comment:** not needed
label: requirement
42207. **comment:** 'fun' is quite rarely used, so no local for it
label: code-design
42208. Return to the bytecode executor caller which will unwind stacks. * Return value is already on the stack top: [... retval].
42209. idx_retval unsigned
42210. DUK_USE_DEBUGGER_SUPPORT
42211. 'thr' is the current thread, as no-one resumes except us and we * switch 'thr' in that case.
42212. There is a caller; it MUST be an Ecmascript caller (otherwise it would * match entry level check)
42213. 'with' target must be created first, in case we run out of memory
42214. Avoid recursive re-entry; enter when we're attached or * detaching (to finish off the pending detach).
42215. 'this' value: * E5 Section 6.b.i ** The only (standard) case where the 'this' binding is non-null is when * (1) the variable is found in an object environment record, and * (2) that object environment record is a 'with' block. *
42216. not needed, as we exit right away
42217. **comment:** * Binary bitwise operations use different coercions (ToInt32, ToUint32) * depending on the operation. We coerce the arguments first using * ToInt32(), and then cast to an 32-bit value if necessary. Note that * such casts must be correct even if there is no native 32-bit type * (e.g., duk_int32_t and duk_uint32_t are 64-bit). ** E5 Sections 11.10, 11.7.1, 11.7.2, 11.7.3
label: code-design
42218. Reconfigure value stack for return to an Ecmascript function at 'act_idx'.
42219. relookup, may have changed
42220. **comment:** unused for label
label: code-design
42221. **comment:** * Arithmetic, binary, and logical helpers. ** Note: there is no opcode for logical AND or logical OR; this is on * purpose, because the evalution order semantics for them make such * opcodes pretty pointless: short circuiting means they are most * comfortably implemented as jumps. However, a logical NOT opcode * is useful. ** Note: careful with duk_tval pointers here: they are potentially * invalidated by any DECREF and almost any API call. It's still * preferable to work without making a copy but that's not always * possible.
label: code-design
42222. **comment:** 'funcs' is quite rarely used, so no local for it
label: code-design
42223. relookup if changed
42224. Shouldn't happen but check anyway.
42225. **comment:** XXX: the best typing needs to be validated by perf measurement: * e.g. using a small type which is the cast to a larger duk_idx_t * may be slower than declaring the variable as a duk_idx_t in the * first place.
label: code-design
42226. -> [... escaped_source bytecode]
42227. result
42228. not protected, respect reclimit, not constructor
42229. [... env target]
42230. * Debugger handling for executor restart ** Check for breakpoints, stepping, etc, and figure out if we should execute * in checked or normal mode. Note that we can't do this when an activation * is created, because breakpoint status (and stepping status) may change * later, so we must recheck every time we're executing an activation. * This primitive should be side effect free to avoid changes during check.
42231. ignore
42232. special result value handling
42233. skip jump slots
42234. Opcode only emitted by compiler when debugger * support is enabled. Ignore it silently without * debugger support, in case it has been loaded * from precompiled bytecode.
42235. We can directly access value stack here.
42236. Set up curr_pc for opcode dispatch.
42237. val
42238. **comment:** XXX: use macros for the repetitive tval/refcount handling.
label: code-design
42239. **comment:** unused
label: code-design
42240. Note: target registers a and a+1 may overlap with DUK_REGCONSTP(b) * and DUK_REGCONSTP(c). Careful here.
42241. Ecmascript activation + Duktape.Thread.resume() activation
42242. always in executor
42243. reachable through activation
42244. thr->heap->lj.value2 is 'thread', will be wiped out at the end
42245. **comment:** XXX: faster initialization (direct access or better primitives)
label: code-design
42246. **comment:** XXX: shared decoding of 'b' and 'c'?
label: code-design
42247. important to use normalized NaN with 8-byte tagged types
42248. +0 = catch
42249. get before side effects
42250. **comment:** Note: combining comparison ops must be done carefully because * of uncomparable values (NaN): it's not necessarily true that * (x >= y) === !(x < y). Also, evaluation order matters, and * although it would only seem to affect the compiler this is * actually not the case, because there are also run-time coercions * of the arguments (with potential side effects). ** XXX: can be combined; check code size.
label: code-design
42251. * E5 Section 11.4.8
42252. always provideThis=true

42253. duk_small_uint_fast_t c = DUK_DEC_C(ins);
42254. **comment:** XXX: fastint fast path would be very useful here
 label: code-design
42255. no specific action, resume normal execution
42256. Execute bytecode until returned or longjmp().
42257. Relookup if relocated
42258. Not necessary to unwind catchstack: break/continue * handling will do it. The finally flag of 'cat' is * no longer set. The catch flag may be set, but it's * not checked by break/continue handling.
42259. * Rate limit check for sending status update or peeking into * the debug transport. Both can be expensive operations that * we don't want to do on every opcode. *
 * Making sure the interval remains reasonable on a wide variety * of targets and bytecode is difficult without a timestamp, so * we use a Date-provided timestamp for the rate limit check. * But since it's also expensive to get a timestamp, a bytecode * counter is used to rate limit getting timestamps.
42260. num_stack_args
42261. * Find a matching label catcher or 'finally' catcher in * the same function. * * A label catcher must always exist and will match unless * a 'finally' captures the break/continue first. It is the * compiler's responsibility to ensure that labels are used * correctly.
42262. is resume, not a tail call
42263. avoid referencing, invalidated
42264. +1 = one retval
42265. B -> target register for next key * C -> enum register
42266. Force restart caused by a function return; must recheck * debugger breakpoints before checking line transitions, * see GH-303. Restart and then handle interrupt_counter * zero again.
42267. **comment:** Registers 'bc' and 'bc + 1' are written in longjmp handling * and if their previous values (which are temporaries) become * unreachable -and- have a finalizer, there'll be a function * call during error handling which is not supported now (GH-287). * Ensure that both 'bc' and 'bc + 1' have primitive values to * guarantee no finalizer calls in error handling. Scrubbing also * ensures finalizers for the previous values run here rather than * later. Error handling related values are also written to 'bc' * and 'bc + 1' but those values never become unreachable during * error handling, so there's no side effect problem even if the * error value has a finalizer.
 label: code-design
42268. no_block
42269. cat_idx catcher is kept, even for finally
42270. XXX: switch cast?
42271. keep label catcher
42272. state updated, restart bytecode execution
42273. Must be restored here to handle e.g. yields properly.
42274. **comment:** * Addition operator is different from other arithmetic * operations in that it also provides string concatenation. * Hence it is implemented separately. * *
 There is a fast path for number addition. Other cases go * through potentially multiple coercions as described in the * E5 specification. It may be possible to reduce the number * of coercions, but this must be done carefully to preserve * the exact semantics. * * E5 Section 11.6.1. * * Custom types also have special behavior implemented here.
 label: code-design
42275. Longjmp callers are required to sync-and-null thr->ptr_curr_pc * before longjmp.
42276. ToNumber() for a double is a no-op.
42277. [... enum] -> [... next_key]
42278. leave 'cat' as top catcher (also works if catchstack exhausted)
42279. XXX: direct manipulation, or duk_replace_tval()
42280. We'll need to interrupt early so recompute the init * counter to reflect the number of bytecode instructions * executed so that step counts for e.g. debugger rate * limiting are accurate.
42281. A -> register of target object * B -> first register of value data (start_index, value1, value2, ..., valueN) * C -> number of key/value pairs (N)
42282. terminate
42283. unchanged by Duktape.Thread.resume()
42284. unchanged from Duktape.Thread.yield()
42285. A -> result reg * B -> object reg * C -> key reg/const
42286. **comment:** XXX: allow object to be a const, e.g. in 'foo'.toString()? * On the other hand, DUK_REGCONSTP() is slower and generates * more code.
 label: code-design
42287. invalidated
42288. [... val obj]
42289. env created above to stack top
42290. -> [Object res]
42291. Trigger at zero or below
42292. 'this' binding
42293. end switch
42294. Successful return: restore jmpbuf and return to caller.
42295. resume caller must be an ecmascript func
42296. y
42297. 'with' binding has no catch clause, so can't be here unless a normal try-catch
42298. caller must change active thread, and set thr->resumer to NULL
42299. unresolvable, no stack changes
42300. [... s1 s2] -> [... s1+s2]
42301. * Throw the error in the resumed thread's context; the * error value is pushed onto the resumee valstack. * * Note: the callstack of the target may empty in this case * too (i.e. the target thread has never been resumed). The * value stack will contain the initial function in that case, * which we simply ignore.
42302. Clamp so that values at 'clamp_top' and above are wiped and won't * retain reachable garbage. Then extend to 'nregs' because we're * returning to an Ecmascript function.
42303. **comment:** not caught by current thread, thread terminates (yield error to resumer); * note that this may cause a cascade if the resumer terminates with an uncaught * exception etc (this is OK, but needs careful testing)
 label: code-design
42304. Recompute argument count: bound function handling may have shifted.
42305. **comment:** XXX: declvar takes an duk_tval pointer, which is awkward and * should be reworked.
 label: code-design
42306. no prototype, updated below
42307. unwind to 'resume' caller
42308. guarantees entry_callstack_top - 1 >= 0
42309. may be changed
42310. Rethrow error to calling state.
42311. * Update the interrupt counter
42312. Stepped? Step out is handled by callstack unwind.
42313. **comment:** XXX: does not work if thr->catchstack is allocated but lowest pointer
 label: code-design
42314. relookup after side effects (no side effects currently however)
42315. resumee: [... initial_func undefined(= this) resume_value]
42316. must relookup act in case of side effects
42317. fastint downgrade check for return values

42318. XXX: optimize reconfig valstack operations so that resize, clamp, and setting * top are combined into one pass.
42319. **comment:** Figure out all active breakpoints. A breakpoint is * considered active if the current function's fileName * matches the breakpoint's fileName, AND there is no * inner function that has matching line numbers * (otherwise a breakpoint would be triggered both * inside and outside of the inner function which would * be confusing). Example: * * function foo() { * print('foo'); * function bar() { <-. breakpoints in these * print('bar'); | lines should not affect * } <-' foo() execution * bar(); * } * * We need a few things that are only available when * debugger support is enabled: (1) a line range for * each function, and (2) access to the function template to access the inner functions (and their * line ranges). * * It's important to have a narrow match for active * breakpoints so that we don't enter checked execution * when that's not necessary. For instance, if we're * running inside a certain function and there's * breakpoint outside in (after the call site), we * don't want to slow down execution of the function.
- label:** code-design
42320. pre-incremented, points to first jump slot
42321. nop
42322. * NEXTENUM checks whether the enumerator still has unenumerated * keys. If so, the next key is loaded to the target register * and the next instruction is skipped. Otherwise the next instruction * will be executed, jumping out of the enumeration loop.
42323. * Note: lj.value1 is 'value', lj.value2 is 'resume'. * This differs from YIELD.
42324. In-place unary operation.
42325. **comment:** Lookup current thread; use the stable 'entry_thread' for this to * avoid clobber warnings. Any valid, reachable 'thr' value would be * fine for this, so using 'entry_thread' is just to silence warnings.
- label:** code-design
42326. Thread may have changed, e.g. YIELD converted to THROW.
42327. Already declared but no initializer value * (e.g. 'var xyz;'), no-op.
42328. throw
42329. unreferenced without assertions
42330. For cross-checking during development: ensure dispatch count * matches cumulative interrupt counter init value sums.
42331. valstack should not need changes
42332. [-0x80000000,0x7fffffff]
42333. Catches both doubles and cases where only one argument is a fastint
42334. x == -Infinity
42335. * Note: prototype chain is followed BEFORE first comparison. This * means that the instanceof lval is never itself compared to the * rval.prototype property. This is apparently intentional, see E5 * Section 15.3.5.3, step 4.a. * * Also note: * * js> (function() {}) instanceof Function * true * js> Function instanceof Function * true * * For the latter, h_proto will be Function.prototype, which is the * built-in Function prototype. Because Function.[[Prototype]] is * also the built-in Function prototype, the result is true.
42336. heap pointer comparison suffices
42337. E5 Section 9.3.1
42338. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack (e.g. if * lval is already a string).
42339. truncate towards zero
42340. func support for [[HasInstance]] checked in the beginning of the loop
42341. prefix matches, lengths matter now
42342. Note: undefined from Section 11.8.5 always results in false * return (see e.g. Section 11.8.3) - hence special treatment here.
42343. normal comparison; known: * - both x and y are not NaNs (but one of them can be) * - both x and y are not zero (but one of them can be) * - x and y may be denormal or infinite
42344. mimic semantics for strings
42345. **comment:** At least 'magic' has a significant impact on function * identity.
- label:** code-design
42346. implementation specific
42347. x == +Infinity
42348. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack. * * The right hand side could be a light function (as they generally * behave like objects). Light functions never have a 'prototype' * property so E5.1 Section 15.3.5.3 step 3 always throws a TypeError. * Using duk_require_hobject() is thus correct (except for error msg).
42349. -> x in J-2**32, 2**32[
42350. String-number-buffer/object -> coerce object to primitive (apparently without hint), then try again.
42351. exposed, used by e.g. duk.bi_date.c
42352. Buffer/string -> compare contents.
42353. * ToNumber() (E5 Section 9.3) * * Value to convert must be on stack top, and is popped before exit. * * See: <http://www.cs.indiana.edu/~burger/FP-Printing-PLDI96.pdf> * <http://www.cs.indiana.edu/~burger/fp/index.html> * * Notes on the conversion: * * - There are specific requirements on the accuracy of the conversion * through a "Mathematical Value" (MV), so this conversion is not * trivial. * * - Quick rejects (e.g. based on first char) are difficult because * the grammar allows leading and trailing white space. * * - Quick reject based on string length is difficult even after * accounting for white space; there may be arbitrarily many * decimal digits. * * - Standard grammar allows decimal values ("123"), hex values * ("0x123") and infinities * * - Unlike source code literals, ToNumber() coerces empty strings * and strings with only whitespace to zero (not NaN).
42354. **comment:** Straightforward algorithm, makes fewer compiler assumptions.
- label:** code-design
42355. Handle lightfuncs through object coercion for now.
42356. **comment:** unused
- label:** code-design
42357. x = sign(x) * floor(abs(x)), i.e. truncate towards zero, keep sign
42358. Fast path for numbers (one of which may be a fastint)
42359. TypeError if rval is not an object (or lightfunc which should behave * like a Function instance).
42360. number
42361. radix
42362. Number/string-or-buffer -> coerce string to number (e.g. "'1.5' -> true).
42363. Undefined/null are considered equal (e.g. "null == undefined" -> true).
42364. **comment:** * ToString() (E5 Section 9.8) * * ==> implemented in the API.
- label:** requirement
42365. * 'func' is now a non-bound object which supports [[HasInstance]] * (which here just means DUK_HOBJECT_FLAG_CALLABLE). Move on * to execute E5 Section 15.3.5.3.
42366. **comment:** XXX: FP_ZERO check can be removed, the else clause handles it * correctly (preserving sign).
- label:** code-design
42367. **comment:** XXX: direct implementation
- label:** requirement
42368. SameValue(NaN, NaN) = true, regardless of NaN sign or extra bits
42369. For all combinations: +0 < +0, +0 < -0, -0 < +0, -0 < -0, * steps e, f, and g.
42370. **comment:** XXX: make fast paths optional for size minimization?
- label:** code-design
42371. **comment:** XXX: expensive, but numconv now expects to see a string
- label:** code-design
42372. * ToInteger() (E5 Section 9.4)
42373. invalidates tv
42374. * E5 Section 11.8.6 describes the main algorithm, which uses * [[HasInstance]]. [[HasInstance]] is defined for only * function objects: * * - Normal functions: * E5 Section 15.3.5.3 * - Functions established with Function.prototype.bind(): * E5 Section 15.3.4.5.3 * * For other objects, a TypeError is thrown. * * Limited

Proxy support: don't support 'getPrototypeOf' trap but * continue lookup in Proxy target if the value is a Proxy.

42375. never here

42376. Note that this is the same operation for strict and loose equality: * - E5 Section 11.9.3, step 1.c (loose) * - E5 Section 11.9.6, step 4 (strict)

42377. DUK_MEMCMP() is guaranteed to return zero (equal) for zero length * inputs so no zero length check is needed.

42378. $y == +\text{Infinity}$

42379. * Same type? * * Note: since number values have no explicit tag in the 8-byte * representation, need the awkward if + switch.

42380. **comment:** Note: reuse variables

label: code-design

42381. * CheckObjectCoercible() (E5 Section 9.10) * * Note: no API equivalent now.

42382. **comment:** Coerce like a string. This makes sense because addition also treats * buffers like strings.

label: code-design

42383. NOTE: fmod(x) result sign is same as sign of x, which * differs from what Javascript wants (see Section 9.6).

42384. equals and strict equals

42385. non-strict equality for buffers compares contents

42386. +(function(){}) -> NaN

42387. * Loose equality, strict equality, and SameValue (E5 Sections 11.9.1, 11.9.4, * 9.12). These have much in common so they can share some helpers. * * Future work notes: * * - Current implementation (and spec definition) has recursion; this should * be fixed if possible. * * - String-to-number coercion should be possible without going through the * value stack (and be more compact) if a shared helper is invoked.

42388. Called by duk_hstring.h macros

42389. Careful overflow handling. When multiplying by 10: * - 0x19999998 x 10 = 0xffffffff: no overflow, and adding * 0...9 is safe. * - 0x19999999 x 10 = 0xfffffff9: no overflow, adding * 0...5 is safe, 6...9 overflows. * - 0x1999999a x 10 = 0x10000004: always overflow.

42390. non-strict equality from here on

42391. +0.0

42392. Better equivalent algorithm. If the compiler is compliant, C and * Ecmascript semantics are identical for this particular comparison. * In particular, NaNs must never compare equal and zeroes must compare * equal regardless of sign. Could also use a macro, but this inlines * already nicely (no difference on gcc, for instance).

42393. * E5 Sections 11.8.7, 8.12.6. * * Basically just a property existence check using [[HasProperty]].

42394. Boolean/any -> coerce boolean to number and try again. If boolean is * compared to a pointer, the final comparison after coercion now always * yields false (as pointer vs. number compares to false), but this is * not special cased.

42395. -> [... lval new_rval]

42396. Using classification has smaller footprint than direct comparison.

42397. coerce lval with ToString()

42398. **comment:** * typeof * * E5 Section 11.4.3. * * Very straightforward. The only question is what to return for our * non-standard tag / object types. * * There is an unfortunate string constant define naming problem with * typeof return values for e.g. "Object" and "object"; careful with * the built-in string defines. The LC_XXX defines are used for the * lowercase variants now.

label: code-design

42399. IEEE requires that zeros compare the same regardless * of their signed, so if both x and y are zeroes, they * are caught above.

42400. * ToObject() (E5 Section 9.9) * * ==> implemented in the API.

42401. * ToBoolean() (E5 Section 9.2)

42402. follow prototype chain

42403. Difference to non-strict/strict comparison is that NaNs compare * equal and signed zero signs matter.

42404. Ordering should not matter (E5 Section 11.8.5, step 3.a) but * preserve it just in case.

42405. combined algorithm matching E5 Sections 9.5 and 9.6

42406. **comment:** XXX: this is possible without resorting to the value stack

label: code-design

42407. Similar to String comparison.

42408. Slow path

42409. **comment:** XXX: this is a very narrow check, and doesn't cover * zeroes, subnormals, infinities, which compare normally.

label: code-design

42410. **comment:** * Ecmascript specification algorithm and conversion helpers. * * These helpers encapsulate the primitive Ecmascript operation * semantics, and are used by the bytecode executor and the API * (among other places). Note that some primitives are only * implemented as part of the API and have no "internal" helper. * (This is the case when an internal helper would not really be * useful; e.g. the operation is rare, uses value stack heavily, * etc.) * * The operation arguments depend on what is required to implement * the operation: * * - If an operation is simple and stateless, and has no side * effects, it won't take an duk_hthread argument and its * arguments may be duk_tval pointers (which are safe as long * as no side effects take place). * * - If complex coercions are required (e.g. a "ToNumber" coercion) * or errors may be thrown, the operation takes an duk_hthread * argument. This also implies that the operation may have * arbitrary side effects, invalidating any duk_tval pointers. * * - For operations with potential side effects, arguments can be * taken in several ways: * * a) as duk_tval pointers, which makes sense if the "common case" * can be resolved without side effects (e.g. coercion); the * arguments are pushed to the valstack for coercion if * necessary * * b) as duk_tval values * * c) implicitly on value stack top * * d) as indices to the value stack * * Future work: * * - Argument styles may not be the most sensible in every case now. * * - In-place coercions might be useful for several operations, if * in-place coercion is OK for the bytecode executor and the API.

label: code-design

42411. recursive call for a primitive value (guaranteed not to cause second * recursion).

42412. **comment:** 'tv' becomes invalid

label: code-design

42413. $y == -\text{Infinity}$

42414. Always overflow.

42415. buffer values are coerced first to string here

42416. DUK_USE_FASTINT

42417. -> x in [-2**31,2**31[

42418. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)

42419. * [[DefaultValue]] (E5 Section 8.12.8) * * ==> implemented in the API.

42420. -> [... lval rval new_rval]

42421. * instanceof

42422. **comment:** * ToPrimitive() (E5 Section 9.1) * * ==> implemented in the API.

label: requirement

42423. Quite lenient, e.g. allow empty as zero, but don't allow trailing * garbage.

42424. IEEE requires that NaNs compare false

42425. * Comparisons ($x \geq y$, $x > y$, $x \leq y$, $x < y$) * * E5 Section 11.8.5: implement 'x < y' and then use negate and eval_left_first * flags to get the rest.

42426. Note: ToPrimitive(object,hint) == [[DefaultValue]](object,hint), * so use [[DefaultValue]] directly.

42427. XXX: this bound function resolution also happens elsewhere, * move into a shared helper.

42428. **comment:** * Note: of native Ecmascript objects, only Function instances * have a [[HasInstance]] internal property. Custom objects might * also have it, but not in current implementation. * * XXX: add a separate flag, DUK_HOBJECT_FLAG_ALLOW_INSTANCEEOF?

label: code-design

42429. [... lval rval]

42430. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

42431. range

42432. ToNumber(bool) is +1.0 or 0.0. Tagged boolean value is always 0 or 1.

42433. DUK_USE_PARANOIAD_MATH

42434. * For bound objects, [[HasInstance]] just calls the target function * [[HasInstance]]. If that is again a bound object, repeat until * we find a non-bound Function object.

42435. -> x in [0, 2**32[

42436. **comment:** * IsCallable() (E5 Section 9.11) ** XXX: API equivalent is a separate implementation now, and this has * currently no callers.
label: code-design

42437. [0x00000000, 0xffffffff]

42438. If flags != 0 (strict or SameValue), thr can be NULL. For loose * equals comparison it must be != NULL.

42439. **comment:** Note: not a typo, "object" is returned for a null value
label: documentation

42440. **comment:** XXX: fastint
label: code-design

42441. should be a safe way to compute this

42442. x in [2**31, 2**32[

42443. Note: this must match ToObject() behavior

42444. Coerce like boolean

42445. e.g. "x" < "xx"

42446. **comment:** XXX: The ES5/5.1/6 specifications require that the key in 'key in obj' * must be string coerced before the internal HasProperty() algorithm is * invoked. A fast path skipping coercion could be safely implemented for * numbers (as number-to-string coercion has no side effects). For ES6 * proxy behavior, the trap 'key' argument must be in a string coerced * form (which is a shame).
label: code-design

42447. 'It is true'

42448. *ToInt32(), ToUint32(), ToUint16() (E5 Sections 9.5, 9.6, 9.7)

42449. SameValue

42450. Nothing worked -> not equal.

42451. -> [...] lval rval]

42452. Accept 32-bit decimal integers, no leading zeroes, signs, etc. * Leading zeroes are not accepted (zero index "0" is an exception * handled above).

42453. the next 'if' is guaranteed to match after swap

42454. flags:nonstrict

42455. [...] lval rval(func)]

42456. **comment:** * Array index and length ** Array index: E5 Section 15.4 * Array length: E5 Section 15.4.5.1 steps 3.c - 3.d (array length write) ** The DUK_HSTRING_GET_ARRIDX_SLOW() and DUK_HSTRING_GET_ARRIDX_FAST() macros * call duk_js_to_arrayindex_string_helper().
label: code-design

42457. Fast path for fastints

42458. **comment:** return a specific NaN (although not strictly necessary)
label: code-design

42459. whole, won't clip

42460. * String comparison (E5 Section 11.8.5, step 4), which * needs to compare codepoint by codepoint. ** However, UTF-8 allows us to use strcmp directly: the shared * prefix will be encoded identically (UTF-8 has unique encoding) * and the first differing character can be compared with a simple * unsigned byte comparison (which strcmp does). ** This will not work properly for non-xutf-8 strings, but this * is not an issue for compliance.

42461. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: ** signbit(x) == signbit(y)

42462. check func supports [[HasInstance]] (this is checked for every function * in the bound chain, including the final one)

42463. * Types are different; various cases for non-strict comparison ** Since comparison is symmetric, we use a "swap trick" to reduce * code size.

42464. **comment:** XXX: this should probably just operate on the stack top, because it * needs to push stuff on the stack anyway...
label: code-design

42465. * in

42466. -> [...] lval rval rval.prototype]

42467. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

42468. apparently no hint?

42469. -> [...] closure template newobj closure]

42470. * Make a value copy of the input val. This ensures that * side effects cannot invalidate the pointer.

42471. func is NULL for lightfuncs

42472. **comment:** XXX: any way to detect faster whether something needs to be closed? * We now look up _Callee and then skip the rest.
label: requirement

42473. formal arg limits

42474. Note: getprop may invoke any getter and invalidate any * duk_tval pointers, so this must be done first.

42475. * GETVAR ** See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is GetValue'd] * 8.7.1 GetValue (V) * 8.12.1 [[GetOwnProperty]] (P) * 8.12.2 [[GetProperty]] (P) * 8.12.3 [[Get]] (P) ** If 'throw' is true, always leaves two values on top of stack: [val this]. ** If 'throw' is false, returns 0 if identifier cannot be resolved, and the * stack will be unaffected in this case. If identifier is resolved, returns * 1 and leaves [val this] on top of stack. ** Note: the 'strict' flag of a reference returned by GetIdentifierReference * is ignored by GetValue. Hence we don't take a 'strict' parameter. ** The 'throw' flag is needed for implementing 'typeof' for an unreferenced * identifier. An unreference identifier in other contexts generates a * ReferenceError.

42476. * In strict mode E5 protects 'eval' and 'arguments' from being * assigned to (or even declared anywhere). Attempt to do so * should result in a compile time SyntaxError. See the internal * design documentation for details. ** Thus, we should never come here, run-time, for strict code, * and name 'eval' or 'arguments'.

42477. **comment:** * Prototype walking starting from 'env'. ** ('act' is not needed anywhere here.)
label: code-design

42478. [...] closure template]

42479. [...] closure template env]

42480. **comment:** * "prototype" is, by default, a fresh object with the "constructor" * property. ** Note that this creates a circular reference for every function * instance (closure) which prevents refcount-based collection of * function instances. ** XXX: Try to avoid creating the default prototype object, because * many functions are not used as constructors and the default * prototype is unnecessary. Perhaps it could be created on-demand * when it is first accessed?
label: code-design

42481. [holder name val] -> [holder]

42482. NB: 'val' may be invalidated here because put_value may realloc valstack, * caller beware.

42483. * Finish

42484. **comment:** * "arguments" and "caller" must be mapped to throwers for strict * mode and bound functions (E5 Section 15.3.5). ** XXX: This is expensive to have for every strict function instance. * Try to implement as virtual properties or on-demand created properties.
label: code-design

42485. lookup name from an open declarative record's registers

42486. env and act may be NULL

42487. [...] env callee]

42488. registers are mutable, non-deletable

42489. side effects

42490. -> [...] closure template newobj]

42491. * Local result type for duk_get_identifier_reference() lookup.

42492. Identifier found in registers (always non-deletable) * or declarative environment record and non-configurable.

42493. [...] env callee varmap key val]

42494. -> [...] closure template]

42495. * PUTVAR ** See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is PutValue'd] * 8.7.2 PutValue (V,W) [see especially step 3.b, undefined -> automatic global in non-strict mode] * 8.12.4 [[CanPut]] (P) * 8.12.5 [[Put]] (P) ** Note: may invalidate any valstack (or object) duk_tval pointers because * putting a value may reallocate any object or any valstack. Caller beware.

42496. must be found: was found earlier, and cannot be inherited

42497. * Not found: write to global object (non-strict) or ReferenceError * (strict); see E5 Section 8.7.2, step 3.

42498. [... closure]

42499. [... env]

42500. [this value] -> [value this]

42501. * Object environment record. * * Binding (target) object is an external, uncontrolled object. * Identifier may be bound in an ancestor property, and may be * an accessor. Target can also be a Proxy which we must support * here.

42502. non-standard

42503. Note: in strict mode the compiler should reject explicit * declaration of 'eval' or 'arguments'. However, internal * bytecode may declare 'arguments' in the function prologue. * We don't bother checking (or asserting) for these now.

42504. always for register bindings

42505. If function creation fails due to out-of-memory, the data buffer * pointer may be NULL in some cases. That's actually possible for * GC code, but shouldn't be possible here because the incomplete * function will be unwound from the value stack and never instantiated.

42506. can't get value, may be accessor

42507. unsigned

42508. writable but not deletable

42509. may be NULL

42510. assertions: env must be closed in the same thread as where it runs

42511. * "length" maps to number of formals (E5 Section 13.2) for function * declarations/expressions (non-bound functions). Note that 'nargs' * is NOT necessarily equal to the number of arguments.

42512. 1 -> needs a PUTVAR

42513. just check 'env'

42514. It's important that duk_xdef_prop() is a 'raw define' so that any * properties in an ancestor are never an issue (they should never be * e.g. non-writable, but just in case).

42515. * Named function expression, name needs to be bound * in an intermediate environment record. The "outer" * lexical/variable environment will thus be: * * a) { funcname: <func>, __prototype: outer_lex_env } * b) { funcname: <func>, __prototype: <globalenv> } (if outer_lex_env missing)

42516. global object doesn't have array part

42517. * Define new property * * Note: this may fail if the holder is not extensible.

42518. * HASVAR: check identifier binding from a given environment record * without traversing its parents. * * This primitive is not exposed to user code as such, but is used * internally for e.g. declaration binding instantiation. * * See E5 Sections: * 10.2.1.1 HasBinding(N) * 10.2.1.2.1 HasBinding(N) * * Note: strictness has no bearing on this check. Hence we don't take * a 'strict' parameter.

42519. lookup name from current activation record's functions' registers

42520. already closed

42521. -> [... closure template env]

42522. follow parent chain

42523. **comment:** XXX: this is awkward as we use an internal method which doesn't handle * extensibility etc correctly. Basically we'd want to do a [[DefineOwnProperty]] * or Object.defineProperty() here.
label: code-design

42524. **comment:** * Compact the closure, in most cases no properties will be added later. * Also, without this the closures end up having unused property slots * (e.g. in Duktape 0.9.0, 8 slots would be allocated and only 7 used). * A better future solution would be to allocate the closure directly * to correct size (and setup the properties directly without going * through the API).
label: code-design

42525. required if NAMEBINDING set

42526. [... closure template formals]

42527. env[funcname] = closure

42528. Note: we rely on the _Varmap having a bunch of nice properties, like: * - being compacted and unmodified during this process * - not containing an array part * - having correct value types

42529. * GETIDREF: a GetIdentifierReference-like helper. * * Provides a parent traversing lookup and a single level lookup * (for HasBinding). * * Instead of returning the value, returns a bunch of values allowing * the caller to read, write, or delete the binding. Value pointers * are duk_tval pointers which can be mutated directly as long as * refcounts are properly updated. Note that any operation which may * reallocate valstacks or compact objects may invalidate the returned * duk_tval (but not object) pointers, so caller must be very careful. * * If starting environment record 'env' is given, 'act' is ignored. * However, if 'env' is NULL, the caller may identify, in 'act', an * activation which hasn't had its declarative environment initialized * yet. The activation registers are then looked up, and its parent * traversed normally. * * The 'out' structure values are only valid if the function returns * success (non-zero).

42530. Update duk_tval in-place if pointer provided and the * property is writable. If the property is not writable * (immutable binding), use duk_hobject_putprop() which * will respect mutability.

42531. regnum is sane

42532. DUK_HOBJECT_FLAG_ARRAY_PART: don't care

42533. XXX: additional conditions when to close variables? we don't want to do it * unless the environment may have "escaped" (referenced in a function closure). * With delayed environments, the existence is probably good enough of a check.

42534. Note: all references inside 'data' need to get their refcounts * upped too. This is the case because refcounts are decreased * through every function referencing 'data' independently.

42535. * Lookup variable and update in-place if found.

42536. -> [... closure template env funcname]

42537. [... closure template val]

42538. **comment:** order: most frequent to least frequent
label: code-design

42539. * Not found in registers, proceed to the parent record. * Here we need to determine what the parent would be, * if 'env' was not NULL (i.e. same logic as when initializing * the record). * * Note that environment initialization is only deferred when * DUK_HOBJECT_HAS_NEWENV is set, and this only happens for: * - Function code * - Strict eval code * * We only need to check _Lexenv here; _Varev exists only if it * differs from _Lexenv (and thus _Lexenv will also be present).

42540. * Not found (even in global object). * * In non-strict mode this is a silent SUCCESS (!), see E5 Section 11.4.1, * step 3.b. In strict mode this case is a compile time SyntaxError so * we should not come here.

42541. XXX: incrf by count (here 2 times)

42542. * Not found (in registers or record objects). Declare * to current variable environment.

42543. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway

42544. implicit this value always undefined for * declarative environment records.

42545. * Conceptually, we look for the identifier binding by starting from * 'env' and following to chain of environment records (represented * by the prototype chain). * * If 'env' is NULL, the current activation does not yet have an * allocated declarative environment record; this should be treated * exactly as if the environment record existed but had no bindings * other than register bindings. * * Note: we assume that with the DUK_HOBJECT_FLAG_NEWENV cleared * the environment will always be initialized immediately; hence * a NULL 'env' should only happen with the flag set. This is the * case for: (1) function calls, and (2) strict, direct eval calls.

42546. * Not found (even in global object)

42547. * Create a new function object based on a "template function" which contains * compiled bytecode, constants, etc, but lacks a lexical environment. * * ECMAScript requires that each created closure is a separate object, with * its own set of editable properties. However, structured property values * (such as the formal arguments list and the variable map) are shared. * Also the bytecode, constants, and inner functions are shared. * * See E5 Section 13.2 for detailed requirements

on the function objects; * there are no similar requirements for function "templates" which are an * implementation dependent internal feature. Also see function-objects.rst * for a discussion on the function instance properties provided by this * implementation. ** Notes: *** Order of internal properties should match frequency of use, since the * properties will be linearly scanned on lookup (functions usually don't * have enough properties to warrant a hash part). *** The created closure is independent of its template; they do share the * same 'data' buffer object, but the template object itself can be freed * even if the closure object remains reachable.

42548. **comment:** * Identifier access and function closure handling. ** Provides the primitives for slow path identifier accesses: GETVAR, * PUTVAR, DELVAR, etc. The fast path, direct register accesses, should * be used for most identifier accesses. Consequently, these slow path * primitives should be optimized for maximum compactness. ** Ecmascript environment records (declarative and object) are represented * as internal objects with control keys. Environment records have a * parent record ("outer environment reference") which is represented by * the implicit prototype for technical reasons (in other words, it is a * convenient field). The prototype chain is not followed in the ordinary * sense for variable lookups. ** See identifier-handling.rst for more details on the identifier algorithms * and the internal representation. See function-objects.rst for details on * what function templates and instances are expected to look like. ** Care must be taken to avoid duk_tval pointer invalidation caused by * e.g. value stack or object resizing. ** TODO: properties for function instances could be initialized much more * efficiently by creating a property allocation for a certain size and * filling in keys and values directly (and INCREFing both with "bulk incref" * primitives. ** XXX: duk_hobject_getprop() and duk_hobject_putprop() calls are a bit * awkward (especially because they follow the prototype chain); rework * if "raw" own property helpers are added.

label: code-design

42549. **comment:** * Variable already declared, ignore re-declaration. * The only exception is the updated behavior of E5.1 for * global function declarations, E5.1 Section 10.5, step 5.e. * This behavior does not apply to global variable declarations.

label: code-design

42550. assume plain values

42551. SCANBUILD: scan-build produces a NULL pointer dereference warning * below; it never actually triggers because holder is actually never * NULL.

42552. * Function gets no new environment when called. This is the * case for global code, indirect eval code, and non-strict * direct eval code. There is no direct correspondence to the * E5 specification, as global/eval code is not exposed as a * function.

42553. * Some assertions (E5 Section 13.2).

42554. SCANBUILD: NULL pointer dereference, doesn't actually trigger, * asserted above.

42555. ref.value and ref.this.binding invalidated here

42556. * DELVAR ** See E5 Sections: * 11.4.1 The delete operator * 10.2.1.1.5 DeleteBinding (N) [declarative environment record] * 10.2.1.2.5 DeleteBinding (N) [object environment record] * * Variable bindings established inside eval() are deletable (configurable), * other bindings are not, including variables declared in global level. * Registers are always non-deletable, and the deletion of other bindings * is controlled by the configurable flag. ** For strict mode code, the 'delete' operator should fail with a compile * time SyntaxError if applied to identifiers. Hence, no strict mode * run-time deletion of identifiers should ever happen. This function * should never be called from strict mode code!

42557. * Other cases (function declaration, anonymous function expression, * strict direct eval code). The "outer" environment will be whatever * the caller gave us.

42558. **comment:** * Special behavior in E5.1. * * Note that even though parents == 0, the conflicting property * may be an inherited property (currently our global object's * prototype is Object.prototype). Step 5.e first operates on * the existing property (which is potentially in an ancestor) * and then defines a new property in the global object (and * never modifies the ancestor). * * Also note that this logic would become even more complicated * if the conflicting property might be a virtual one. Object * prototype has no virtual properties, though. ** XXX: this is now very awkward, rework.

label: code-design

42559. * Setup environment record properties based on the template and * its flags. * * If DUK_HOBJECT_HAS_NEWENV(fun_temp) is true, the environment * records represent identifiers "outside" the function; the * "inner" environment records are created on demand. Otherwise, * the environment records are those that will be directly used * (e.g. for declarations). * * _Lexenv is always set; _Varenv defaults to _Lexenv if missing, * so _Varenv is only set if _Lexenv != _Varenv. * * This is relatively complex, see doc/identifier-handling.rst.

42560. **comment:** XXX: we could save space by using _Target OR _This. If _Target, assume * this binding is undefined. If _This, assumes this binding is _This, and * target is also _This. One property would then be enough.

label: code-design

42561. Push a new closure on the stack. ** Note: if fun_temp has NEWENV, i.e. a new lexical and variable declaration * is created when the function is called, only outer_lex_env matters * (outer_var_env is ignored and may or may not be same as outer_lex_env).

42562. * Declarative environment record. * * Identifiers can never be stored in ancestors and are * always plain values, so we can use an internal helper * and access the value directly with an duk_tval ptr. * * A closed environment is only indicated by it missing * the "book-keeping" properties required for accessing * register-bound variables.

42563. **comment:** * "name" is a non-standard property found in at least V8, Rhino, smjs. * For Rhino and smjs it is non-writable, non-enumerable, and non-configurable; * for V8 it is writable, non-enumerable, non-configurable. It is also defined * for an anonymous function expression in which case the value is an empty string. * We could also leave name 'undefined' for anonymous functions but that would * differ from behavior of other engines, so use an empty string. ** XXX: make optional? costs something per function.

label: code-design

42564. **comment:** XXX: use helper for size optimization

label: code-design

42565. * Delayed activation environment record initialization (for functions * with NEWENV). ** The non-delayed initialization is handled by duk_handle_call().

42566. [... closure template undefined]

42567. **comment:** XXX: could also copy from template, but there's no way to have any * other value here now (used code has no access to the template).

label: code-design

42568. Note: env_thr != thr is quite possible and normal, so careful * with what thread is used for valstack lookup.

42569. ref.value, ref.this.binding invalidated here by getprop call

42570. lookup results is ignored

42571. **comment:** XXX: duk_hobject_hasprop() would be correct for * non-Proxy objects too, but it is about ~20-25% * slower at present so separate code paths for * Proxy and non-Proxy now.

label: code-design

42572. [... env callee varmap]

42573. env is closed, should be missing _Callee, _Thread, _Regbase

42574. **comment:** * Copy some internal properties directly * * The properties will be writable and configurable, but not enumerable.

label: code-design

42575. **comment:** XXX: use a helper for prototype traversal; no loop check here

label: code-design

42576. ref.holder is global object, holder is the object with the * conflicting property.

42577. * Get holder object

42578. for object-bound identifiers

42579. DUK_HOBJECT_FLAG_NEWENV: handled below

42580. Target may be a Proxy or property may be an accessor, so we must * use an actual, Proxy-aware hasprop check here. ** out->holder is NOT set to the actual duk_hobject where the * property is found, but rather the object binding target object.

42581. **comment:** unused

label: code-design

42582. holder will be set to the target object, not the actual object * where the property was found (see duk_get_identifier_reference()).

42583. * Delayed env creation check

42584. * Delayed initialization only occurs for 'NEWENV' functions.

42585. no prototype, updated below

42586. [... closure template len_value]

42587. [this value]

42588. Here val would be potentially invalid if we didn't make * a value copy at the caller.

42589. **comment:** XXX: slightly awkward

label: code-design

42590. implicit this value always undefined for * declarative environment records.

42591. property attributes for identifier (relevant if value != NULL)

42592. 0 = no throw

42593. * Closing environment records. ** The environment record MUST be closed with the thread where its activation * is. In other words (if 'env' is open): ** - 'thr' must match _env.thread * - 'func' must match _env.callee * - 'regbase' must match _env.regbase ** These are not looked up from the env to minimize code size. ** XXX: should access the own properties directly instead of using the API

42594. * Init/assert flags, copying them where appropriate. Some flags * (like NEWENV) are processed separately below.

42595. compact the prototype

42596. assume keys are compacted

42597. * Try registers

42598. * DECLVAR ** See E5 Sections: * 10.4.3 Entering Function Code * 10.5 Declaration Binding Instantiation * 12.2 Variable Statement * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution ** Variable declaration behavior is mainly discussed in Section 10.5, * and is not discussed in the execution semantics (Sections 11-13). ** Conceptually declarations happen when code (global, eval, function) * is entered, before any user code is executed. In practice, register- * bound identifiers are 'declared' automatically (by virtue of being * allocated to registers with the initial value 'undefined'). Other * identifiers are declared in the function prologue with this primitive. ** Since non-register bindings eventually back to an internal object's * properties, the 'prop_flags' argument is used to specify binding * type: ** - Immutable binding: set DUK_PROPDESC_FLAG_WRTABLE to false * - Non-deletable binding: set DUK_PROPDESC_FLAG_CONFIGURABLE to false * - The flag DUK_PROPDESC_FLAG_ENUMERABLE should be set, although it * doesn't really matter for internal objects * All bindings are non-deletable mutable bindings except: ** - Declarations in eval code (mutable, deletable) * - 'arguments' binding in strict function code (immutable) * - Function name binding of a function expression (immutable) ** Declarations may go to declarative environment records (always * so for functions), but may also go to object environment records * (e.g. global code). The global object environment has special * behavior when re-declaring a function (but not a variable); see * E5.1 specification, Section 10.5, step 5.e. ** Declarations always go to the 'top-most' environment record, i.e. * we never check the record chain. It's not an error even if a * property (even an immutable or non-deletable one) of the same name * already exists. ** If a declared variable already exists, its value needs to be updated * (if possible). Returns 1 if a PUTVAR needs to be done by the caller; * otherwise returns 0.

42599. for register-bound and declarative env identifiers

42600. [... closure template name]

42601. * Check whether already declared. ** We need to check whether the binding exists in the environment * without walking its parents. However, we still need to check * register-bound identifiers and the prototype chain of an object * environment target object.

42602. bound functions are never in act 'func'

42603. **comment:** XXX: copy flags using a mask**label:** code-design

42604. Although NAMEBINDING is not directly needed for using * function instances, it's needed by bytecode dump/load * so copy it too.

42605. **comment:** XXX: more accurate?**label:** code-design

42606. irrelevant when out->value == NULL

42607. **comment:** XXX: if duk_hobject_define_property_internal() was updated * to handle a pre-existing accessor property, this would be * a simple call (like for the ancestor case).**label:** code-design

42608. Note: not set in template (has no "prototype")

42609. -> [... closure template env funcname closure]

42610. assume value is a number

42611. **comment:** XXX: what to do if _Formals is not empty but compiler has * optimized it away -- read length from an explicit property * then?**label:** code-design

42612. if property already exists, overwrites silently

42613. Note: val is a stable duk_tvval pointer. The caller makes * a value copy into its stack frame, so 'tv_val' is not subject * to side effects here.

42614. parent env is the prototype

42615. open scope information, for compiled functions only

42616. shared helper

42617. explicit PropertyList

42618. How large a loop detection stack to use

42619. gap (if empty string, NULL)

42620. * Defines for JSON, especially duk_bi_json.c.

42621. How much stack to require on entry to object/array decode

42622. indexed by recursion_depth

42623. type bit mask: types which certainly produce 'undefined'

42624. escape any non-ASCII characters

42625. Encoding/decoding flags

42626. extended types: compatible encoding

42627. extended types: custom encoding

42628. Encoding state. Heap object references are all borrowed.

42629. valstack index of loop detection object

42630. avoid key quotes when key is an ASCII Identifier

42631. DUK_JSON_H_INCLUDED

42632. replacer function

42633. How much stack to require on entry to object/array encode

42634. output bufwriter

42635. '='

42636. Store lexer position, restoring if quantifier is invalid.

42637. * Set lexer input position and reinitialize lookup window.

42638. '}'

42639. '.' verbatim

42640. One digit octal escape, digit validated.

42641. (

42642. DUK_USE_REGEXP_SUPPORT

42643. escaped NonEscapeCharacter

42644. The E5.1 specification does not seem to allow IdentifierPart characters * to be used as identity escapes. Unfortunately this includes '\$', which * cannot be escaped as '\$'; it needs to be escaped e.g. as '\u0024'. * Many other implementations (including V8 and Rhino, for instance) do * accept '\$' as a valid identity escape, which is quite pragmatic. * See: test-regexp-identity-escape-dollar.js.

42645. mark range 'direct', bypass canonicalization (see Wiki)

42646. **comment:** * Lexer for source files, ToNumber() string conversions, RegExp expressions, * and JSON. ** Provides a stream of ECMAScript tokens from an UTF-8/CESU-8 buffer. The * caller can also rewind the token stream into a certain position which is * needed by the compiler part for multi-pass scanning. Tokens are * represented as duk_token structures, and contain line number information. * Token types are identified with DUK_TOK_* defines. ** Characters are decoded into a fixed size lookup window consisting of * decoded Unicode code points, with window positions past the end of the * input filled with an invalid codepoint (-1). The tokenizer can thus * perform multiple character lookups efficiently and with few sanity * checks (such as access outside the end of the input), which keeps the * tokenization code small at the cost of performance. ** Character data in tokens, such as identifier names and string literals, * is encoded into CESU-8 format on-the-fly while parsing the token in * question. The string data is made reachable to garbage collection by * placing the token-related values in value stack entries allocated for * this purpose by the caller. The characters exist in Unicode code point * form only in the fixed size lookup window, which keeps character data * expansion (of especially ASCII data) low. ** Token parsing supports the full range of Unicode characters as described * in the E5 specification. Parsing has been

optimized for ASCII characters * because ordinary Ecmascript code consists almost entirely of ASCII * characters. Matching of complex Unicode codepoint sets (such as in the * IdentifierStart and IdentifierPart productions) is optimized for size, * and is done using a linear scan of a bit-packed list of ranges. This is * very slow, but should never be entered unless the source code actually * contains Unicode characters. * * Ecmascript tokenization is partially context sensitive. First, * additional future reserved words are recognized in strict mode (see E5 * Section 7.6.1.2). Second, a forward slash character ('/') can be * recognized either as starting a RegExp literal or as a division operator, * depending on context. The caller must provide necessary context flags * when requesting a new token. * * Future work: * * * Make line number tracking optional, as it consumes space. * * * Add a feature flag for disabling UTF-8 decoding of input, as most * source code is ASCII. Because of Unicode escapes written in ASCII, * this does not allow Unicode support to be removed from e.g. * duk_unicode_is_identifier_start() nor does it allow removal of CESU-8 * encoding of e.g. string literals. * * * Add a feature flag for disabling Unicode compliance of e.g. identifier * names. This allows for a build more than a kilobyte smaller, because * Unicode ranges needed by duk_unicode_is_identifier_start() and * duk_unicode_is_identifier_part() can be dropped. String literals * should still be allowed to contain escaped Unicode, so this still does * not allow removal of CESU-8 encoding of e.g. string literals. * * * Character lookup tables for codepoints above BMP could be stripped. * * * Strictly speaking, E5 specification requires that source code consists * of 16-bit code units, and if not, must be conceptually converted to * that format first. The current lexer processes Unicode code points * and allows characters outside the BMP. These should be converted to * surrogate pairs while reading the source characters into the window, * not after tokens have been formed (as is done now). However, the fix * is not trivial because two characters are decoded from one codepoint. * * * Optimize for speed as well as size. Large if-else ladders are (at * least potentially) slow.

label: code-design

42647. * Parse a RegExp token. The grammar is described in E5 Section 15.10. * Terminal constructions (such as quantifiers) are parsed directly here. * * 0xffffffffU is used as a marker for "infinity" in quantifiers. Further, * DUK_MAX_RE_QUANT_DIGITS limits the maximum number of digits that * will be accepted for a quantifier.

42648. "/=" and not in regexp mode

42649. Note: if DecimalLiteral starts with a '0', it can only be * followed by a period or an exponent indicator which starts * with 'e' or 'E'. Hence the if-check above ensures that * OctalIntegerLiteral is the only valid NumericLiteral * alternative at this point (even if y is, say, '9').

42650. 'C'

42651. * Intern the temporary byte buffer into a valstack slot * (in practice, slot1 or slot2).

42652. Fast path.

42653. '/'

42654. LF line terminator; CR LF and Unicode lineterms are handled in slow path

42655. line tracking

42656. * Interned identifier is compared against reserved words, which are * currently interned into the heap context. See genbuiltins.py. * * Note that an escape in the identifier disables recognition of * keywords; e.g. "\u0069f = 1;" is a valid statement (assigns to * identifier named "if"). This is not necessarily compliant, * see test-dec-escaped-char-in-keyword.js. * * Note: "get" and "set" are awkward. They are not officially * ReservedWords (and indeed e.g. "var set = 1;" is valid), and * must come out as DUK_TOK_IDENTIFIER. The compiler needs to * work around this a bit.

42657. val1 = count

42658. '{'

42659. 0xxx xxxx -> fast path

42660. radix

42661. having this as a separate function provided a size benefit

42662. * E5 Section 7.4. If the multi-line comment contains a newline, * it is treated like a single line terminator for automatic * semicolon insertion.

42663. second, parse flags

42664. * Since character data is only generated by decoding the source or by * the compiler itself, we rely on the input codepoints being correct * and avoid a check here. * * Character data can also come here through decoding of Unicode * escapes ("udead\ubef") so all 16-bit unsigned values can be * present, even when the source file itself is strict UTF-8.

42665. **comment:** * (Re)initialize the temporary byte buffer. May be called extra times * with little impact.

label: code-design

42666. **comment:** XXX: because of the final check below (that the literal is not * followed by a digit), this could maybe be simplified, if we bail * out early from a leading zero (and if there are no periods etc). * Maybe too complex.

label: code-design

42667. dash is already 0

42668. * Various defines and file specific helper macros

42669. multi-character sets not allowed as part of ranges, see * E5 Section 15.10.2.15, abstract operation CharacterRange.

42670. '?'

42671. re-lookup first char on first loop

42672. '\"'

42673. state == 3

42674. * Matching order: * * Punctuator first chars, also covers comments, regexps * LineTerminator * Identifier or reserved word, also covers null/true/false literals * NumericLiteral * StringLiteral * EOF * * The order does not matter as long as the longest match is * always correctly identified. There are order dependencies * in the clauses, so it's not trivial to convert to a switch.

42675. ','

42676. Automatic semicolon insertion is allowed if a token is preceded * by line terminator(s), or terminates a statement list (right curly * or EOF).

42677. no point in supporting encodings of 5 or more bytes

42678. line continuation

42679. Use temporaries and update lex_ctx only when finished.

42680. '|'

42681. check that byte has the form 10xx xxxx

42682. lookup for 0x00a above assumes shortest encoding now

42683. unsigned

42684. form: { DecimalDigits , }, val1 = min count

42685. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

42686. note: long live range

42687. 1110 xxxx 10xx xxxx 10xx xxxx

42688. 110x xxxx 10xx xxxx

42689. 0=before period/exp, * 1=after period, before exp * 2=after exp, allow '+' or '-' * 3=after exp and exp sign

42690. re-lookup curr char on first round

42691. NB: duk_lexer_getpoint() is a macro only

42692. lexer character window helpers

42693. eat closing quote

42694. avoid multiply

42695. Not enough data to provide a full window, so "scroll" window to * start of buffer and fill up the rest.

42696. * Parse Ecmascript source InputElementDiv or InputElementRegExp * (E5 Section 7), skipping whitespace, comments, and line terminators. * * Possible results are: * (1) a token * (2) a line terminator (skipped) * (3) a comment (skipped) * (4) EOF * * White space is automatically skipped from the current position (but * not after the input element). If input has already ended, returns * DUK_TOK_EOF indefinitely. If a parse error occurs, uses an DUK_ERROR() * macro call (and hence a longjmp through current heap longjmp context). * Comments and line terminator tokens are automatically skipped. * * The input element being matched is determined by regexp_mode; if set, * parses a InputElementRegExp, otherwise a InputElementDiv. The * difference between these are handling of productions starting with a * forward slash. * * If strict_mode is set, recognizes additional future reserved words * specific to strict mode, and refuses to parse octal literals. * * The matching strategy below is to (currently) use a six character * lookup window to quickly determine which production is the -longest- * matching one, and then parse that. The top-level if-else clauses * match the first character, and the code blocks for each clause * handle -all- alternatives for that first character. Ecmascript * specification uses the "longest match wins" semantics, so the order * of the if-clauses matters. * * Misc notes: * * * Ecmascript numeric literals do not accept a

sign character. * Consequently e.g. "-1.0" is parsed as two tokens: a negative * sign and a positive numeric literal. The compiler performs * the negation during compilation, so this has no adverse impact. * * * There is no token for "undefined": it is just a value available * from the global object (or simply established by doing a reference * to an undefined value). * * * Some contexts want Identifier tokens, which are IdentifierNames * excluding reserved words, while some contexts want IdentifierNames * directly. In the latter case e.g. "while" is interpreted as an * identifier name, not a DUK_TOK_WHILE token. The solution here is * to provide both token types: DUK_TOK_WHILE goes to 't' while * DUK_TOK_IDENTIFIER goes to 't_noes', and 'slot1' always contains * the identifier / keyword name. * * * Directive prologue needs to identify string literals such as * "use strict" and 'use strict', which are sensitive to line * continuations and escape sequences. For instance, "use\u0020strict" * is a valid directive but is distinct from "use strict". The solution * here is to decode escapes while tokenizing, but to keep track of the * number of escapes. Directive detection can then check that the * number of escapes is zero. * * * Multi-line comments with one or more internal LineTerminator are * treated like a line terminator to comply with automatic semicolon * insertion.

42697. got lineterm preceding non-whitespace, non-lineterm token

42698. **comment:** XXX: this duplicates functionality in duk_reEXP.c where a similar loop is * required anyway. We could use that BUT we need to update the regexp compiler * 'nranges' too. Work this out a bit more cleanly to save space.

label: code-design

42699. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx

42700. Call with count == DUK_LEXER_WINDOW_SIZE to fill buffer initially.

42701. **comment:** XXX: shared parsing?

label: code-design

42702. CR LF again a special case

42703. **comment:** XXX: logic for handling character ranges is now incorrect, it will accept * e.g. [\d-z] whereas it should croak from it? SMJS accepts this too, though. *

* Needs a read through and a lot of additional tests.

label: code-design

42704. * E5 Section 7.3: CR LF is detected as a single line terminator for * line numbers. Here we also detect it as a single line terminator * token.

42705. Track number of escapes: necessary for proper keyword * detection.

42706. (?=

42707. re-read to avoid spill / fetch

42708. Note: decimal number may start with a period, but must be followed by a digit

42709. (?:

42710. '}'

42711. **comment:** initialized to avoid warnings of unused var

label: code-design

42712. Three digit octal escape, digits validated.

42713. validation performed by duk_hexval

42714. validation of the regexp is caller's responsibility

42715. character represents itself

42716. '~'

42717. '['

42718. IdentifierStart is stricter than IdentifierPart, so if the first * character is escaped, must have a stricter check here.

42719. '%'

42720. **comment:** not strictly necessary, but avoids "uninitialized variable" warnings

label: code-design

42721. * To avoid creating a heavy intermediate value for the list of ranges, * only the start token ('[or '[^') is parsed here. The regexp * compiler parses the ranges itself.

42722. Note: first character is checked against this. But because * IdentifierPart includes all IdentifierStart characters, and * the first character (if unescaped) has already been checked * in the if condition, this is OK.

42723. **comment:** init is unnecessary but suppresses "may be used uninitialized" warnings

label: code-design

42724. **comment:** temporary, must be signed and 32-bit to hold Unicode code points

label: code-design

42725. eat closing slash

42726. not strictly necessary because of lookahead '}' above

42727. DUK_L0 -> '\ char * DUK_L1 ... DUK_L5 -> more lookup

42728. '+'

42729. default: char escape (two chars)

42730. * Parse an identifier and then check whether it is: * - reserved word (keyword or other reserved word) * - "null" (NullLiteral) * - "true" (BooleanLiteral) * - "false" (BooleanLiteral) * - anything else => identifier * * This does not follow the E5 productions cleanly, but is * useful and compact. * * Note that identifiers may contain Unicode escapes, * see E5 Sections 6 and 7.6. They must be decoded first, * and the result checked against allowed characters. * The above if-clause accepts an identifier start and an * '\ character -- no other token can begin with a '\'. * * Note that "get" and "set" are not reserved words in E5 * specification so they are recognized as plain identifiers * (the tokens DUK_TOK_GET and DUK_TOK_SET are actually not * used now). The compiler needs to work around this. * * Strictly speaking, following Ecmascript longest match * specification, an invalid escape for the first character * should cause a syntax error. However, an invalid escape * for IdentifierParts should just terminate the identifier * early (longest match), and let the next tokenization * fail. For instance Rhino croaks with 'foo\z' when * parsing the identifier. This has little practical impact.

42731. adv = 2 default OK

42732. **comment:** XXX: the handling of character range detection is a bit convoluted. * Try to simplify and make smaller.

label: code-design

42733. * Special parser for character classes; calls callback for every * range parsed and returns the number of ranges present.

42734. fill window

42735. Failed to match the quantifier, restore lexer and parse * opening brace as a literal.

42736. '<'

42737. 10xx xxxx -> invalid

42738. * "/" followed by something in regexp mode. See E5 Section 7.8.5. * * RegExp parsing is a bit complex. First, the regexp body is delimited * by forward slashes, but the body may also contain forward slashes as * part of an escape sequence or inside a character class (delimited by * square brackets). A mini state machine is used to implement these. * * Further, an early (parse time) error must be thrown if the regexp * would cause a run-time error when used in the expression new RegExp(...). * Parsing here simply extracts the (candidate) regexp, and also accepts * invalid regular expressions (which are delimited properly). The caller * (compiler) must perform final validation and regexp compilation. * * RegExp first char may not be '/' (single line comment) or '*' (multi- * line comment). These have already been checked above, so there is no * need below for special handling of the first regexp character as in * the E5 productions. * * About unicode escapes within regexp literals: * * E5 Section 7.8.5 grammar does NOT accept \uHHHH escapes. * However, Section 6 states that regexps accept the escapes, * see paragraph starting with "In string literals...". * The regexp grammar, which sees the decoded regexp literal * (after lexical parsing) DOES have a \uHHHH unicode escape. * So, for instance: * * \u1234/ * * should first be parsed by the lexical grammar as: * * '\u' RegularExpressionBackslashSequence * '1' RegularExpressionNonTerminator * '2' RegularExpressionNonTerminator * '3' RegularExpressionNonTerminator * '4' RegularExpressionNonTerminator * * and the escape itself is then parsed by the regexp engine. * This is the current implementation. * * Minor spec inconsistency: * * E5 Section 7.8.5 RegularExpressionBackslashSequence is: * * '\ RegularExpressionNonTerminator * * while Section A.1 RegularExpressionBackslashSequence is: * * '\ NonTerminator * * The latter is not normative and a typo. *

42739. * Octal escape or zero escape: * \0 (lookahead not DecimalDigit) * \1 ... \7 (lookahead not DecimalDigit) * \ZeroToThree OctalDigit (lookahead not DecimalDigit) * \FourToSeven OctalDigit (no lookahead restrictions) * \ZeroToThree OctalDigit OctalDigit (no lookahead restrictions) * * Zero escape is part of the standard syntax. Octal escapes are * defined in E5 Section B.1.2, and are only allowed in non-strict mode. * Any other productions starting with a decimal digit are invalid.

42740. out_token->lineterm set by caller

42741. (?!

42742. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

42743. E5 Section 15.10.2.11
42744. Keep current size
42745. 0=base, 1=esc, 2=class, 3=class+esc
42746. **comment:** Throwing an error this deep makes the error rather vague, but * saves hundreds of bytes of code.
 label: code-design
42747. XXX: potential issue with signed pointers, p_end < p.
42748. lookup shorthands (note: assume context variable is named 'lex_ctx')
42749. **comment:** Zero 'count' is also allowed to make call sites easier. * Arithmetic in bytes generates better code in GCC.
 label: code-design
42750. numeric value of a hex digit (also covers octal and decimal digits)
42751. DecimalEscape, only \0 is allowed, no leading zeroes are allowed
42752. XXX: optimize by adding the token numbers directly into the * always interned duk_hstring objects (there should be enough * flag bits free for that)?
42753. ;'
42754. "/" and not in regexp mode
42755. * Lexing helpers
42756. EOF
42757. whether to use macros or helper function depends on call count
42758. **comment:** (advance << 8) + token_type, updated at function end, * init is unnecessary but suppresses "may be used uninitialized" warnings.
 label: code-design
42759. first, parse regexp body roughly
42760. Production allows 'DecimalDigits', including leading zeroes
42761. Does not allow e.g. 2**31-1, but one more would allow overflows of u32.
42762. '*'
42763. **comment:** * DecimalLiteral, HexIntegerLiteral, OctalIntegerLiteral * "pre-parsing", followed by an actual, accurate parser step. * * Note: the leading sign character ('+' or '-') is -not- part of * the production in E5 grammar, and that the a DecimalLiteral * starting with a '0' must be followed by a non-digit. Leading * zeroes are syntax errors and must be checked for. * * XXX: the two step parsing process is quite awkward, it would * be more straightforward to allow numconv to parse the longest * valid prefix (it already does that, it only needs to indicate * where the input ended). However, the lexer decodes characters * using a lookup window, so this is not a trivial change.
 label: code-design
42764. ']'
42765. **comment:** XXX: better coercion
 label: code-design
42766. '!'
42767. check final character validity
42768. * The escapes are same as outside a character class, except that \b has a * different meaning, and \B and backreferences are prohibited (see E5 * Section 15.10.2.19). However, it's difficult to share code because we * handle e.g. "\n" very differently: here we generate a single character * range for it.
42769. * Init lexer context
42770. Here 'x' is a Unicode codepoint
42771. How much to advance before next loop; note that next loop * will advance by 1 anyway, so -1 from the total escape * length (e.g. len("\uXXXX") - 1 = 6 - 1). As a default, * 1 is good.
42772. '>'
42773. Note: '\b' in char class is different than outside (assertion), * '\B' is not allowed and is caught by the duk_unicode_is_identifier_part() * check below.
42774. PatternCharacter, all excluded characters are matched by cases above
42775. skip opening slash on first loop
42776. Reuse buffer as is unless buffer has grown large.
42777. **comment:** DUK__ADVANCECHARS(lex_ctx, 2) would be correct here, but it unnecessary
 label: code-design
42778. Note: duk_uint8_t type yields larger code
42779. DUK_USE_LEXER_SLIDING_WINDOW
42780. '-' as a range indicator
42781. Two digit octal escape, digits validated. * * The if-condition is a bit tricky. We could catch e.g. *\039 in the three-digit escape and fail it there (by * validating the digits), but we want to avoid extra * additional validation code.
42782. not LS/PS
42783. '""'
42784. Slow path.
42785. No other escape beginning with a digit in strict mode
42786. invalid codepoint encoding or codepoint
42787. IdentityEscape, with dollar added as a valid additional * non-standard escape (see test-regexp-identity-escape-dollar.js). * Careful not to match end-of-buffer (<0) here.
42788. '&'
42789. **comment:** packed advance/token number macro used by multiple functions
 label: code-design
42790. fast paths for space and tab
42791. Although these could be parsed as PatternCharacters unambiguously (here), * E5 Section 15.10.1 grammar explicitly forbids these as PatternCharacters.
42792. clipped codepoint
42793. fall through to error
42794. * Shared exit path
42795. val2 = min count, val1 = max count
42796. ch is a literal character here or -1 if parsed entity was * an escape such as "\s".
42797. DUK__L0() cannot be a digit, because the loop doesn't terminate if it is
42798. line terminator will be handled on next round
42799. part of string
42800. **comment:** This would be nice, but parsing is faster without resetting the * value slots. The only side effect is that references to temporary * string values may linger until lexing is finished; they're then * freed normally.
 label: code-design
42801. '.'
42802. '^'
42803. * Append a Unicode codepoint to the temporary byte buffer. Performs * CESU-8 surrogate pair encoding for codepoints above the BMP. * Existing surrogate pairs are allowed and also encoded into CESU-8.
42804. eat backslash on entry
42805. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
42806. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
42807. **comment:** Period followed by a digit can only start DecimalLiteral * (handled in slow path). We could jump straight into the * DecimalLiteral handling but should avoid goto to inside * a block.
 label: code-design
42808. IdentityEscape

42809. could also just pop?
42810. ''
42811. adv = 2 - 1 default OK
42812. eat '{' on entry
42813. Zero 'count' is also allowed to make call sites easier.
42814. Note: 'e' and 'E' are also accepted here.
42815. Zero escape (also allowed in non-strict mode)
42816. **comment:** * Advance lookup window by N characters, filling in new characters as * necessary. After returning caller is guaranteed a character window of * at least DUK_LEXER_WINDOW_SIZE characters. ** The main function duk__advance_bytes() is called at least once per every * token so it has a major lexer/compiler performance impact. There are two * variants for the main duk__advance_bytes() algorithm: a sliding window * approach which is slightly faster at the cost of larger code footprint, * and a simple copying one. ** Decoding directly from the source string would be another lexing option. * But the lookup window based approach has the advantage of hiding the * source string and its encoding effectively which gives more flexibility * going forward to e.g. support chunked streaming of source from flash. ** Decodes UTF-8/CESU-8 leniently with support for code points from U+0000 to * U+10FFFF, causing an error if the input is unparseable. Leniency means: *** Unicode code point validation is intentionally not performed, * except to check that the codepoint does not exceed 0x10ffff. * * In particular, surrogate pairs are allowed and not combined, which * allows source files to represent all SourceCharacters with CESU-8. * Broken surrogate pairs are allowed, as Ecmascript does not mandate * their validation. *** Allow non-shortest UTF-8 encodings. ** Leniency here causes few security concerns because all character data is * decoded into Unicode codepoints before lexer processing, and is then * re-encoded into CESU-8. The source can be parsed as strict UTF-8 with * a compiler option. However, Ecmascript source characters include -all- * 16-bit unsigned integer codepoints, so leniency seems to be appropriate. * * Note that codepoints above the BMP are not strictly SourceCharacters, * but the lexer still accepts them as such. Before ending up in a string * or an identifier name, codepoints above BMP are converted into surrogate * pairs and then CESU-8 encoded, resulting in 16-bit Unicode data as * expected by Ecmascript. ** An alternative approach to dealing with invalid or partial sequences * would be to skip them and replace them with e.g. the Unicode replacement * character U+FFFD. This has limited utility because a replacement character * will most likely cause a parse error, unless it occurs inside a string. * Further, Ecmascript source is typically pure ASCII. ** See: ** <http://en.wikipedia.org/wiki=UTF-8> * <http://en.wikipedia.org/wiki/CESU-8> * <http://tools.ietf.org/html/fc3629> * http://en.wikipedia.org/wiki/UTF-8#Invalid_byte_sequences ** Future work: *** Reject other invalid Unicode sequences (see Wikipedia entry for examples) * in strict UTF-8 mode. *** Size optimize. An attempt to use a 16-byte lookup table for the first * byte resulted in a code increase though. *** Is checking against maximum 0x10ffff really useful? 4-byte encoding * imposes a certain limit anyway. *** Support chunked streaming of source code. Can be implemented either * by streaming chunks of bytes or chunks of codepoints.
label: code-design
42817. * E5 Section 7.4, allow SourceCharacter (which is any 16-bit * code point).
42818. **comment:** not necessary to init, disabled for faster parsing
label: code-design
42819. Section 7.8.3 (note): NumericLiteral must be followed by something other than * IdentifierStart or DecimalDigit.
42820. unreachable
42821. **comment:** never reached, but avoids warnings of * potentially unused variables.
label: code-design
42822. switch
42823. Track number of escapes; count not really needed but directive * prologues need to detect whether there were any escapes or line * continuations or not.
42824. free some memory
42825. **comment:** XXX: naming is inconsistent: ATOM_END_GROUP ends an ASSERT_START_LOOKAHEAD
label: code-design
42826. 'advtok' indicates how much to advance and which token id to assign * at the end. This shared functionality minimizes code size. All * code paths are required to set 'advtok' to some value, so no default * init value is used. Code paths calling DUK_ERROR() never return so * they don't need to set advtok.
42827. marker: copy t if not changed
42828. eat opening quote on first loop
42829. ''
42830. Note: intentionally allow leading zeroes here, as the * actual parser will check for them.
42831. input offset tracking
42832. Constants for duk_lexer_ctx.buf.
42833. token type (with reserved word identification)
42834. valstack slot for 2nd token value
42835. Lexer context. Same context is used for Ecmascript and Regexp parsing.
42836. identifier names (E5 Section 7.6)
42837. token was preceded by a lineterm
42838. token type (with reserved words as DUK_TOK_IDENTIFIER)
42839. number of tokens parsed
42840. Regexp tokens
42841. Sanity checks for string and token defines
42842. A regexp token value.
42843. number of escapes and line continuations (for directive prologue)
42844. * Lexer defines.
42845. Lexer codepoint with additional info like offset/line number
42846. **comment:** "get" and "set" are tokens but NOT ReservedWords. They are currently * parsed and identifiers and these defines are actually now unused.
label: code-design
42847. A token value. Can be memcpy()'d, but note that slot1/slot2 values are on the valstack. * Some fields (like num, str1, str2) are only valid for specific token types and may have * stale values otherwise.
42848. input linenum at input_offset (not window[0]), init to 1
42849. DUK_USE_REGEX_SUPPORT
42850. returned after EOF (infinite amount)
42851. unicode code points, window[0] is always next, points to 'buffer'
42852. bufwriter for temp accumulation
42853. input offset for window leading edge (not window[0])
42854. Convert heap string index to a token (reserved words)
42855. token allows automatic semicolon insertion (eof or preceded by newline)
42856. named "arithmetic" because result is signed
42857. start byte offset of token in lexer input
42858. A structure for 'snapshotting' a point for rewinding
42859. temp accumulation buffer
42860. * Prototypes
42861. reserved words: additional future reserved words in strict mode
42862. valstack slot for 1st token value
42863. numeric value of token
42864. exclusive
42865. reserved words: keywords
42866. string 1 of token (borrowed, stored to ctx->slot1_idx)
42867. reserved words: future reserved words
42868. input byte length
42869. unicode code points, window[0] is always next
42870. "null", "true", and "false" are always reserved words. * Note that "get" and "set" are not!

42871. DUK_LEXER_H_INCLUDED
42872. valstack slot for temp buffer
42873. currently 6 characters of lookup are actually needed (duk_lexer.c)
42874. token type
42875. numeric value (character, count)
42876. punctuators (unlike the spec, also includes "/" and "/=")
42877. inclusive
42878. maximum token count before error (sanity backstop)
42879. start line of token (first char)
42880. literals (E5 Section 7.8), except null, true, false, which are treated * like reserved words (above).
42881. input string (may be a user pointer)
42882. Sanity check
42883. * A token is interpreted as any possible production of InputElementDiv * and InputElementRegExp, see E5 Section 7 in its entirety. Note that * the E5 "Token" production does not cover all actual tokens of the * language (which is explicitly stated in the specification, Section 7.5). * Null and boolean literals are defined as part of both ReservedWord * (E5 Section 7.6.1) and Literal (E5 Section 7.8) productions. Here, * null and boolean values have literal tokens, and are not reserved * words. * * Decimal literal negative/positive sign is -not- part of DUK_TOK_NUMBER. * The number tokens always have a non-negative value. The unary minus * operator in "-1.0" is optimized during compilation to yield a single * negative constant. * * Token numbering is free except that reserved words are required to be * in a continuous range and in a particular order. See genstrings.py.
42884. string 2 of token (borrowed, stored to ctx->slot2_idx)
42885. thread; minimizes argument passing
42886. Pad significand with "virtual" zero digits so that Dragon4 will * have enough (apparent) precision to work with.
42887. need to normalize, may even cancel to 0
42888. Zero special case: fake requested number of zero digits; ensure * no sign bit is printed. Relative and absolute fixed format * require separate handling.
42889. This may happen even after the fast path check, if exponent is * not balanced (e.g. "0e1"). Remember to respect zero sign.
42890. * seeeeeeee eeeeefff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * ieee value = 1.ffff... * 2^(e - 1023) (normal) * = 0.ffff... * 2^(-1022) (denormal) * * algorithm v = f * b^e
42891. Note: 'count' is currently not adjusted by rounding (i.e. the * digits are not "chopped off". That shouldn't matter because * the digit position (absolute or relative) is passed on to the * convert-and-push function.
42892. (\geq e 0) AND (not ($=$ f (expt b (- p 1)))) * * be <- (expt b e) == b^e * r <- (* f be 2) == 2 * f * b^e [if b==2 -> f * b^(e+1)] * s <- 2 * m+ <- be == b^e * m- <- be == b^e * k < 0 * B <- B * low_ok <- round * high_ok <- round
42893. DUK_USE_64BIT_OPS
42894. zero handled by caller
42895. Don't check for Infinity unless the context allows it. * 'Infinity' is a valid integer literal in e.g. base-36: * * parseInt('Infinity', 36) * 1461559270678
42896. borrow literal Infinity from builtin string
42897. d <- (quotient (* r B) s) (in range 0...B-1)
42898. exponential notation forced
42899. mp <- b^(e+1)
42900. input radix
42901. **comment:** * The string conversion here incorporates all the necessary Ecmascript * semantics without attempting to be generic. nc_ctx->digits contains * nc_ctx->count digits (\geq 1), with the topmost digit's 'position' * indicated by nc_ctx->k as follows: * * digits="123" count=3 k=0 --> 0.123 * digits="123" count=3 k=1 --> 1.23 * digits="123" count=3 k=5 --> 12300 * digits="123" count=3 k=-1 --> 0.0123 * * Note that the identifier names used for format selection are different * in Burger-Dybvig paper and Ecmascript specification (quite confusingly * so, because e.g. 'k' has a totally different meaning in each). See * documentation for discussion. * * Ecmascript doesn't specify any specific behavior for format selection * (e.g. when to use exponent notation) for non-base-10 numbers. * * The bigint space in the context is reused for string output, as there * is more than enough space for that (>1kB at the moment), and we avoid * allocating even more stack.
label: code-design
42902. 0 or -1
42903. **comment:** XXX: this could be optimized; there is only one call site now though
label: code-design
42904. x <- y + z
42905. exponent digit
42906. **comment:** XXX: this could be optimized
label: code-design
42907. * Convert and push final string.
42908. mm <- b^e
42909. e.g. 12.3 -> digits="123" k=2 -> 1.23e1
42910. **comment:** This is essentially the 'scale' algorithm, with recursion removed. * Note that 'k' is either correct immediately, or will move in one * direction in the loop. There's no need to do the low/high checks * on every round (like the Scheme algorithm does). * * The scheme algorithm finds 'k' and updates 's' simultaneously, * while the logical algorithm finds 'k' with 's' having its initial * value, after which 's' is updated separately (see the Burger-Dybvig * paper, Section 3.1, steps 2 and 3). * * The case where m+ == m- (almost always) is optimized for, because * it reduces the bigint operations considerably and almost always * applies. The scale loop only needs to work with m+, so this works.
label: code-design
42911. **comment:** Currently about 7*152 = 1064 bytes. The space for these * duk__bigints is used also as a temporary buffer for generating * the final string. This is a bit awkward; a union would be * more correct.
label: code-design
42912. s <- 2
42913. (quotient-remainder (* r B) s) using a dummy subtraction loop
42914. * Scan number and setup for Dragon4. * * The fast path case is detected during setup: an integer which * can be converted without rounding, no net exponent. The fast * path could be implemented as a separate scan, but may not really * be worth it: the multiplications for building 'f' are not * expensive when 'f' is small. * * The significand ('f') must contain enough bits of (apparent) * accuracy, so that Dragon4 will generate enough binary output digits. * For decimal numbers, this means generating a 20-digit significand, * which should yield enough practical accuracy to parse IEEE doubles. * In fact, the Ecmascript specification explicitly allows an * implementation to treat digits beyond 20 as zeroes (and even * to round the 20th digit upwards). For non-decimal numbers, the * appropriate number of digits has been precomputed for comparable * accuracy. * * Digit counts: * * [dig_lzero] * | * .+-...-[dig_prec]---. * || | * 0000123.45678901234567890e+123456 * | | | | * `--+-`-----[dig_frac]-----`-+-` * || * [dig_whole] [dig_expt] * * dig_frac and dig_expt are -1 if not present * dig_lzero is only computed for whole number part * * Parsing state * * Parsing whole part dig_frac < 0 AND dig_expt < 0 * Parsing fraction part dig_frac >= 0 AND dig_expt < 0 * Parsing exponent part dig_expt >= 0 (dig_frac may be < 0 or >= 0) * * Note: in case we hit an implementation limit (like exponent range), * we should throw an error, NOT return NaN or Infinity. Even with * very large exponent (or significand) values the final result may be * finite, so NaN/Infinity would be incorrect.
42915. 12 to 21
42916. r <- (2 * f) * b^(e+1)
42917. 31 to 36
42918. This upper value has been experimentally determined; debug build will check * bigint size with assertions.
42919. **comment:** unused
label: code-design
42920. x <- y * z
42921. Leading zero is not counted towards precision digits; not * in the integer part, nor in the fraction part.
42922. expt==1023 -> bitstart=0 (leading 1); * expt==1024 -> bitstart=-1 (one left of leading 1), etc
42923. **comment:** m+ != m- (very rarely)

label: code-design

42924. * Dragon4 slow path digit generation.
42925. free-form
42926. interpret e.g. '0x' and '0xg' as a NaN (= parse error)
42927. Maximum number of digits generated.
42928. * Exposed number-to-string API * * Input: [number] * Output: [string]
42929. http://en.wikipedia.org/wiki/Exponentiation_by_squaring
42930. empty ("") is allowed in some formats (e.g. Number(""), as zero
42931. r <- (remainder (* r B) s)
42932. if 1, doing a fixed format output (not free format)
42933. tc1 = true, tc2 = false
42934. Leading zero.
42935. m+ <- (* m+ B)
42936. toString() conditions
42937. Digit generation
42938. significant from precision perspective
42939. Loop structure ensures that we don't compute $t1^2$ unnecessarily * on the final round, as that might create a bignum exceeding the * current DUK_BL_MAX_PARTS limit.
42940. Most common cases first.
42941. Ignore digits beyond a radix-specific limit, but note them * in expt_adj.
42942. Return value: $<0 \iff x < y \iff 0 \iff x == y \iff >0 \iff x > y$
42943. no net exponent
42944. needed for inf: causes mantissa to become zero, * and rounding to be skipped.
42945. carry
42946. add E
42947. A 32-bit unsigned integer formats to at most 32 digits (the * worst case happens with radix == 2). Output the digits backwards, * and use a memmove() to get them in the right place.
42948. Fast path the binary case
42949. Infinity
42950. low to high
42951. x <- (1<<y)
42952. absolute position for digit considered for rounding
42953. r <- (* r B) * s <- s * m+ <- (* m+ B) * m- <- (* m- B) * k <- (-k 1)
42954. r <- (* f 2) * s <- (* (expt b (- e)) 2) == b^(e) * 2 [if b==2 -> b^(1-e)] * m+ <- 1 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round
42955. tc1 = false, tc2 = false
42956. r <- r * s <- (* s B) * m+ <- m+ * m- <- m- * k <- (+ k 1)
42957. + 1 for rounding
42958. t1 <- (* r B)
42959. Both inputs are zero; cases where only one is zero can go * through main algorithm.
42960. Exponent without a sign or with a +/- sign is accepted * by all call sites (even JSON.parse()).
42961. * Convert binary digits into an IEEE double. Need to handle * denormals and rounding correctly.
42962. allow e.g. '0x0009' and '00077'
42963. tc1 = false, tc2 = true
42964. add AC*2^32
42965. large context; around 2kB now
42966. mp <- b^e
42967. s <- b^(e) * 2
42968. Would be nice to bulk clear the allocation, but the context * is 1-2 kilobytes and nothing should rely on it being zeroed.
42969. x <- x + y, use t as temp
42970. setup many variables in nc_ctx
42971. add BC*2^16
42972. Note: if 'x' is zero, x->n becomes 0 here
42973. **comment:** * Dragon4 setup. * * Convert double from IEEE representation for conversion; * normal finite values have an implicit leading 1-bit. The * slow path algorithm doesn't handle zero, so zero is special * cased here but still creates a valid nc_ctx, and goes * through normal formatting in case special formatting has * been requested (e.g. forced exponential format: 0 -> "0e+0").
label: code-design
42974. * Multiply + add + carry for 32-bit components using only 16x16->32 * multiplies and carry detection based on unsigned overflow. * * 1st mult, 32-bit: (A*2^16 + B) * 2nd mult, 32-bit: (C*2^16 + D) * 3rd add, 32-bit: E * 4th add, 32-bit: F * * (AC*2^16 + B) * (C*2^16 + D) + E + F * = AC*2^32 + AD*2^16 + BC*2^16 + BD + E + F * = AC*2^32 + (AD + BC)*2^16 + (BD + E + F)
42975. (-Number.MAX_VALUE).toString(2).length == 1025, + spare
42976. position of highest digit changed
42977. * Round-up limit. * * For even values, divides evenly, e.g. 10 -> roundup_limit=5. * * For odd values, rounds up, e.g. 3 -> roundup_limit=2. * If radix is 3, 0/3 -> down, 1/3 -> down, 2/3 -> up.
42978. Note: computes with 'idx' in assertions, so caller beware. * 'idx' is preincremented, i.e. '1' on first call, because it * is more convenient for the caller.
42979. * Digit generation loop. * * Different termination conditions: * * 1. Free format output. Terminate when shortest accurate * representation found. * * 2. Fixed format output, with specific number of digits. * Ignore termination conditions, terminate when digits * generated. Caller requests an extra digit and rounds. * * 3. Fixed format output, with a specific absolute cut-off * position (e.g. 10 digits after decimal point). Note * that we always generate at least one digit, even if * the digit is below the cut-off point already.
42980. number of digits changed
42981. **comment:** No NUL term checks in this debug code.
label: code-design
42982. triggers garbage digit check below
42983. digit count
42984. impose a reasonable exponent limit, so that exp * doesn't need to get tracked using a bigint.
42985. interpret e.g. '09' as '0', not NaN
42986.toFixed()
42987. * Tables generated with src/gennumdigits.py. * * duk_str2num_digits_for_radix indicates, for each radix, how many input * digits should be considered significant for string-to-number conversion. * The input is also padded to this many digits to give the Dragon4 * conversion enough (apparent) precision to work with. * * duk_str2num_exp_limits indicates, for each radix, the radix-specific * minimum/maximum exponent values (for a Dragon4 integer mantissa) * below and above which the number is guaranteed to underflow to zero * or overflow to Infinity. This allows parsing to keep bigint values * bounded.
42988. Detect zero special case.
42989. Number and (minimum) size of bigints in the nc_ctx structure.
42990. Careful with borrow condition: * - If borrow not subtracted: 0x12345678 - 0 - 0xffffffff = 0x12345679 (> 0x12345678) * - If borrow subtracted: 0x12345678 - 1 - 0xffffffff = 0x12345678 (== 0x12345678)
42991. **comment:** Note: not honoring round-to-even should work but now generates incorrect * results. For instance, 1e23 serializes to "a000...", i.e. the first digit * equals the radix (10). Scaling stops one step too early in this case. * Don't know why this is the case, but since this code path is unused, it * doesn't matter.
label: code-design
42992. r <- (2 * f) * b^e

42993. -Infinity

42994. * Exponent notation for non-base-10 numbers isn't specified in Ecmascript * specification, as it never explicitly turns up: non-decimal numbers can * only be formatted with Number.prototype.toString([radix]) and for that, * behavior is not explicitly specified. * * Logical choices include formatting the exponent as decimal (e.g. binary * 100000 as 1e+5) or in current radix (e.g. binary 100000 as 1e+101). * The Dragon4 algorithm (in the original paper) prints the exponent value * in the target radix B. However, for radix values 15 and above, the * exponent separator 'e' is no longer easily parseable. Consider, for * instance, the number "1.faecee+1c".

42995. Start position (inclusive) and end position (exclusive)

42996. * Preliminaries: trim, sign, Infinity check * * We rely on the interned string having a NUL terminator, which will * cause a parse failure wherever it is encountered. As a result, we * don't need separate pointer checks. * * There is no special parsing for 'NaN' in the specification although * 'Infinity' (with an optional sign) is allowed in some contexts. * Some contexts allow plus/minus sign, while others only allow the * minus sign (like JSON.parse()). * * Automatic hex number detection (leading '0x' or '0X') and octal * number detection (leading '0' followed by at least one octal digit) * is done here too.

42997. denormal

42998. max possible

42999. fixed-format

43000. Count is incremented before DUK__DRAGON4_OUTPUT_PREINC() call * on purpose, which is taken into account by the macro.

43001. Sometimes this assert is not true right now; it will be true after * rounding. See: test-bug-numconv-mantissa-assert.js.

43002. IEEE exp without bias

43003. Exponent handling: if exponent format is used, record exponent value and * fake k such that one leading digit is generated (e.g. digits=123 -> "1.23"). * * toFixed() prevents exponent use; otherwise apply a set of criteria to * match the other API calls (toString(), toPrecision, etc).

43004. low 32 bits is complete

43005. Validity checks for various fraction formats ("0.1", ".1", "1.", ".").

43006. generate mantissa with a single leading whole number digit

43007. requested number of output digits; 0 = free-format

43008. s <- 2 * b

43009. ($\geq e 0$) AND ($= f (\text{expt } b (- p 1))$) * * be <- ($\text{expt } b e$) == $b^e * be1 <- (* be b) == (\text{expt } b (+ e 1)) == b^{(e+1)} * r <- (* f be1 2) == 2 * f * b^{(e+1)}$ [if $b==2 \rightarrow f * b^{(e+2)}$] * s <- (* b 2) [if $b==2 \rightarrow 4$] * m+ <- be1 == $b^{(e+1)} * m- <- be == b^e * k <- 0 * B <- B * \text{low_ok} <- \text{round} * \text{high_ok} <- \text{round}$

43010. Perform fixed-format rounding.

43011. Overestimate required size; debug code so not critical to be tight.

43012. see algorithm

43013. t1 <- (+ r m+)

43014. denormal or zero

43015. **comment:** no special formatting

label: code-design

43016. Fast path check.

43017. caller handles sign change

43018. r <- (* f b 2) [if $b==2 \rightarrow (* f 4)$] * s <- (* (expt b (- 1 e)) 2) == $b^{(1-e)} * 2$ [if $b==2 \rightarrow b^{(2-e)}$] * m+ <- b == 2 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round

43019. 22 to 31

43020. **comment:** * Handle integers in 32-bit range (that is, $[-(2^{32}-1), 2^{32}-1]$) * specially, as they're very likely for embedded programs. This * is now done for all radix values. We must be careful not to use * the fast path when special formatting (e.g. forced exponential) * is in force. * * XXX: could save space by supporting radix 10 only and using * sprintf "%lu" for the fast path and for exponent formatting.

label: code-design

43021. x <- x - y, use t as temp

43022. Carry is detected based on wrapping which relies on exact 32-bit * types.

43023. current digit

43024. fixed precision and zero padding would be required

43025. t has high mantissa

43026. max size: radix=2 + sign

43027. We can't shortcut zero here if it goes through special formatting * (such as forced exponential notation).

43028. "." is not accepted in any format

43029. Maximum number of characters in formatted value.

43030. Terminating conditions. For fixed width output, we just ignore the * terminating conditions (and pretend that tc1 == tc2 == false). The * the current shortcut for fixed-format output is to generate a few * extra digits and use rounding (with carry) to finish the output.

43031. **comment:** Borrow is detected based on wrapping which relies on exact 32-bit * types.

label: code-design

43032. fixed-format termination conditions

43033. * Limited functionality bigint implementation. * * Restricted to non-negative numbers with less than 32 * DUK__BI_MAX_PARTS bits, * with the caller responsible for ensuring this is never exceeded. No memory * allocation (except stack) is needed for bigint computation. Operations * have been tailored for number conversion needs. * * Argument order is "assignment order", i.e. target first, then arguments: * x <- y * z --> duk__bi_mul(x, y, z);

43034. **comment:** * Handle special cases (NaN, infinity, zero).

label: code-design

43035. exponent for 'f'

43036. 32-bit value

43037. Buffer used for generated digits, values are in the range [0,B-1].

43038. absolute req_digits; e.g. digits = 1 -> last digit is 0, * but add an extra digit for rounding.

43039. fall through and continue for-loop

43040. * Create mantissa

43041. 0.000...

43042. r <- r (updated above: r <- (remainder (* r B) s) * s <- s * m+ <- m+ (updated above: m+ <- (* m+ B) * m- <- m- (updated above: m- <- (* m- B) * B, low_ok, high_ok are fixed

43043. skip leading digit

43044. count is already incremented, take into account

43045. t2 = (* (+ r m+) B)

43046. exponent non-negative (and thus not minimum exponent)

43047. true, because v[1] has at least one bit set

43048. **comment:** This seems to waste a lot of stack frame entries, but good compilers * will compute these as needed below. Some of these initial flags are * also modified in the code below, so they can't all be removed.

label: code-design

43049. s <- b^(1-e) * 2

43050. * Number-to-string and string-to-number conversions. * * Slow path number-to-string and string-to-number conversion is based on * a Dragon4 variant, with fast paths for small integers. Big integer * arithmetic is needed for guaranteeing that the conversion is correct * and uses a minimum number of digits. The big number arithmetic has a * fixed maximum size and does not require dynamic allocations. * * See: doc/number-conversion.rst.

43051. is valid size

43052. integer number in range

43053. no need to normalize

43054. Current size is about 152 bytes.

43055. build result as: (r << 32) + s: start with (BD + E + F)

43056. x <- b^y; use t1 and t2 as temps

43057. $k > 0 \rightarrow k$ was too low, and cannot be too high
43058. * Exposed string-to-number API ** Input: [string] * Output: [number] ** If number parsing fails, a NaN is pushed as the result. If number parsing * fails due to an internal error, an InternalError is thrown.
43059. $x <- y - z$, require $x \geq y \Rightarrow z \geq 0$, i.e. $y \geq z$
43060. $m \leftarrow (* m - B)$
43061. Exponent
43062. $x <- y$
43063. terminating conditions
43064. $t1 = (+ r m +)$
43065. if 1, doing a string-to-number; else doing a number-to-string
43066. (e.g. $k=3$, digits=2 \rightarrow "12X")
43067. For string-to-number, pretend we never have the lowest mantissa as there * is no natural "precision" for inputs. Having lowest_mantissa == 0, we'll * fall into the base cases for both $e \geq 0$ and $e < 0$.
43068. whole or fraction digit
43069. * A Dragon4 number-to-string variant, based on: ** Guy L. Steele Jr., Jon L. White: "How to Print Floating-Point Numbers * Accurately" ** Robert G. Burger, R. Kent Dybvig: "Printing Floating-Point Numbers * Quickly and Accurately" ** The current algorithm is based on Figure 1 of the Burger-Dybvig paper, * i.e. the base implementation without logarithm estimation speedups * (these would increase code footprint considerably). Fixed-format output * does not follow the suggestions in the paper; instead, we generate an * extra digit and round-with-carry. ** The same algorithm is used for number parsing (with $b=10$ and $B=2$) * by generating one extra digit and doing rounding manually. * See doc/number-conversion.rst for limitations.
43070. essentially tracks digit position of lowest 'f' digit
43071. Some contexts don't allow fractions at all; this can't be a * post-check because the state ('f' and expt) would be incorrect.
43072. $r \leftarrow 2 * f$
43073. lead zero + 'digits' fractions + 1 for rounding
43074. A leading digit is not required in some cases, e.g. accept ".123". * In other cases (JSON.parse()) a leading digit is required. This * is checked for after the loop.
43075. Note: we don't parse back exponent notation for anything else * than radix 10, so this is not an ambiguous check (e.g. hex * exponent values may have 'e' either as a significand digit * or as an exponent separator). ** If the exponent separator occurs twice, 'e' will be interpreted * as a digit (= 14) and will be rejected as an invalid decimal * digit.
43076. "123.456"
43077. for
43078. Bigint is 2^{52} . Used to detect normalized IEEE double mantissa values * which are at the lowest edge (next floating point value downwards has * a different exponent). The lowest mantissa has the form: ** 1000.....000 (52 zeroes; only "hidden bit" is set)
43079. distinct bignums, easy mistake to make
43080. \rightarrow sets 'f' and 'e'
43081. **comment:** Leading / trailing whitespace is sometimes accepted and * sometimes not. After white space trimming, all valid input * characters are pure ASCII.
 label: code-design
43082. NB: use 's' as temp on purpose
43083. **comment:** XXX: could accept numbers larger than 32 bits, e.g. up to 53 bits?
 label: code-design
43084. Significand ('f') padding.
43085. **comment:** XXX: join these ops (multiply-accumulate), but only if * code footprint decreases.
 label: code-design
43086. Exponent without digits (e.g. "1e" or "1e+"). If trailing garbage is * allowed, ignore exponent part as garbage (= parse as "1", i.e. exp 0).
43087. don't increase 'count'
43088. this is the case for normalized numbers
43089. arbitrary marker, outside valid exp range
43090. **comment:** XXX: union would be more correct
 label: code-design
43091. NaN sign bit is platform specific with unpacked, un-normalized NaNs
43092. * Conversion helpers
43093. add AD* 2^{16}
43094. Fast path is triggered for no exponent and also for balanced exponent * and fraction parts, e.g. for "1.23e2" == "123". Remember to respect * zero sign.
43095. normal
43096. use bigint area as a temp
43097. "123"
43098. is normalized
43099. lowest mantissa for this exponent
43100. $mm \leftarrow mp$
43101. Round up digits to a given position. If position is out-of-bounds, * does nothing. If carry propagates over the first digit, a '1' is * prepended to digits and 'k' will be updated. Return value indicates * whether carry propagated over the first digit. ** Note that nc_ctx->count is NOT updated based on the rounding position * (it is updated only if carry overflows over the first digit and an * extra digit is prepended).
43102. * Figure out how generated digits match up with the mantissa, * and then perform rounding. If mantissa overflows, need to * recompute the exponent (it is bumped and may overflow to * infinity). ** For normal numbers the leading '1' is hidden and ignored, * and the last bit is used for rounding; ** rounding pt * <-----52----->| * 1 x x x x ... x x x |y ==> x x x x ... x x x x * * For denormals, the leading '1' is included in the number, * and the rounding point is different: ** rounding pt * <--52 or less--->| * 1 x x x x ... x x |x y ==> 0 0 ... 1 x x ... x x * * The largest denormals will have a mantissa beginning with * a '1' (the explicit leading bit); smaller denormals will * have leading zero bits. ** If the exponent would become too high, the result becomes * Infinity. If the exponent is so small that the entire * mantissa becomes zero, the result becomes zero. ** Note: the Dragon4 'k' is off-by-one with respect to the IEEE * exponent. For instance, $k==0$ indicates that the leading '1' * digit is at the first binary fraction position (0.1xxx...); * the corresponding IEEE exponent would be -1.
43103. ".123"
43104. no negative sign for zero
43105. (Number.MAX_VALUE).toString(2).length == 1024, + spare
43106. normal: implicit leading 1-bit
43107. **comment:** Should not be required because the code below always sets both high * and low parts, but at least gcc-4.4.5 fails to deduce this correctly * (perhaps because the low part is set (seemingly) conditionally in a * loop), so this is here to avoid the bogus warning.
 label: code-design
43108. 'roundpos' is relative to nc_ctx->k and increases to the right * (opposite of how 'k' changes).
43109. $r \leftarrow (2 * b) * f$
43110. digit position is absolute, not relative
43111. $x \leftarrow x * y$, use t as temp
43112. $t1 \leftarrow t1 - s$
43113. Assume IEEE round-to-even, so that shorter encoding can be used * when round-to-even would produce correct result. By removing * this check (and having low_ok == high_ok == 0) the results would * still be accurate but in some cases longer than necessary.
43114. When doing string-to-number, lowest_mantissa is always 0 so * the exponent check, while incorrect, won't matter.
43115. $(\leftarrow (* r 2) s)$
43116. not minimum exponent
43117. add F
43118. output radix
43119. **comment:** Quick reject of too large or too small exponents. This check * would be incorrect for zero (e.g. "0e1000" is zero, not Infinity) * so zero check must be above.

label: code-design

43120. * Dragon4 slow path (binary) digit generation. * An extra digit is generated for rounding.

43121. biased exp == 0 -> denormal, exp -1022

43122. $37 \times 32 = 1184$ bits43123. **comment:** slow init, do only for slow path cases**label:** code-design

43124. "123." is allowed in some formats

43125. **comment:** Corner case: see test-numconv-parse-mant-carry.js. We could * just bump the exponent and update bitstart, but it's more robust * to recompute (but avoid rounding twice).**label:** code-design

43126. tc1 = true, tc2 = true

43127. * round_idx points to the digit which is considered for rounding; the * digit to its left is the final digit of the rounded value. If round_idx * is zero, rounding will be performed; the result will either be an empty * rounded value or if carry happens a '1' digit is generated.

43128. 2 to 11

43129. $x <- y - z$ 43130. **comment:** XXX: this algorithm could be optimized quite a lot by using e.g. * a logarithm based estimator for 'k' and performing B^n multiplication * using a lookup table or using some bit-representation based exp * algorithm. Currently we just loop, with significant performance * impact for very large and very small numbers.**label:** code-design43131. Careful with carry condition: * - If carry not added: $0x12345678 + 0 + 0xffffffff = 0x12345677 (< 0x12345678)$ * - If carry added: $0x12345678 + 1 + 0xffffffff = 0x12345678 (= 0x12345678)$

43132. Force exponential format. Used for toExponential().

43133. Allow naked fraction (e.g. ".123")

43134. Digit count indicates number of fractions (i.e. an absolute * digit index instead of a relative one). Used together with * DUK_N2S_FLAG_FIXED_FORMAT for toFixed().

43135. Allow leading zeroes (e.g. "0123" -> "123")

43136. Maximum exponent value when parsing numbers. This is not strictly * compliant as there should be no upper limit, but as we parse the * exponent without a bigint, impose some limit.

43137. * String-to-number conversion

43138. Output a specified number of digits instead of using the shortest * form. Used for toPrecision() and toFixed().

43139. Allow trailing garbage (e.g. treat "123foo" as "123")

43140. * Prototypes

43141. Allow exponent

43142. DUK_NUMCONV_H_INCLUDED

43143. Allow empty string to be interpreted as 0

43144. **comment:** If number would need zero padding (for whole number part), use * exponential format instead. E.g. if input number is 12300, 3 * digits are generated ("123"), output "1.23e+4" instead of "12300". * Used for toPrecision().**label:** code-design

43145. Allow 'Infinity'

43146. * Number-to-string conversion. The semantics of these is very tightly * bound with the EcmaScript semantics required for call sites.

43147. Allow automatic detection of hex base ("0x" or "0X" prefix), * overrides radix argument and forces integer mode.

43148. Allow leading plus sign

43149. Allow leading minus sign

43150. Allow fraction part

43151. Trim white space (= allow leading and trailing whitespace)

43152. Allow empty fraction (e.g. "123.")

43153. Allow automatic detection of octal base, overrides radix * argument and forces integer mode.

43154. regexp opcodes

43155. maximum bytecode copies for {n,m} quantifiers

43156. * Regular expression structs, constants, and bytecode defines.

43157. 1e8

43158. regexp execution limits

43159. flags

43160. DUK_REGEX_H_INCLUDED

43161. internal temporary value, used for char classes

43162. 1e9

43163. * Prototypes

43164. regexp compilation limits

43165. **comment:** hacky helper for String.prototype.split()**label:** code-design

43166. allocated from valstack (fixed buffer)

43167. highest capture number emitted so far (used as: ++captures)

43168. insert code for matching the remainder - infinite or finite

43169. expensive init, just want to fill window

43170. [...] regexp_object escaped_source]

43171. record previous atom info in case next token is a quantifier

43172. DUK_USE_REGEX_SUPPORT

43173. retval (sub-atom char length) unused, tainted as complex above

43174. return '(?:)'

43175. two encoding attempts suffices

43176. * The remaining matches are emitted as sequence of SPLITs and atom * copies; the SPLITs skip the remaining copies and match the sequel. * This sequence needs to be emitted starting from the last copy * because the SPLITs are variable length due to the variable length * skip offset. This causes a lot of memory copying now. * * Example structure (greedy, match maximum # atoms): ** SPLIT1 LSEQ * (atom) * SPLIT1 LSEQ ; <- the byte length of this instruction is needed * (atom) ; to encode the above SPLIT1 correctly * ... * LSEQ:

43177. -> [...] escaped_source bytecode]

43178. prefer direct execution

43179. flags always encodes to 1 byte

43180. **comment:** These are not needed to implement quantifier capture handling, * but might be needed at some point.**label:** requirement

43181. * duk_re_range_callback for generating character class ranges. ** When ignoreCase is false, the range is simply emitted as is. * We don't, for instance, eliminate duplicates or overlapping * ranges in a character class. ** When ignoreCase is true, the range needs to be normalized through * canonicalization. Unfortunately a canonicalized version of a * continuous range is not necessarily continuous (e.g. [x-{ }] is * continuous but [X-{ }] is not). The current algorithm creates the * canonicalized range(s) space efficiently at the cost of compile * time execution time (see doc/regexp.rst for discussion). ** Note that the ctx->nranges is a context-wide temporary value * (this is OK because there cannot be multiple character classes * being parsed simultaneously).

43182. * Complex atom * * The original code is used as a template, and removed at the end * (this differs from the handling of simple quantifiers). ** NOTE: there is no current solution for empty atoms in complex * quantifiers. This would need some sort of a 'progress' instruction. ** XXX: impose limit on maximum result size, i.e. atom_code_len * atom_copies?

43183. mark as complex

43184. **comment:** unused
label: code-design

43185. insert (DUK_REOP_WIPERANGE, start, count) in reverse order so the order ends up right

43186. reuse last emitted atom for remaining 'infinite' quantifier

43187. [... pattern flags escaped_source]

43188. [... pattern flags escaped_source bytecode]

43189. qmin and qmax will be 0 or 1

43190. bytecode start offset of the atom parsed in this loop * (allows quantifiers to copy the atom bytecode)

43191. [... regexp_object escaped_source bytecode]

43192. * Exposed regexp compilation primitive. * * Sets up a regexp compilation context, and calls duk_parse_disjunction() to do the * actual parsing. Handles generation of the compiled regexp header and the * "boilerplate" capture of the matching substring (save 0 and 1). Also does some * global level regexp checks after recursive compilation has finished. * * An escaped version of the regexp source, suitable for use as a RegExp instance * 'source' property (see E5 Section 15.10.3), is also left on the stack. * * Input stack: [pattern flags] * Output stack: [bytecode escaped_source] (both as strings)

43193. * Insert a jump offset at 'offset' to complete an instruction * (the jump offset is always the last component of an instruction). * The 'skip' argument must be computed relative to 'offset', * -without- taking into account the skip field being inserted. * * ... A B C ins X Y Z ... (ins may be a JUMP, SPLIT1/SPLIT2, etc) * => ... A B C ins SKIP X Y Z * * Computing the final (adjusted) skip value, which is relative to the * first byte of the next instruction, is a bit tricky because of the * variable length UTF-8 encoding. See doc/regexp.rst for discussion.

43194. * Encoding helpers * * Some of the typing is bytecode based, e.g. slice sizes are unsigned 32-bit * even though the buffer operations will use duk_size_t.

43195. insert range count

43196. * Create escaped RegExp source (E5 Section 15.10.3). * * The current approach is to special case the empty RegExp * (" -> '(?:')") and otherwise replace unescaped '/' characters * with '/' regardless of where they occur in the regexp. * * Note that normalization does not seem to be necessary for * RegExp literals (e.g. '/foo/') because to be acceptable as * a RegExp literal, the text between forward slashes must * already match the escaping requirements (e.g. must not contain * unescaped forward slashes or be empty). Escaping IS needed * for expressions like 'new RegExp("...","")' however. * Currently, we re-escape in either case. * * Also note that we process the source here in UTF-8 encoded * form. This is correct, because any non-ASCII characters are * passed through without change.

43197. prefer direct

43198. **comment:** * Check for invalid backreferences; note that it is NOT an error * to back-reference a capture group which has not yet been introduced * in the pattern (as in /1(foo)/; in fact, the backreference will * always match! It IS an error to back-reference a capture group * which will never be introduced in the pattern. Thus, we can check * for such references only after parsing is complete.

label: code-design

43199. * Init compilation context

43200. [... escaped_source bytecode]

43201. -1 if disjunction is complex, char length if simple

43202. insert ranges instruction, range count patched in later

43203. add a new pending split to the beginning of the entire disjunction

43204. never here

43205. wipe the capture range made by the atom (if any)

43206. negative -> complex atom

43207. -1 for opcode

43208. no change

43209. 'aint' result as complex

43210. * Helper macros

43211. * Disjunction struct: result of parsing a disjunction

43212. [... escape_source bytecode]

43213. [... pattern flags]

43214. -> [... flags escaped_source bytecode]

43215. **comment:** note: ctx-wide temporary
label: code-design

43216. * Create normalized 'source' property (E5 Section 15.10.3).

43217. * Compilation

43218. set by atom case clauses

43219. value of re_ctx->captures at start of atom

43220. remove the original 'template' atom

43221. * Flags parsing (see E5 Section 15.10.4.1).

43222. * Init lexer

43223. -> [... escaped_source]

43224. Special case: original qmin was zero so there is nothing * to repeat. Emit an atom copy but jump over it here.

43225. insert the required matches (qmin) by copying the atom

43226. expect_eof

43227. coerce to string

43228. * Range parsing is done with a special lexer function which calls * us for every range parsed. This is different from how rest of * the parsing works, but avoids a heavy, arbitrary size intermediate * value type to hold the ranges. * * Another complication is the handling of character ranges when * case insensitive matching is used (see docs for discussion). * The range handler callback given to the lexer takes care of this * as well. * * Note that duplicate ranges are not eliminated when parsing character * classes, so that canonicalization of * * [0-9a-fA-Fx-{}]* creates the result (note the duplicate ranges): * * [0-9A-FA-FX-Z-{}]* where [x-{}] is split as a result of canonicalization. The duplicate * ranges are not a semantics issue: they work correctly.

43229. +1 for opcode

43230. * Finalize stack

43231. always at least the header

43232. Unescaped '/' ANYWHERE in the regexp (in disjunction, * inside a character class, ...) => same escape works.

43233. * Emit compiled regexp header: flags, ncaptures * (insertion order inverted on purpose)

43234. Note: successive characters could be joined into string matches * but this is not trivial (consider e.g. '/xyz+/'; see docs for * more discussion).

43235. complete 'sub atom'

43236. * Create a RegExp instance (E5 Section 15.10.7). * * Note: the output stack left by duk_compile() is directly compatible * with the input here. * * Input stack: [escaped_source bytecode] (both as strings) * Output stack: [RegExp]

43237. XXX: the insert helpers should ensure that the bytecode result is not * larger than expected (or at least assert for it). Many things in the * bytecode, like skip offsets, won't work correctly if the bytecode is * larger than say 2G.

43238. char length of the atom parsed in this loop

43239. * Regexp compilation. * * See doc/regexp.rst for a discussion of the compilation approach and * current limitations. * * Regexp bytecode assumes jumps can be expressed with signed 32-bit * integers. Consequently the bytecode size must not exceed 0x7fffffffL. * The implementation casts duk_size_t (buffer size) to duk_(u)int32_t * in many places. Although this could be changed, the bytecode format * limit would still prevent regexps exceeding the signed 32-bit limit * from working. * * XXX: The implementation does not prevent bytecode from exceeding the * maximum supported size. This could be done by limiting the maximum * input string size (assuming an upper bound can be computed for number * of bytecode bytes emitted per input byte) or checking buffer maximum * size when emitting bytecode (slower).

43240. silence scan-build warning

43241. **comment:** * Canonicalize a range, generating result ranges as necessary. * Needs to exhaustively scan the entire range (at most 65536 * code points). If 'direct' is set, caller (lexer) has ensured * that the range is already canonicalization compatible (this * is used to avoid unnecessary canonicalization of built-in * ranges like \W, which are not affected by canonicalization). * * NOTE: here is one place where we don't want to support chars * outside the BMP, because the exhaustive search would be * massively larger.

label: code-design

43242. only advance if not tainted
43243. **comment:** XXX: solve into closed form (smaller code)
 label: code-design
43244. re_ctx->captures at the start of the atom parsed in this loop
43245. special helper for emitting u16 lists (used for character ranges for built-in char classes)
43246. mark as complex (capture handling)
43247. [...] regexp_object]
43248. negative -> no atom matched on previous round
43249. * Parse regexp Disjunction. Most of regexp compilation happens here. * Handles Disjunction, Alternative, and Term productions directly without * recursion.
 The only constructs requiring recursion are positive/negative * lookaheads, capturing parentheses, and non-capturing parentheses. * * The function determines whether the entire disjunction is a 'simple atom' * (see doc/regexp.rst discussion on 'simple quantifiers') and if so, * returns the atom character length which is needed by the caller to keep * track of its own atom character length. A disjunction with more than one * alternative is never considered a simple atom (although in some cases * that might be the case). * * Return value: simple atom character length or < 0 if not a simple atom. * Appends the bytecode for the disjunction matcher to the end of the temp * buffer. * * Regexp top level structure is: * * Disjunction = Term* * | Term* | Disjunction * * Term = Assertion * | Atom * | Atom Quantifier * * An empty Term sequence is a valid disjunction alternative (e.g. /|||c||/). * * Notes: * * * Tracking of the 'simple-ness' of the current atom vs. the entire * disjunction are separate matters. For instance, the disjunction * may be complex, but individual atoms may be simple. Furthermore, * simple quantifiers are used whenever possible, even if the * disjunction as a whole is complex. * * * The estimate of whether an atom is simple is conservative now, * and it would be possible to expand it. For instance, captures * cause the disjunction to be marked complex, even though captures * -can- be handled by simple quantifiers with some minor modifications. * * * Disjunction 'tainting' as 'complex' is handled at the end of the * main for loop collectively for atoms. Assertions, quantifiers, * and '| tokens need to taint the result manually if necessary. * Assertions cannot add to result char length, only atoms (and * quantifiers) can; currently quantifiers will taint the result * as complex though.
43250. TypeError if fails
43251. a complex (new) atom taints the result
43252. finish up pending jump and split for last alternative
43253. duplicate zeroing, expect for (possible) NULL inits
43254. patch in range count later
43255. patch pending jump and split
43256. * Args validation
43257. atom_char_length, atom_start_offset, atom_start_offset reflect the * atom matched on the previous loop. If a quantifier is encountered * on this loop, these are needed to handle the quantifier correctly. * new_atom_char_length etc are for the atom parsed on this round; * they're written to atom_char_length etc at the end of the round.
43258. regexp support disabled
43259. * Simple atom * * If atom_char_length is zero, we'll have unbounded execution time for e.g. * /()*/x/.exec('x'). We can't just skip the match because it might have some * side effects (for instance, if we allowed captures in simple atoms, the * capture needs to happen). The simple solution below is to force the * quantifier to match at most once, since the additional matches have no effect. * * With a simple atom there can be no capture groups, so no captures need * to be reset.
43260. offset is now target of the pending split (right after jump)
43261. re_ctx->captures at start and end of atom parsing. * Since 'captures' indicates highest capture number emitted * so far in a DUK_REOP_SAVE, the captures numbers saved by * the atom are:]start_captures,end_captures].
43262. add a new pending match jump for latest finished alternative
43263. [...] pattern flags escaped_source buffer]
43264. parse ranges until character class ends
43265. Call sites don't need the result length so it's not accumulated.
43266. XXX: return type should probably be duk_size_t, or explicit checks are needed for * maximum size.
43267. 'taint' result as complex -- this is conservative, * as lookaheads do not backtrack.
43268. prefer jump
43269. Number of characters that the atom matches (e.g. 3 for 'abc'), * -1 if atom is complex and number of matched characters either * varies or is not known.
43270. * The handling here is a bit tricky. If a previous '| has been processed, * we have a pending split1 and a pending jump (for a previous match). These * need to be back-patched carefully. See docs for a detailed example.
43271. Note: can be safely scanned as bytes (undecoded)
43272. **comment:** pre-check how many atom copies we're willing to make (atom_copies not needed below)
 label: requirement
43273. match fail
43274. * Helpers for dealing with the input string
43275. split2: prefer jump execution (not direct)
43276. DUK_USE_REGEXP_SUPPORT
43277. non-global regexp: lastIndex never updated on match
43278. split1: prefer direct execution (no jump)
43279. [...] re_obj input bc]
43280. must fit into duk_small_int_t
43281. note: caller 'sp' is intentionally not updated here
43282. * Byte-based matching would be possible for case-sensitive * matching but not for case-insensitive matching. So, we * match by decoding the input and bytecode character normally. * * Bytecode characters are assumed to be already canonicalized. * Input characters are canonicalized automatically by * duk_inp_get_cp() if necessary. * * There is no opcode for matching multiple characters. The * regexp compiler has trouble joining strings efficiently * during compilation. See doc/regexp.rst for more discussion.
43283. must have start and end
43284. * No match, E5 Section 15.10.6.2, step 9.a.i - 9.a.ii apply, regardless * of 'global' flag of the RegExp. In particular, if lastIndex is invalid * initially, it is reset to zero.
43285. * Regexp recursive matching function. * * Returns 'sp' on successful match (points to character after last matched one), * NULL otherwise. * * The C recursion depth limit check is only performed in this function, this * suffices because the function is present in all true recursion required by * regexp execution.
43286. **comment:** not really necessary
 label: code-design
43287. E5 Sections 15.10.2.8, 7.3
43288. match: keep wiped/resaved values
43289. no regexp instance should exist without a non-configurable bytecode property
43290. Note: not necessary to check p against re_ctx->input_end: * the memory access is checked by duk_inp_get_cp(), while * valid compiled regexps cannot write a saved[] entry * which points to outside the string.
43291. **comment:** integer, but may be +/- Infinite, +/- zero (not NaN, though)
 label: code-design
43292. Note: if encoding ends by hitting end of input, we don't check that * the encoding is valid, we just assume it is.
43293. advance by one character (code point) and one char_offset
43294. TypeError if wrong; class check, see E5 Section 15.10.6
43295. fall through
43296. Note: allow backtracking from p == ptr_end
43297. **comment:** * Note: * * - duk_match_regexp() is required not to longjmp() in ordinary "non-match" * conditions; a longjmp() will terminate the entire matching process. * * - Clearing saved[] is not necessary because backtracking does it * * - Backtracking also rewinds ctx.recursion back to zero, unless an * internal/limit error occurs (which causes a longjmp()) * * - If we supported anchored matches, we would break out here * unconditionally; however, ECMAScript regexps don't have anchored * matches. It might make sense to implement a fast bail-out if * the regexp begins with '^' and sp is not 0: currently we'll just * run through the entire input string, trivially failing the match * at every non-zero offset.

label: code-design

43298. never here
43299. * Regexp instance check, bytecode check, input coercion. ** See E5 Section 15.10.6.
43300. * Get starting character offset for match, and initialize 'sp' based on it. ** Note: lastIndex is non-configurable so it must be present (we check the * internal class of the object above, so we know it is). User code can set * its value to an arbitrary (garbage) value though; E5 requires that lastIndex * be coerced to a number before using. The code below works even if the * property is missing: the value will then be coerced to zero. ** Note: lastIndex may be outside Uint32 range even after ToInteger() coercion. * For instance, ToInteger(+Infinity) = +Infinity. We track the match offset * as an integer, but pre-check it to be inside the 32-bit range before the loop. * If not, the check in E5 Section 15.10.6.2, step 9.a applies.
43301. * Matching complete, create result array or return a 'null'. Update lastIndex * if necessary. See E5 Section 15.10.6.2. ** Because lastIndex is a character (not byte) offset, we need the character * length of the match which we conveniently get as a side effect of interning * the matching substring (0th index of result array). ** saved[0] start pointer (~ byte offset) of current match * saved[1] end pointer (~ byte offset) of current match (exclusive) * char_offset start character offset of current match (-> .index of result) * char_end_offset end character offset (computed below)
43302. Assumes that saved[0] and saved[1] are always * set by regexp bytecode (if not, char_end_offset * will be zero). Also assumes clen reflects the * correct char length.
43303. char offset in [0, h_input->clen] (both ends inclusive), checked before entry
43304. backref n -> saved indices [n*2, n*2+1]
43305. Note: ctx.steps is intentionally not reset, it applies to the entire unanchored match
43306. buffer is automatically zeroed
43307. [... res_obj re_obj input bc saved_buf]
43308. **comment:** Note: don't bail out early, we must read all the ranges from * bytecode. Another option is to skip them efficiently after * breaking out of here. Prefer smallest code.
- label:** code-design
43309. NB: 'length' property is automatically updated by the array setup loop
43310. * Regexp executor. ** Safety: the EcmaScript executor should prevent user from reading and * replacing regexp bytecode. Even so, the executor must validate all * memory accesses etc. When an invalid access is detected (e.g. a 'save' * opcode to invalid, unallocated index) it should fail with an internal * error but not cause a segmentation fault. ** Notes: * - Backtrack counts are limited to unsigned 32 bits but should * technically be duk_size_t for strings longer than 4G chars. * This also requires a regexp bytecode change.
43311. avoid calling at end of input, will DUK_ERROR (above check suffices to avoid this)
43312. [... re_obj input] -> [... re_obj input bc]
43313. idx is unsigned, < 0 check is not necessary
43314. * Helpers for UTF-8 handling ** For bytecode readers the duk_uint32_t and duk_int32_t types are correct * because they're used for more than just codepoints.
43315. [... res_obj]
43316. **comment:** XXX: lastIndex handling produces a lot of asm
- label:** code-design
43317. * Exposed matcher function which provides the semantics of RegExp.prototype.exec(). ** RegExp.prototype.test() has the same semantics as exec() but does not return the * result object (which contains the matching string and capture groups). Currently * there is no separate test() helper, so a temporary result object is created and * discarded if test() is needed. This is intentional, to save code space. ** Input stack: [... re_obj input] * Output stack: [... result]
43318. canonicalized by compiler
43319. not a wordchar
43320. Note: if atom were to contain e.g. captures, we would need to * re-match the atom to get correct captures. Simply quantifiers * do not allow captures in their atom now, so this is not an issue.
43321. -> [... re_obj input bc saved_buf lastIndex]
43322. * Note: * - Intentionally attempt (empty) match at char_offset == k_input->clen ** - Negative char_offsets have been eliminated and char_offset is duk_uint32_t * -> no need or use for a negative check
43323. fail: restore saves
43324. [... re_obj input bc saved_buf]
43325. ToInteger(lastIndex)
43326. **comment:** XXX: these last tricks are unnecessary if the function is made * a genuine native function.
- label:** code-design
43327. capture is 'undefined', always matches!
43328. **comment:** Wipe capture range and save old values for backtracking. ** XXX: this typically happens with a relatively small idx_count. * It might be useful to handle cases where the count is small * (say <= 8) by saving the values in stack instead. This would * reduce memory churn and improve performance, at the cost of a * slightly higher code footprint.
- label:** code-design
43329. **comment:** * Needs a save of multiple saved[] entries depending on what range * may be overwritten. Because the regexp parser does no such analysis, * we currently save the entire saved array here. Lookaheads are thus * a bit expensive. Note that the saved array is not needed for just * the lookahead sub-match, but for the matching of the entire sequel. ** The temporary save buffer is pushed on to the valstack to handle * errors correctly. Each lookahead causes a C recursion and pushes * more stuff on the value stack. If the C recursion limit is less * than the value stack spare, there is no need to check the stack. * We do so regardless, just in case.
- label:** code-design
43330. signed integer encoding needed to work with UTF-8
43331. Captures which are undefined have NULL pointers and are returned * as 'undefined'. The same is done when saved[] pointers are insane * (this should, of course, never happen in practice).
43332. * Byte matching for back-references would be OK in case- * sensitive matching. In case-insensitive matching we need * to canonicalize characters, so back-reference matching needs * to be done with codepoints instead. So, we just decode * everything normally here, too. ** Note: back-reference index which is 0 or higher than * NCapturingParens (= number of capturing parens in the * -entire- regexp) is a compile time error. However, a * backreference referring to a valid capture which has * not matched anything always succeeds! See E5 Section * 15.10.2.9, step 5, sub-step 3.
43333. Get a (possibly canonicalized) input character from current sp. The input * itself is never modified, and captures always record non-canonicalized * characters even in case-insensitive matching.
43334. force_global
43335. match: keep saves
43336. * Basic context initialization. ** Some init values are read from the bytecode header * whose format is (UTF-8 codepoints): * * uint flags * uint nsaved (even, 2n+2 where n = num captures)
43337. regexp support disabled
43338. Backtrack utf-8 input and return a (possibly canonicalized) input character.
43339. [... re_obj input bc saved_buf res_obj val]
43340. lastIndex must be ignored for non-global regexps, but get the * value for (theoretical) side effects. No side effects can * really occur, because lastIndex is a normal property and is * always non-configurable for RegExp instances.
43341. **comment:** XXX: Array size is known before and (2 * re_ctx.nsaved) but not taken * advantage of now. The array is not compacted either, as regexp match * objects are usually short lived.
- label:** code-design
43342. and even number
43343. read header
43344. * Match loop. ** Try matching at different offsets until match found or input exhausted.
43345. * E5 Section 15.10.2.6. The previous and current character * should -not- be canonicalized as they are now. However, * canonicalization does not affect the result of IsWordChar() * (which depends on Unicode characters never canonicalizing * into ASCII characters) so this does not matter.
43346. [... re_obj input bc saved_buf res_obj]
43347. **comment:** This variant is needed by String.prototype.split(); it needs to perform * global-style matching on a cloned RegExp which is potentially non-global.

label: code-design

43348. dummy so sp won't get updated
43349. utf-8 continuation bytes have the form 10xx xxxx
43350. regexp compiler should catch these
43351. global regexp: lastIndex updated on match
43352. expt 0xffff is infinite/nan
43353. expt 0x000 is zero/subnormal
43354. expt values [0x001,0x7fe] = normal
43355. * Replacements for missing platform functions. * * Unlike the originals, fpclassify() and signbit() replacements don't * work on any floating point types, only doubles. The C typing here * mimics the standard prototypes.

43356. !DUK_SINGLE_FILE

43357. DUK_REPLACEMENTS_H_INCLUDED

43358. * Byte order. Important to self check, because on some exotic platforms * there is no actual detection but rather assumption based on platform * defines.

43359. Test without volatiles

43360. >>> struct.unpack('>d', '4000112233445566'.decode('hex')) * (2.008366013071895,)

43361. * Zero sign, see misc/tcc_zerosign2.c.

43362. Some internal code now assumes that all duk_uint_t values * can be expressed with a duk_size_t.

43363. * Union aliasing, see misc/clang_aliasing.c.

43364. little endian

43365. Test signaling NaN and alias assignment in all endianness combinations.

43366. Catch a double-to-int64 cast issue encountered in practice.

43367. * Casting

43368. **comment:** * Struct size/alignment if platform requires it * * There are some compiler specific struct padding pragmas etc in use, this * selftest ensures they're correctly detected and used.

label: code-design

43369. * Self tests to ensure execution environment is sane. Intended to catch * compiler/platform problems which cannot be detected at compile time.

43370. * Unions and structs for self tests

43371. big endian

43372. nop

43373. * Packed tval sanity

43374. DUK_USE_SELF_TESTS

43375. * 64-bit arithmetic * * There are some platforms/compilers where 64-bit types are available * but don't work correctly. Test for known cases.

43376. **comment:** Same test with volatiles

label: test

43377. * Self test main

43378. * This test fails on an exotic ARM target; double-to-uint * cast is incorrectly clamped to -signed- int highest value. * *

<https://github.com/svaarala/duktape/issues/336>

43379. * Basic double / byte union memory layout.

43380. mixed endian

43381. Allow very small lenience because some compilers won't parse * exact IEEE double constants (happened in matrix testing with * Linux gcc-4.8 -m32 at least).

43382. * >>> struct.pack('>d', 102030405060).encode('hex') * '4237c17c6dc40000'

43383. Note that byte order doesn't affect this test: all bytes in * 'test' will be 0xFF for two's complement.

43384. Oxdeadbeef in decimal

43385. * Two's complement arithmetic.

43386. * DUK_BSswap macros

43387. * <https://github.com/svaarala/duktape/issues/127#issuecomment-77863473>

43388. no check

43389. This testcase fails when Emscripten-generated code runs on Firefox. * It's not an issue because the failure should only affect packed * duk_tval representation, which is not used with Emscripten.

43390. **comment:** * Various sanity checks for typing

label: code-design

43391. * Selftest code

43392. DUK_SELFTEST_H_INCLUDED

43393. Object property access

43394. Compiler

43395. Misc

43396. Limits

43397. Mostly API and built-in method related

43398. **comment:** still in use with verbose messages

label: code-design

43399. Regexp

43400. **comment:** * Shared error message strings * * To minimize code footprint, try to share error messages inside Duktape * code. Modern compilers will do this automatically anyway, this is mostly * for older compilers.

label: code-design

43401. JSON

43402. DUK_ERRMSG_H_INCLUDED

43403. **comment:** still in use with verbose messages

label: code-design

43404. !DUK_SINGLE_FILE

43405. **comment:** * Shared error messages: declarations and macros * * Error messages are accessed through macros with fine-grained, explicit * error message distinctions. Concrete error messages are selected by the * macros and multiple macros can map to the same concrete string to save * on code footprint. This allows flexible footprint/verbosity tuning with * minimal code impact. There are a few limitations to this approach: * (1) switching between plain messages and format strings doesn't work * conveniently, and (2) conditional strings are a bit awkward to handle. * * Because format strings behave differently in the call site (they need to * be followed by format arguments), they have a special prefix (DUK_STR_FMT_ * and duk_str_fmt_). * * On some compilers using explicit shared strings is preferable; on others * it may be better to use straight literals because the compiler will combine * them anyway, and such strings won't end up unnecessarily in a symbol table.

label: code-design

43406. DUK_USE_FASTINT

43407. zero

43408. exponent 1070

43409. **comment:** unused

label: code-design

43410. exponent 0

43411. positive

43412. * Manually optimized number-to-double conversion

43413. 0

43414. negative

43415. exponents 1023 to 1069

43416. Note: reject negative zero.

43417. **comment:** * Manually optimized double-to-fastint downgrade check. ** This check has a large impact on performance, especially for fastint * slow paths, so must be changed carefully. The code should probably be * optimized for the case where the result does not fit into a fastint, * to minimize the penalty for "slow path code" dealing with fractions etc. ** At least on one tested soft float ARM platform double-to-int64 coercion * is very slow (and sometimes produces incorrect results, see self tests). * This algorithm combines a fastint compatibility check and extracting the * integer value from an IEEE double for setting the tagged fastint. For * other platforms a more naive approach might be better. ** See doc/fastint.rst for details.

label: code-design

43418. **comment:** XXX: optimize for packed duk_tval directly?

label: code-design

43419. implicit leading one

43420. avoid unary minus on unsigned

43421. $1 \ll 52$

43422. DUK_USE_FASTINT && DUK_USE_PACKED_TVAL

43423. **comment:** Note: not initializing all bytes is normally not an issue: Duktape won't * read or use the uninitialized bytes so valgrind won't issue warnings. * In some special cases a harmless valgrind warning may be issued though. * For example, the DumpHeap debugger command writes out a compiled function's * 'data' area as is, including any uninitialized bytes, which causes a * valgrind warning.

label: code-design

43424. Sign extend: $0x0000\#\#00 \rightarrow 0x\#\#000000 \rightarrow$ sign extend to $0x\#\#\#\#\#\#$

43425. * Portable 12-byte representation

43426. XXX: clumsy sign extend and masking of 16 topmost bits

43427. marker; not actual tagged type

43428. marker; not actual tagged value

43429. **comment:** This is performance critical because it's needed for every DECREF. * Take advantage of the fact that the first heap allocated tag is 8, * so that bit 3 is set for all heap allocated tags (and never set for * non-heap-allocated tags).

label: code-design

43430. DUK_USE_64BIT_OPS

43431. Assumes that caller has normalized NaNs, otherwise trouble ahead.

43432. use duk_double_union as duk_tval directly

43433. This seems reasonable overall.

43434. **comment:** two casts to avoid gcc warning: "warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]"

label: code-design

43435. DUK_TAG_NUMBER is intentionally first, as it is the default clause in code * to support the 8-byte representation. Further, it is a non-heap-allocated * type so it should come before DUK_TAG_STRING. Finally, it should not break * the tag value ranges covered by case-clauses in a switch-case.

43436. for convenience

43437. **comment:** avoid tag 0xffff0, no risk of confusion with negative infinity

label: code-design

43438. embed: integer value

43439. DUK_TAG_NUMBER would logically go here, but it has multiple 'tags'

43440. **comment:** XXX: fast int-to-double

label: code-design

43441. max, excl. varargs marker

43442. DUK_USE_PACKED_TVAL

43443. if present, forces 16-byte duk_tval

43444. varargs marker

43445. Note: masking is done for 'i' to deal with negative numbers correctly

43446. * Packed 8-byte representation

43447. in non-packed representation we don't care about which NaN is used

43448. embed: void ptr

43449. decoding

43450. 0xffff0 is -Infinity

43451. embed: duk_hobject ptr

43452. DUK_USE_FASTINT

43453. getters

43454. **comment:** This is performance critical because it appears in every DECREF.

label: code-design

43455. **comment:** the NaN variant we use

label: code-design

43456. Double casting for pointer to avoid gcc warning (cast from pointer to integer of different size)

43457. DUK_TVAL_H_INCLUDED

43458. embed: func ptr

43459. sanity

43460. fastint constants etc

43461. **comment:** * Tagged type definition (duk_tval) and accessor macros. ** Access all fields through the accessor macros, as the representation * is quite tricky. ** There are two packed type alternatives: an 8-byte representation * based on an IEEE double (preferred for compactness), and a 12-byte * representation (portability). The latter is needed also in e.g. * 64-bit environments (it usually pads to 16 bytes per value). ** Selecting the tagged type format involves many trade-offs (memory * use, size and performance of generated code, portability, etc), * see doc/types.rst for a detailed discussion (especially of how the * IEEE double format is used to pack tagged values). ** NB: because macro arguments are often expressions, macros should * avoid evaluating their argument more than once.

label: code-design

43462. embed: duk_hstring ptr

43463. Lightfunc flags packing and unpacking.

43464. first heap allocated, match bit boundary

43465. embed: 0 or 1 (false or true)

43466. setters

43467. not exposed

43468. embed: duk_hbuffer ptr

43469. tags

43470. embed: nothing

43471. * Convenience (independent of representation)

43472. =====

43473. * Unicode helpers

43474. zero-width joiner

43475. * Automatically generated by extract_chars.py, do not edit!

43476. * ASCII character constants ** C character literals like 'x' have a platform specific value and do * not match ASCII (UTF-8) values on e.g. EBCDIC platforms. So, use * these (admittedly awkward) constants instead. These constants must * also have signed values to avoid unexpected coercions in comparisons. ** <http://en.wikipedia.org/wiki/ASCII>

43477. up to 36 bit codepoints

43478. * Unicode tables

43479. duk_unicode_support.c
43480. !DUK_SINGLE_FILE
43481. all codepoints up to U+10FFFF
43482. DUK_UNICODE_H_INCLUDED
43483. * Automatically generated by extract_caseconv.py, do not edit!
43484. * Useful Unicode codepoints * * Integer constants must be signed to avoid unexpected coercions * in comparisons.
43485. * Prototypes
43486. all codepoints up to U+FFFF
43487. * Extern
43488. zero-width non-joiner
43489. * UTF-8 / XUTF-8 / CESU-8 constants
43490. http://en.wikipedia.org/wiki/Replacement_character#Replacement_character
43491. * E5 Section 15.10.2.6 "IsWordChar" abstract operation. Assume * x < 0 for characters read outside the string.
43492. does not fit into an uchar
43493. 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [26 bits]
43494. * UTF-8 decoder which accepts longer than standard byte sequences. * This allows full 32-bit code points to be used.
43495. DUK_USE_REGEXP_SUPPORT
43496. curr char
43497. 0x20...0x2f
43498. * Unicode letter check.
43499. * Replace valstack top with case converted version.
43500. 0x60...0x6f
43501. 0xxx xxxx [7 bits]
43502. 0x40...0x4f
43503. exposed because lexer needs these too
43504. * XUTF-8 and CESU-8 encoding/decoding
43505. 1110 xxxx 10xx xxxx 10xx xxxx [16 bits]
43506. without explicit non-BMP support, assume non-BMP characters * are always accepted as identifier characters.
43507. 110x xxxx 10xx xxxx [11 bits]
43508. * Unicode codepoints above U+FFFF are encoded as surrogate * pairs here. This ensures that all CESU-8 codepoints are * 16-bit values as expected in EcmaScript.
The surrogate * pairs always get a 3-byte encoding (each) in CESU-8. * See: http://en.wikipedia.org/wiki/Surrogate_pair * * 20-bit codepoint, 10 bits (A and B)
per surrogate pair: * * x = 0b00000000 0000AAAA AAAAAABB BBBB BBBB * sp1 = 0b110110AA AAAAAAAA (0xd800 + ((x >> 10) & 0x3ff)) * sp2 =
0b110111BB BBBB BBBB (0xdc00 + (x & 0x3ff)) * * Encoded into CESU-8: * * sp1 -> 0b1101101 (0xe0 + ((sp1 >> 12) & 0x0f)) * -> 0b1010AAAA (0x80 +
((sp1 >> 6) & 0x3f)) * -> 0b10AAAAAA (0x80 + (sp1 & 0x3f)) * sp2 -> 0b1101101 (0xe0 + ((sp2 >> 12) & 0x0f)) * -> 0b1011BBBB (0x80 + ((sp2 >> 6) &
0x3f)) * -> 0b10BBBBBB (0x80 + (sp2 & 0x3f)) * * Note that 0x10000 must be subtracted first. The code below * avoids the sp1, sp2 temporaries which saves
around 20 bytes * of code.
43509. XXX: add 'language' argument when locale/language sensitive rule * support added.
43510. next does not exist or next is not a letter
43511. Encoded as surrogate pair, each encoding to 3 bytes for * 6 bytes total. Codepoints above U+10FFFF encode as 6 bytes * too, see duk_unicode_encode_cesu8().
43512. 11 bits
43513. 0x50...0x5f
43514. **comment:** Fast canonicalization lookup at the cost of 128kB footprint.
label: code-design
43515. DUK_USE_PREFER_SIZE
43516. fall through
43517. * "IdentifierPart" production check.
43518. fast path for ASCII
43519. prev exists and is not a letter
43520. 36 bits
43521. verified at beginning
43522. complex, multicharacter conversion
43523. DUK_USE_ASSERTIONS
43524. Note: masking of 'x' is not necessary because of * range check and shifting -> no bits overlapping * the marker should be set.
43525. * Various Unicode help functions for character classification predicates, * case conversion, decoding, etc.
43526. Compute (extended) utf-8 length without codepoint encoding validation, * used for string interning. * * NOTE: This algorithm is performance critical, more so *
than string hashing * in some cases. It is needed when interning a string and needs to scan * every byte of the string with no skipping. Having an ASCII fast path *
is useful if possible in the algorithm. The current algorithms were * chosen from several variants, based on x64 gcc -O2 testing. See: *
<https://github.com/svaaraala/duktape/pull/422> * * NOTE: must match src/dukutil.py:duk_unicode_unvalidated_utf8_length().
43527. Fall through to handle the rest.
43528. check pointer at end
43529. Full, aligned 4-byte reads.
43530. * "IdentifierStart" production check.
43531. **comment:** XXX: turkish / azeri, lowercase rules
label: code-design
43532. without explicit non-BMP support, assume non-BMP characters * are always accepted as letters.
43533. 0x00...0x0f
43534. **comment:** XXX: there are language sensitive rules for the ASCII range. * If/when language/locale support is implemented, they need to * be implemented here for
the fast path. There are no context * sensitive rules for ASCII range.
label: code-design
43535. NULL is allowed, no output
43536. invalidates h_buf pointer
43537. Decode helper. Return zero on error.
43538. * Unicode range matcher * * Matches a codepoint against a packed bitstream of character ranges. * Used for slow path Unicode matching.
43539. 31 bits
43540. * Canonicalize() abstract operation needed for canonicalization of individual * codepoints during regexp compilation and execution, see E5 Section 15.10.2.8. *
Note that codepoints are canonicalized one character at a time, so no context * specific rules can apply. Locale specific rules can apply, though.
43541. * Fast path tables
43542. * Regexp range tables
43543. 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [31 bits]
43544. Small variant; roughly 150 bytes smaller than the fast variant.
43545. cp == -1 (EOF) never matches and causes return value 0
43546. * Complex case conversion helper which decodes a bit-packed conversion * control stream generated by unicode/extract_caseconv.py. The conversion * is very
slow because it runs through the conversion data in a linear * fashion to save space (which is why ASCII characters have a special * fast path before arriving here).
* * The particular bit counts etc have been determined experimentally to * be small but still sufficient, and must match the Python script *
(src/extract_caseconv.py). * * The return value is the case converted codepoint or -1 if the conversion * results in multiple characters (this is useful for regexp
Canonicalization * operation). If 'buf' is not NULL, the result codepoint(s) are also * appended to the hbuffer. * * Context and locale specific rules must be
checked before consulting * this function.

43547. Encode to CESU-8; 'out' must have space for at least * DUK_UNICODE_MAX_CESU8_LENGTH bytes; codepoints above U+10FFFF * will encode to garbage but won't overwrite the output buffer.

43548. default: no change

43549. * Final sigma context specific rule. This is a rather tricky * rule and this handling is probably not 100% correct now. * The rule is not locale/language specific so it is supported.

43550. Must match src/extract_chars.py, generate_match_table3()

43551. * E5 Section 7.3 * * A LineTerminatorSequence essentially merges <CR> <LF> sequences * into a single line terminator. This must be handled by the caller.

43552. 1:1 or special conversions, but not locale/context specific: script generated rules

43553. 10xx xxxx -> invalid

43554. * Unicode letter is now taken to be the categories: * * Lu, Ll, Lt, Lm, Lo * * (Not sure if this is exactly correct.) * * The ASCII fast path consists of: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z']

43555. Capital sigma occurred at "end of word", lowercase to * U+03C2 = GREEK SMALL LETTER FINAL SIGMA. Otherwise * fall through and let the normal rules lowercase it to * U+03C3 = GREEK SMALL LETTER SIGMA.

43556. 0x10...0x1f

43557. Align 'p' to 4; the input data may have arbitrary alignment. * End of string check not needed because blen >= 16.

43558. This seems like a good overall approach. Fast path for ASCII in 4 byte * blocks.

43559. on first round, skip

43560. surrogate pairs get encoded here

43561. **comment:** XXX: could add a fast path to process chunks of input codepoints, * but relative benefit would be quite small.
label: code-design

43562. end marker

43563. used by e.g. duk_regex_executor.c, string built-ins

43564. prev char

43565. ASCII fast path

43566. 0x70...0x7f

43567. **comment:** * E5 Section 7.6: * * IdentifierPart: * IdentifierStart * UnicodeCombiningMark * UnicodeDigit * UnicodeConnectorPunctuation * <ZWNJ> [U+200C] * <ZWJ> [U+200D] * * IdentifierPart production has one multi-character production * as part of its IdentifierStart alternative. The '\ character * of an escape sequence is not matched here, see discussion in * duk_unicode_is_identifier_start(). * * To match non-ASCII characters (codepoints >= 0x80), a very slow * linear range-by-range scan is used. The codepoint is first compared * to the IdentifierStart ranges, and if it doesn't match, then to a * set consisting of code points in IdentifierPart but not in * IdentifierStart. This is done to keep the unicode range data small, * at the expense of speed. * * The ASCII fast path consists of: * * 0x0030 ... 0x0039 ['0' ... '9', UnicodeDigit] * 0x0041 ... 0x005a ['A' ... 'Z', IdentifierStart] * 0x0061 ... 0x007a ['a' ... 'z', IdentifierStart] * 0x0024 ['\$', IdentifierStart] * 0x005f ['_', IdentifierStart and * UnicodeConnectorPunctuation] * * UnicodeCombiningMark has no code points <= 0x7f. * * The matching code reuses the "identifier start" tables, and then * consults a separate range set for characters in "identifier part" * but not in "identifier start". These can be extracted with the * "src/extract_chars.py" script. * * UnicodeCombiningMark -> categories Mn, Mc * UnicodeDigit -> categories Nd * UnicodeConnectorPunctuation -> categories Pc
label: code-design

43568. **comment:** XXX: lithuanian, explicit dot rules
label: code-design

43569. **comment:** * Note: the description in E5 Section 15.10.2.6 has a typo, it * contains 'A' twice and lacks 'a'; the intent is [0-9a-zA-Z_].
label: documentation

43570. 0x30...0x3f

43571. **comment:** * E5 Section 7.6: * * IdentifierStart: * UnicodeLetter * \$ * _ * \ UnicodeEscapeSequence * * IdentifierStart production has one multi-character production: * * \ UnicodeEscapeSequence * * The '\' character is -not- matched by this function. Rather, the caller * should decode the escape and then call this function to check whether the * decoded character is acceptable (see discussion in E5 Section 7.6). * * The "UnicodeLetter" alternative of the production allows letters * from various Unicode categories. These can be extracted with the * "src/extract_chars.py" script. * * Because the result has hundreds of Unicode codepoint ranges, matching * for any values >= 0x80 are done using a very slow range-by-range scan * and a packed range format. * * The ASCII portion (codepoints 0x00 ... 0x7f) is fast-patched below because * it matters the most. The ASCII related ranges of IdentifierStart are: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z'] * 0x0024 ['\$'] * 0x005f ['_']
label: code-design

43572. 26 bits

43573. 64-bit OK because always >= 0

43574. **comment:** XXX: lithuanian not implemented
label: requirement

43575. [r1,r2] is the range

43576. ASCII (and EOF) fast path -- quick accept and reject

43577. Encode to extended UTF-8; 'out' must have space for at least * DUK_UNICODE_MAX_XUTF8_LENGTH bytes. Allows encoding of any * 32-bit (unsigned) codepoint.

43578. always >= 0

43579. number of continuation (non-initial) bytes in [0x80,0xbff]

43580. 1:1 conversion

43581. * "LineTerminator" production check.

43582. **comment:** 8-byte format could be: * 1111 1111 10xx xxxx [41 bits] * * However, this format would not have a zero bit following the * leading one bits and would not allow 0xFF to be used as an * "invalid xutf-8" marker for internal keys. Further, 8-byte * encodings (up to 41 bit code points) are not currently needed.
label: code-design

43583. multiple codepoint conversion or non-ASCII mapped to ASCII * --> leave as is.

43584. * E5 Section 7.2 specifies six characters specifically as * white space: * * 0009;<control>;Cc;0;S;;;;N;CHARACTER TABULATION;;;, * 000B; <control>;Cc;0;S;;;;N;LINE TABULATION;;;, * 000C;<control>;Cc;0;WS;;;;N;FORM FEED (FF);;, * 0020;SPACE;Zs;0;WS;;;;N;;;, * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;N;NON-BREAKING SPACE;;;, * FEFF;ZERO WIDTH NO-BREAK SPACE;Cf;0;BN;;;;N;BYTE ORDER MARK;;;, * It also specifies any Unicode category 'Zs' characters as white * space. These can be extracted with the "src/extract_chars.py" script. * Current result: * * RAW OUTPUT: * ===== * 0020;SPACE;Zs;0;WS;;;;N;,, * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;N;NON-BREAKING SPACE;;;, * 1680;OGHAM SPACE MARK;Zs;0;WS;;;;N;,, * 180E;MONGOLIAN VOWEL SEPARATOR;Zs;0;WS;;;;N;,, * 2000;EN QUAD;Zs;0;WS;2002;;;N;,, * 2001;EM QUAD;Zs;0;WS;2003;;;N;,, * 2002;EN SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 2003;EM SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 2004;THREE-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 2005;FOUR-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 2006;SIX-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 2007;FIGURE SPACE;Zs;0;WS;<noBreak> 0020;;;N;,, * 2008;PUNCTUATION SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 2009;THIN SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 200A;HAIR SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 202F;NARROW NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;N;,, * 205F;MEDIUM MATHEMATICAL SPACE;Zs;0;WS;<compat> 0020;;;N;,, * 3000;IDEOGRAPHIC SPACE;Zs;0;WS;<wide> 0020;;;N;,, * * RANGES: * ===== * 0x0020 * 0x00a0 * 0x1680 * 0x180e * 0x2000 ... 0x200a * 0x202f * 0x205f * 0x3000 * * A manual decoder (below) is probably most compact for this.

43585. end of input and last char has been processed

43586. 16 bits

43587. **comment:** DUK_USE_REGEX_CANON_WORKAROUND
label: code-design

43588. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx [21 bits]

43589. **comment:** XXX: turkish / azeri
label: code-design

43590. U+03A3 = GREEK CAPITAL LETTER SIGMA

43591. 1111 1110 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [36 bits]

43592. Ensure space for maximum multi-character result; estimate is overkill.
43593. [... input buffer]
43594. uppercase
43595. * Case conversion helper, with context/local sensitivity. * For proper case conversion, one needs to know the character * and the preceding and following characters, as well as * locale/language.
43596. range conversion with a "skip"
43597. next char
43598. 7 bits
43599. * "WhiteSpace" production check.
43600. **comment:** Flip highest bit of each byte which changes * the bit pattern 10xxxxxx into 00xxxxxx which * allows an easy bit mask test.
 label: test
43601. 21 bits
43602. **comment:** unused now, not needed until Turkish/Azeri
 label: requirement
43603. **comment:** context and locale specific rules which cannot currently be represented * in the caseconv bitstream: hardcoded rules in C
 label: code-design
43604. **comment:** Non-ASCII slow path (range-by-range linear comparison), very slow
 label: code-design
43605. **comment:** XXX: 'internal error' is a bit of a misnomer
 label: code-design
43606. 0: not IdentifierStart or IdentifierPart * 1: IdentifierStart and IdentifierPart * -1: IdentifierPart only
43607. * Automatically generated by extract_chars.py, do not edit!
43608. * Unicode support tables automatically generated during build.
43609. IdentifierStart production with Letter, ASCII, and non-BMP excluded
43610. duk_unicode_ids_m_let_noabmp[]
43611. duk_unicode_ids_m_let_noa[]
43612. duk_unicode_ids_noabmp[]
43613. IdentifierStart production with ASCII excluded
43614. duk_unicode_caseconv_lc[]
43615. IdentifierPart production with IdentifierStart and ASCII excluded
43616. duk_unicode_ids_noa[]
43617. IdentifierStart production with ASCII and non-BMP excluded
43618. IdentifierStart production with Letter and ASCII excluded
43619. IdentifierPart production with IdentifierStart, ASCII, and non-BMP excluded
43620. duk_unicode_idp_m_ids_noa[]
43621. * Case conversion tables generated using src/extract_caseconv.py.
43622. duk_unicode_idp_m_ids_noabmp[]
43623. * Automatically generated by extract_caseconv.py, do not edit!
43624. duk_unicode_caseconv_uc[]
43625. **comment:** * Unicode tables containing ranges of Unicode characters in a * packed format. These tables are used to match non-ASCII * characters of complex productions by resorting to a linear * range-by-range comparison. This is very slow, but is expected * to be very rare in practical Ecmascript source code, and thus * compactness is most important. * * The tables are matched using uni_range_match() and the format * is described in src/extract_chars.py.
 label: code-design
43626. !DUK_SINGLE_FILE
43627. Insert bytes in the middle of the buffer from an external buffer.
43628. **comment:** * Buffer writer (dynamic buffer only) * * Helper for writing to a dynamic buffer with a concept of a "spare" area * to reduce resizes. You can ensure there is enough space beforehand and * then write for a while without further checks, relying on a stable data * pointer. Spare handling is automatic so call sites only indicate how * much data they need right now. * * There are several ways to write using bufwriter. The best approach * depends mainly on how much performance matters over code footprint. * The key issues are (1) ensuring there is space and (2) keeping the * pointers consistent. Fast code should ensure space for multiple writes * with one ensure call. Fastest inner loop code can temporarily borrow * the 'p' pointer but must write it back eventually. * * Be careful to ensure all macro arguments (other than static pointers like * 'thr' and 'bw_ctx') are evaluated exactly once, using temporaries if * necessary (if that's not possible, there should be a note near the macro). * Buffer write arguments often contain arithmetic etc so this is * particularly important here.
 label: code-design
43629. * Utilities
43630. * Endian conversion
43631. Ensuring (reserving) space.
43632. Evaluates to (duk_uint8_t *) pointing to start of area.
43633. * Externs and prototypes
43634. NOTE: Multiple evaluation of 'ptr' in this macro.
43635. For now only needed by the debugger.
43636. Pointers may be NULL for a while when 'buf' size is zero and before any * ENSURE calls have been made. Once an ENSURE has been made, the pointers * are required to be non-NULL so that it's always valid to use memcpy() and * memmove(), even for zero size.
43637. **comment:** XXX: Migrate bufwriter and other read/write helpers to its own header?
 label: code-design
43638. 2/4 -> 1/16 = 6.25% spare
43639. Working with the pointer and current size.
43640. **comment:** * Raw write/read macros for big endian, unaligned basic values. * Caller ensures there's enough space. The macros update the pointer * argument automatically on resizes. The idiom seems a bit odd, but * leads to compact code.
 label: code-design
43641. DUK_UTIL_H_INCLUDED
43642. **comment:** XXX: Rework to use an always-inline function?
 label: code-design
43643. Insert a reserved area somewhere in the buffer; caller fills it. * Evaluates to a (duk_uint_t *) pointing to the start of the reserved * area for convenience.
43644. No difference between raw/ensure because the buffer shrinks.
43645. Initialization and finalization (compaction), converting to other types.
43646. Miscellaneous.
43647. **comment:** XXX: add temporary duk__p pointer here too; sharing
 label: code-design
43648. 'ptr' is evaluated both as LHS and RHS.
43649. Safe write calls which will ensure space first.
43650. must match genhashsizes.py
43651. * Bitstream decoder
43652. Append bytes from a slice already in the buffer.
43653. Note: assumes that duk_util_probe_steps size is 32
43654. * Bitstream encoder
43655. Fast write calls which assume you control the spare beforehand. * Multibyte write variants exist and use a temporary write pointer * because byte writes alias with anything; with a stored pointer * explicit pointer load/stores get generated (e.g. gcc -Os).
43656. Insert bytes in the middle of the buffer from a slice already * in the buffer. Source offset is interpreted "before" the operation.

43657. Make underlying buffer compact to match DUK_BW_GET_SIZE().
43658. No duk_bw_remove_ensure_slice(), functionality would be identical.
43659. Reset to zero size, keep current limit.
43660. Remove a slice from inside buffer.
43661. remaining
43662. **comment:** Note: cannot read more than 24 bits without possibly shifting top bits out. * Fixable, but adds complexity.
 label: code-design
43663. * Bitstream decoder.
43664. If ctx->offset >= ctx->length, we "shift zeroes in" * instead of croaking.
43665. Decode a one-bit flag, and if set, decode a value of 'bits', otherwise return * default value. Return value is signed so that negative marker value can be * used by caller as a "not present" value.
43666. Extract 'top' bits of curval; note that the extracted bits do not need * to be cleared, we just ignore them on next round.
43667. Decode 'bits' bits from the input stream (bits must be 1...24). * When reading past bitstream end, zeroes are shifted in. The result * is signed to match duk_bd_decode_flagged.
43668. If buffer has been exhausted, truncate bitstream.
43669. **comment:** This limitation would be fixable but adds unnecessary complexity.
 label: code-design
43670. * Bitstream encoder.
43671. buffer size is >= 1
43672. * Fast buffer writer with spare management.
43673. overflow
43674. for manual torture testing: tight allocation, useful with valgrind
43675. Target is before source. Source offset is expressed as * a "before change" offset. Account for the memmove.
43676. point to start of 'reserved area'
43677. not reachable
43678. Don't support "straddled" source now.
43679. **comment:** * Macro support functions for reading/writing raw data. * * These are done using mempcy to ensure they're valid even for unaligned * reads/writes on platforms where alignment counts. On x86 at least gcc * is able to compile these into a bswap+mov. "Always inline" is used to * ensure these macros compile to minimal code. * * Not really bufwriter related, but currently used together.
 label: code-design
43680. Make buffer compact, matching current written size.
43681. **comment:** * Macro support functions (use only macros in calling code)
 label: code-design
43682. We could do this operation without caller updating bw_ctx->ptr, * but by writing it back here we can share code better.
43683. This is important to ensure dynamic buffer data pointer is not * NULL (which is possible if buffer size is zero), which in turn * causes portability issues with e.g. memmove() and memcpy().
43684. Resize target buffer for requested size. Called by the macro only when the * fast path test (= there is space) fails.
43685. **comment:** Portability workaround is required for platforms without * unaligned access. The replacement code emulates little * endian access even on big endian architectures, which is * OK as long as it is consistent for a build.
 label: code-design
43686. DUK_USE_STRHASH_DENSE
43687. * Hash function duk_util_hashbytes(). * * Currently, 32-bit MurmurHash2. * * Don't rely on specific hash values; hash function may be endianness * dependent, for instance.
43688. 'magic' constants for Murmurhash2
43689. prediction corrections for prime list (see genhashsizes.py)
43690. hash size ratio goal, must match genhashsizes.py
43691. Awkward inclusion condition: drop out of compilation if not needed by any * call site: object hash part or probing stringtable.
43692. * Round a number upwards to a prime (not usually the nearest one). * * Uses a table of successive 32-bit primes whose ratio is roughly * constant. This keeps the relative upwards 'rounding error' bounded * and the data size small. A simple 'predict-correct' compression is * used to compress primes to one byte per prime. See genhashsizes.py * for details. * * The minimum prime returned here must be coordinated with the possible * probe sequence steps in duk_hobject and duk_heap stringtable.
43693. prediction: portable variant using doubles if 64-bit values not available
43694. minimum prime
43695. correction
43696. floor(1.15 * (1 << 10))
43697. 32-bit x 11-bit = 43-bit, fits accurately into a double
43698. may happen if size is very close to 2^32-1
43699. probe steps (see genhashsizes.py), currently assumed to be 32 entries long * (DUK_UTIL_GET_HASH_PROBE_STEP macro).
43700. DUK_USE_HOBJECT_HASH_PART || DUK_USE_STRTAB_PROBE
43701. 0x10-0x1f
43702. 0xf0-0xff
43703. 0xb0-0xbf
43704. 0xc0-0xcf
43705. 0x90...0x9f
43706. 0x90-0x9f
43707. 0x10...0x1f
43708. 0xd0...0xdf
43709. 0x20-0x2f
43710. Preshifted << 4. Must use 16-bit entry to allow negative value signaling.
43711. * Misc util stuff
43712. -1 = error, -2 = allowed whitespace, -3 = padding ('='), 0...63 decoded bytes
43713. * Arbitrary byteswap for potentially unaligned values * * Used to byteswap pointers e.g. in debugger code.
43714. For now only needed by the debugger.
43715. 0x60-0x6f
43716. 0x20...0x2f
43717. 0xa0-0xaf
43718. 0x60...0x6f
43719. 0x40...0x4f
43720. 0xe0-0xef
43721. * Table for base-64 decoding
43722. DUK_USE_HEX_FASTPATH
43723. A...P
43724. -1 if invalid
43725. * Table for hex decoding ASCII hex digits
43726. 0x30-0x3f
43727. 0xc0...0xcf
43728. 0x00...0x0f

43729. Lookup to encode one byte directly into 2 characters: * * def genhextab(bswap): * for i in xrange(256): * t = chr(i).encode('hex') * if bswap: * t = t[1] + t[0] * print('0x' + t.encode('hex') + 'U') * print('big endian'); genhextab(False) * print('little endian'); genhextab(True)

43730. 0x80-0x8f

43731. 0xb0...0xbf

43732. Q...f

43733. 0xa0...0xaf

43734. 0x70...0x7f

43735. * Table for hex encoding bytes

43736. 0x40-0x4f

43737. * Table for base-64 encoding

43738. 0x30...0x3f

43739. 0x00-0x0f

43740. 0x80...0x8f

43741. DUK_USE_INTEGER_BE

43742. g...v

43743. 0xf0...0xff

43744. 0x70-0x7f

43745. 0xd0-0xdf

43746. 0x50-0x5f

43747. * Lowercase digits for radix values 2 to 36. Also doubles as lowercase * hex nybble table.

43748. DUK_USE_BASE64_FASTPATH

43749. 0x50...0x5f

43750. 0xe0...0xef

43751. w.../

43752. enough to cover the whole mantissa

43753. if duk_uint32_t is exactly 32 bits, this is a NOP

43754. only use the highest bit

43755. **comment:** * XXX: could make this a lot faster if we create the double memory * representation directly. Feasible easily (must be uniform random).
label: code-design

43756. * A tiny random number generator. * * Currently used for Math.random(). * * http://www.woodmann.com/forum/archive/index.php/t-3100.html

43757. * Context management

43758. Return codes for protected calls (duk_safe_call(), duk_pcall())

43759. RangeError

43760. set/clear writable

43761. DUK_USE_64BIT_OPS

43762. Ecmascript date range is 100 million days from Epoch: * > 100e6 * 24 * 60 * 60 * 1000 // 100M days in millisecs * 8640000000000000 * (= 8.64e15)

43763. * Coercion operations: in-place coercion, return coerced value where * applicable. If index is invalid, throw error. Some coercions may * throw an expected error (e.g. from a toString() or valueOf() call) * or an internal error (e.g. from out of memory).

43764. packed tval

43765. (internal) catch compilation errors

43766. * END PUBLIC API

43767. E == 0x7ff, topmost four bits of F != 0 => assume NaN

43768. APIError

43769. * BEGIN PUBLIC API

43770. timeval breakdown: internal time value NaN -> RangeError (toISOString)

43771. internal flag: external buffer

43772. enumerate internal properties (regardless of enumerability)

43773. Ecmascript E5 specification error codes

43774. * Debugger (debug protocol)

43775. lstring

43776. timeval breakdown: internal time value NaN -> zero

43777. include time part in string conversion result

43778. prefer number

43779. Note: parentheses are required so that the comma expression works in assignments.

43780. no error if file does not exist

43781. internal flag value: throw if mask doesn't match

43782. set getter (given on value stack)

43783. * Memory management * * Raw functions have no side effects (cannot trigger GC).

43784. Coercion hints

43785. is_copy

43786. * Variable access

43787. weekday: 0 to 6, 0=sunday, 1=monday, etc

43788. DUK_DBUNION_H_INCLUDED

43789. lightweight function pointer

43790. * String manipulation

43791. getter: subtract 1900 from year when getting year part

43792. Value mask types, used by e.g. duk_get_type_mask()

43793. Ecmascript undefined

43794. file

43795. Log levels

43796. Ecmascript boolean: 0 or 1

43797. * Public API specific typedefs * * Many types are wrapped by Duktape for portability to rare platforms * where e.g. 'int' is a 16-bit type. See practical typing discussion * in Duktape web documentation.

43798. flags

43799. AssertionError

43800. only enumerate array indices

43801. * Stack management

43802. Flags for duk_def_prop() and its variants

43803. NOTE: when writing a Date provider you only need a few specific * flags from here, the rest are internal. Avoid using anything you * don't need.

43804. * Date provider related constants * * NOTE: These are "semi public" - you should only use these if you write * your own platform specific Date provider, see doc/datetime.rst.

43805. internal: request fixed buffer result

43806. Number of value stack entries (in addition to actual call arguments) * guaranteed to be allocated on entry to a Duktape/C function.

43807. * Indexes of various types with respect to big endian (logical) layout

43808. no error (e.g. from duk_get_error_code())

43809. LICENSE.txt

43810. SyntaxError

43811. * Duktape/C function magic value

43812. internal: request dynamic buffer result
43813. * Avoid C++ name mangling
43814. **comment:** XXX: replace with TypeError?
 label: code-design
43815. **comment:** XXX: to be removed?
 label: code-design
43816. * ===== Duktape license ===== * * (http://opensource.org/licenses/MIT) * * Copyright (c) 2013-2017 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
43817. Git commit, describe, and branch for Duktape build. Useful for * non-official snapshot builds so that application code can easily log * which Duktape snapshot was used. Not available in the Ecmascript * environment.
43818. DUK_API_PUBLIC_H_INCLUDED
43819. Compilation flags for duk_compile() and duk_eval()
43820. * Duktape public API for Duktape 1.8.0. * * See the API reference for documentation on call semantics. * The exposed API is inside the DUK_API_PUBLIC_H_INCLUDED * include guard. Other parts of the header are Duktape * internal and related to platform/compiler/feature detection. * * Git commit 0a70d7e4c5227c84e3fed5209828973117d02849 (v1.8.0). * Git branch v1.8-maintenance. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.
43821. timeval breakdown: convert month and day-of-month parts to one-based (default is zero-based)
43822. Duktape version, (major * 10000) + (minor * 100) + patch. Allows C code * to #ifdef against Duktape API version. The same value is also available * to Ecmascript code in Duktape.version. Unofficial development snapshots * have 99 for patch level (e.g. 0.10.99 would be a development version * after 0.10.0 but before the next official release).
43823. * Buffer
43824. set enumerable (effective if DUK_DEFPROP_HAVE_ENUMERABLE set)
43825. set configurable (effective if DUK_DEFPROP_HAVE_CONFIGURABLE set)
43826. don't walk prototype chain, only check own properties
43827. UnimplementedError
43828. (internal) omit eval result
43829. **comment:** * If no variadic macros, __FILE__ and __LINE__ are passed through globals * which is ugly and not thread safe.
 label: requirement
43830. last value is func pointer, arguments follow in parens
43831. set/clear configurable
43832. Return codes for C functions (shortcut for throwing an error)
43833. * Pop operations
43834. * Require operations: no coercion, throw error if index or type * is incorrect. No defaulting.
43835. * Bytecode load/dump
43836. compile function code (instead of global code)
43837. enumerate a proxy object itself without invoking proxy behavior
43838. **comment:** XXX: native 64-bit byteswaps when available
 label: code-design
43839. **comment:** XXX: These calls are incomplete and not usable now. They are not (yet) * part of the public API.
 label: code-design
43840. Error
43841. DUK_USE_FILE_IO
43842. ReferenceError
43843. timeval breakdown: replace year with equivalent year in the [1971,2037] range for DST calculations
43844. * Ecmascript operators
43845. args
43846. * Some defines forwarded from feature detection
43847. (internal) take strlen() of src_buffer (avoids double evaluation in macro)
43848. set/clear enumerable
43849. internal use
43850. * Error handling
43851. EvalError
43852. * Object prototype
43853. setter: perform 2-digit year fixup (00...99 -> 1900...1999)
43854. nop: no need to normalize
43855. * Compilation and evaluation
43856. or is a normalized NaN
43857. AllocError
43858. month: 0 to 11
43859. **comment:** XXX: Enough space to hold internal suspend/resume structure. * This is rather awkward and to be fixed when the internal * structure is visible for the public API header.
 label: code-design
43860. internal flag: create backing ArrayBuffer; keep in one byte
43861. additional values begin at bit 12
43862. * Constants
43863. **comment:** XXX: replace with plain Error?
 label: code-design
43864. * Object finalizer
43865. Reverse operation is the same.
43866. duk_context is now defined in duk_config.h because it may also be * referenced there by prototypes.
43867. * C++ name mangling
43868. UncaughtError
43869. Ecmascript year range: * > new Date(100e6 * 24 * 3600e3).toISOString() * '+275760-09-13T00:00:00.000Z' * > new Date(-100e6 * 24 * 3600e3).toISOString() * '-271821-04-20T00:00:00.000Z'
43870. create a new global environment
43871. * Global object
43872. DUKTAPE_H_INCLUDED
43873. E == 0x7ff, F != 0 => NaN
43874. DUK_USE_PACKED_TVAL
43875. * Logging
43876. safe variants of a few coercion operations

43877. Indicates that a native function does not have a fixed number of args, * and the argument stack should not be capped/extended at all.

43878. * Get operations: no coercion, returns default value for invalid * indices and invalid value types. ** duk_get_undefined() and duk_get_null() would be pointless and * are not included.

43879. **comment:** use locale specific formatting if available
label: code-design

43880. External duk_config.h provides platform/compiler/OS dependent * typedefs and macros, and DUK_USE_xxx config options so that * the rest of Duktape doesn't need to do any feature detection.

43881. internal: don't care about fixed/dynamic nature

43882. * Property access ** The basic function assumes key is on stack. The _string variant takes * a C string as a property name, while the _index variant takes an array * index as a property name (e.g. 123 is equivalent to the key "123").

43883. * Debugging

43884. 64-bit byteswap, same operation independent of target endianness.

43885. * Type checks ** duk_is_none(), which would indicate whether index is outside of stack, * is not needed; duk_is_valid_index() gives the same information.

43886. sort array indices, use with DUK_ENUM_ARRAY_INDICES_ONLY

43887. force change if possible, may still fail for e.g. virtual properties

43888. Millisecond count constants.

43889. used by packed duk_tval, assumes sizeof(void *) == 4

43890. set value (given on value stack)

43891. Used to represent invalid index; if caller uses this without checking, * this index will map to a non-existent stack entry. Also used in some * API calls as a marker to denote "no value".

43892. * Module helpers: put multiple function or constant properties

43893. raw void pointer

43894. Flags for duk_push_thread_raw()

43895. * ===== Duktape authors * ===== Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang \u00f3rff <llango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6tz <https://github.com/jaseg> * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6man * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * https://github.com/yushli * * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * https://github.com/Kelledin * * https://github.com/sstruktrup * * Michael Drake (<https://github.com/tlsa>) * * https://github.com/chris-y * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn \u00e5s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.

43896. InternalError

43897. Value types, used by e.g. duk_get_type()

43898. **comment:** Duktape specific error codes (must be 8 bits at most, see duk_error.h)
label: code-design

43899. no value, e.g. invalid index

43900. * Stack manipulation (other than push/pop)

43901. Byteswap an IEEE double in the duk_double_union from host to network * order. For a big endian target this is a no-op.

43902. prefer string

43903. external use

43904. Flags for duk_push_string_file_raw()

43905. Duktape debug protocol version used by this build.

43906. **comment:** DUK_COMPILE_xxx bits 0-2 are reserved for an internal 'nargs' argument * (the nargs value passed is direct stack arguments + 1 to account for an * internal extra argument).
label: code-design

43907. fixed or dynamic, garbage collected byte buffer

43908. * Thread management

43909. * Double NaN manipulation macros related to NaN normalization needed when * using the packed duk_tval representation. NaN normalization is necessary * to keep double values compatible with the duk_tval format. ** When packed duk_tval is used, the NaN space is used to store pointers * and other tagged values in addition to NaNs. Actual NaNs are normalized * to a specific quiet NaN. The macros below are used by the implementation * to check and normalize NaN values when they might be created. The macros * are essentially NOPs when the non-packed duk_tval representation is used. ** A FULL check is exact and checks all bits. A NOTFULL check is used by * the packed duk_tval and works correctly for all NaNs except those that * begin with 0x7ff0. Since the 'normalized NaN' values used with packed * duk_tval begin with 0x7ff8, the partial check is reliable when packed * duk_tval is used. The 0x7ff8 prefix means the normalized NaN will be a * quiet NaN regardless of its remaining lower bits. ** The ME variant below is specifically for ARM byte order, which has the * feature that while doubles have a mixed byte order (32107654), unsigned * long long values has a little endian byte order (76543210). When writing * a logical double value through a ULL pointer, the 32-bit words need to be * swapped; hence the #ifdefs below for ULL writes with DUK_USE_DOUBLE_ME. * This is not full ARM support but suffices for some environments.

43910. either not a NaN

43911. convert time value to local time

43912. not directly applicable, byte order differs from a double

43913. end 'extern "C"' wrapper

43914. * Object operations

43915. set writable (effective if DUK_DEFPROP_HAVE_WRITABLE set)

43916. UnsupportedError

43917. **comment:** Internal API call flags, used for various functions in this file. * Certain flags are used by only certain functions, but since the flags * don't overlap, a single flags value can be passed around to multiple * functions. ** The unused top bits of the flags field are also used to pass values * to helpers (duk_get_part_helper() and duk_set_part_helper()). ** (Must be in-sync with genbuiltins.py.)
label: code-design

43918. There are currently no native functions to yield/resume, due to the internal * limitations on coroutine handling. These will be added later.

43919. Concrete macros for NaN handling used by the implementation internals. * Chosen so that they match the duk_tval representation: with a packed * duk_tval, ensure NaNs are properly normalized; with a non-packed duk_tval * these are essentially NOPs.

43920. (internal) no source string on stack

43921. * Function (method) calls

43922. plain

43923. ECMAScript string: CESU-8 / extended UTF-8 encoded

43924. internal flag: don't zero allocated buffer

43925. use strict (outer) context for global, eval, or function code

43926. in non-packed representation we don't care about which NaN is used

43927. * Union for accessing double parts, also serves as packed duk_tval

43928. include date part in string conversion result
43929. internal flag: dynamic buffer
43930. Ecmascript null
43931. enumerate non-enumerable properties in addition to enumerable
43932. **comment:** XXX: replace with RangeError?
 label: code-design
43933. * Push operations * * Push functions return the absolute (relative to bottom of frame) * position of the pushed value for convenience. * * Note: duk_dup() is technically a push.
43934. string
43935. * ROM pointer compression
43936. * Union to access IEEE double memory representation, indexes for double * memory representation, and some macros for double manipulation. * * Also used by packed duk_tval. Use a union for bit manipulation to * minimize aliasing issues in practice. The C99 standard does not * guarantee that this should work, but it's a very widely supported * practice for low level manipulation. * * IEEE double format summary: * * seeeeeeee eeeeefffff ffffffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * See http://en.wikipedia.org/wiki/Double_precision_floating-point_format. * * NaNs are represented as exponent 0x7ff and mantissa != 0. The NaN is a * signaling NaN when the highest bit of the mantissa is zero, and a quiet * NaN when the highest bit is set. * * At least three memory layouts are relevant here: * * A B C D E F G H Big endian (e.g. 68k) DUK_USE_DOUBLE_BE * H G F E D C B A Little endian (e.g. x86) DUK_USE_DOUBLE_LE * D C B A H G F E Mixed/cross endian (e.g. ARM) DUK_USE_DOUBLE_ME * * ARM is a special case: ARM double values are in mixed/cross endian * format while ARM duk_uint64_t values are in standard little endian * format (H G F E D C B A). When a double is read as a duk_uint64_t * from memory, the register will contain the (logical) value * E F G H A B C D. This requires some special handling below. * * Indexes of various types (8-bit, 16-bit, 32-bit) in memory relative to * the logical (big endian) order: * * byte order duk_uint8_t duk_uint16_t duk_uint32_t * BE 01234567 0123 01 * LE 76543210 3210 10 * ME (ARM) 32107654 1032 01 * * Some processors may alter NaN values in a floating point load+store. * For instance, on X86 a FLD + FSTP may convert a signaling NaN to a * quiet one. This is catastrophic when NaN space is used in packed * duk_tval values. See: misc/clang_aliasing.c.
43937. * Helper macros for reading/writing memory representation parts, used * by duk_numconv.c and duk_tval.h
43938. string conversion: use 'T' instead of '' as a separator
43939. Ecmascript number: double
43940. TypeError
43941. Support array for ROM pointer compression. Only declared when ROM * pointer compression is active.
43942. **comment:** Although extra/top could be an unsigned type here, using a signed type * makes the API more robust to calling code calculation errors or corner * cases (where caller might occasionally come up with negative values). * Negative values are treated as zero, which is better than casting them * to a large unsigned number. (This principle is used elsewhere in the * API too.)
 label: code-design
43943. * Misc conversion
43944. Ecmascript object: includes objects, arrays, functions, threads
43945. URIError
43946. all doubles are considered normalized
43947. AUTHORS.rst
43948. day within month: 0 to 30
43949. setter: call is a time setter (affects hour, min, sec, ms); otherwise date setter (affects year, month, day-in-month)
43950. (internal) no filename on stack
43951. compile eval code (instead of global code)
43952. One problem with this macro is that expressions like the following fail * to compile: "(void) duk_error(...)". But because duk_error() is noreturn, * they make little sense anyway.
43953. set setter (given on value stack)
43954. prefer number, unless input is a Date, in which * case prefer string (E5 Section 8.12.8)
43955. Part indices for internal breakdowns. Part order from DUK_DATE_IDX_YEAR * to DUK_DATE_IDX_MILLISECOND matches argument ordering of Ecmascript API * calls (like Date constructor call). Some functions in duk.bi_date.c * depend on the specific ordering, so change with care. 16 bits are not * enough for all parts (year, specifically). * * (Must be in-sync with genbuiltins.py)
43956. E == 0x7ff, F == 8 => normalized NaN
43957. Enumeration flags for duk_enum()
43958. year
43959. * Other state related functions
43960. This should not really happen, but would indicate x64.
43961. **comment:** Minimize warnings for unused internal functions with GCC >= 3.1.1 and * Clang. Based on documentation it should suffice to have the attribute * in the declaration only, but in practice some warnings are generated unless * the attribute is also applied to the definition.
 label: code-design
43962. Emscripten (provided explicitly by user), improve if possible
43963. Both MinGW and MSVC have a 64-bit type.
43964. * Platform autodetection
43965. --- TinyC ---
43966. mint clib is missing these
43967. SuperH
43968. --- x64 ---
43969. MSVC does not have sys/param.h
43970. --- Motorola 68k ---
43971. since gcc-2.5
43972. --- MIPS 64-bit ---
43973. **comment:** XXX: DUK_UNREACHABLE for msvc?
 label: code-design
43974. OpenBSD
43975. NetBSD
43976. Apple OSX, iOS
43977. Cannot determine byte order; __ORDER_PDP_ENDIAN__ is related to 32-bit * integer ordering and is not relevant.
43978. Type for public API calls.
43979. **comment:** Rely as little as possible on compiler behavior for NaN comparison, * signed zero handling, etc. Currently never activated but may be needed * for broken compilers.
 label: code-design
43980. --- Generic BSD ---
43981. uclibc may be missing these
43982. FreeBSD
43983. not defined by default
43984. **comment:** Windows, both 32-bit and 64-bit
 label: code-design
43985. --- OpenBSD ---
43986. **comment:** * Alternative customization header * * If you want to modify the final DUK_USE_xxx flags directly (without * using the available DUK_OPT_xxx flags), define DUK_OPT_HAVE_CUSTOM_H * and tweak the final flags there.
 label: code-design
43987. Cygwin

43988. integer endianness is little on purpose
43989. e.g. `getdate_r`
43990. * Autogenerated defaults
43991. Atari Mint
43992. **comment:** GCC older than 4.6: avoid overflow warnings related to using INFINITY
 label: code-design
43993. Most portable, wastes space
43994. GCC/clang inaccurate math would break compliance and probably `duk_tval`, * so refuse to compile. Relax this if `-ffast-math` is tested to work.
43995. --- Generic ---
43996. VS2012+ has `stdint.h`, < VS2012 does not (but it's available for download).
43997. **comment:** * Alignment requirement and support for unaligned accesses * * Assume unaligned accesses are not supported unless specifically allowed * in the target platform. Some platforms may support unaligned accesses * but alignment to 4 or 8 may still be desirable.
 label: requirement
43998. Clang
43999. AmigaOS. Neither AMIGA nor `__amigaos__` is defined on VBCC, so user must * define 'AMIGA' manually when using VBCC.
44000. * `duk_config.h` configuration header generated by `genconfig.py`. * * Git commit: 0a70d7e4c5227c84e3fed5209828973117d02849 * Git describe: v1.8.0 * Git branch: v1.8-maintenance * * Supported platforms: * - Mac OSX, iPhone, Darwin * - OpenBSD * - Generic BSD * - Atari ST TOS * - AmigaOS * - Windows * - Flashplayer (Crossbridge) * - QNX * - TI-Nspire * - Emscripten * - Linux * - Solaris * - Generic POSIX * - Cygwin * - Generic UNIX * - Generic fallback * * Supported architectures: * - x86 * - x64 * - x32 * - ARM 32-bit * - ARM 64-bit * - MIPS 32-bit * - MIPS 64-bit * - PowerPC 32-bit * - PowerPC 64-bit * - SPARC 32-bit * - SPARC 64-bit * - SuperH * - Motorola 68k * - Emscripten * - Generic * * Supported compilers: * - Clang * - GCC * - MSVC * - Emscripten * - TinyC * - VBCC * - Bruce's C compiler * - Generic *
44001. --- GCC ---
44002. Duktape/C function return value, platform int is enough for now to * represent 0, 1, or negative error code. Must be compatible with * assigning truth values (e.g. `duk_ret_t rc = (foo == bar);`).
44003. Low memory algorithm: separate chaining using arrays, fixed size hash
44004. `__OVERRIDE_DEFINES__`
44005. <http://stackoverflow.com/questions/5919996/how-to-detect-reliably-mac-os-x-ios-linux-windows-in-c-preprocessor>
44006. vbcc + AmigaOS has C99 but no `inttypes.h`
44007. MSVC
44008. * Check whether or not a packed `duk_tval` representation is possible. * What's basically required is that pointers are 32-bit values * (`sizeof(void *) == 4`). Best effort check, not always accurate. * If guess goes wrong, crashes may result; self tests also verify * the guess.
44009. --- Linux ---
44010. --- AmigaOS ---
44011. BCC, assume we're on x86.
44012. Pointer size determination based on `__WORDSIZE` or architecture when * that's not available.
44013. Unsigned index variant.
44014. External provider already defined.
44015. Byte order varies, so rely on autodetect.
44016. `DUK_USE_UNION_INITIALIZERS`: required from compilers, so no fill-in.
44017. **comment:** Special naming to avoid conflict with e.g. `DUK_FREE()` in `duk_heap.h` * (which is unfortunately named). May sometimes need replacement, e.g. * some compilers don't handle zero length or NULL correctly in `realloc()`.
 label: code-design
44018. empty
44019. --- MSVC ---
44020. We're generally assuming that we're working on a platform with a 32-bit * address space. If `DUK_SIZE_MAX` is a typecast value (which is necessary * if `SIZE_MAX` is missing), the check must be avoided because the * preprocessor can't do a comparison.
44021. 64-bit constants. Since LL / ULL constants are not always available, * use computed values. These values can't be used in preprocessor * comparisons; flag them as such.
44022. --- x86 ---
44023. --- MIPS 32-bit ---
44024. XXX: add feature options to force basic types from outside?
44025. Array index values, could be exact 32 bits. * Currently no need for signed `duk_arridx_t`.
44026. `SIZE_MAX` may be missing so use an approximate value for it.
44027. `DUK_F_BCC`
44028. Complex condition broken into separate parts.
44029. POSIX
44030. If not provided, use safe default for alignment.
44031. * Date provider selection * * User may define `DUK_USE_DATE_GET_NOW()` etc directly, in which case we'll * rely on an external provider. If this is not done, revert to previous * behavior and use Unix/Windows built-in provider.
44032. Cannot determine byte order.
44033. --- Cygwin ---
44034. !defined(`DUK_USE_BYTEORDER`) && defined(`__BYTE_ORDER__`)
44035. More or less standard endianness predefines provided by header files. * The ARM hybrid case is detected by assuming that `__FLOAT_WORD_ORDER` * will be big endian, see: <http://lists.mysql.com/internals/443>. * On some platforms some defines may be present with an empty value which * causes comparisons to fail: <https://github.com/svaarala/duktape/issues/453>.
44036. !defined(`DUK_USE_BYTEORDER`)
44037. **comment:** For custom platforms allow user to define byteorder explicitly. * Since endianness headers are not standardized, this is a useful * workaround for custom platforms for which endianness detection * is not directly supported. Perhaps custom hardware is used and * user cannot submit upstream patches.
 label: code-design
44038. **comment:** Don't know how to declare unreachable point, so don't do it; this * may cause some spurious compilation warnings (e.g. "variable used * uninitialized").
 label: code-design
44039. Good default is a bit arbitrary because alignment requirements * depend on target. See <https://github.com/svaarala/duktape/issues/102>.
44040. --- Windows ---
44041. User provided `InitJS`.
44042. These have been tested from VS2008 onwards; may work in older VS versions * too but not enabled by default.
44043. Error codes are represented with platform int. High bits are used * for flags and such, so 32 bits are needed.
44044. Intermediate define for 'have `inttypes.h`'
44045. build is not C99 or C++11, play it safe
44046. GCC: test not very accurate; enable only in relatively recent builds * because of bugs in gcc-4.4 (<http://lists.debian.org/debian-gcc/2010/04/msg00000.html>)
44047. Intel x86 (32-bit), x64 (64-bit) or x32 (64-bit but 32-bit pointers), * define only one of `DUK_F_X86`, `DUK_F_X64`, `DUK_F_X32`. * <https://sites.google.com/site/x32abi/>
44048. Enabled with debug/assertions just so that any issues can be caught.
44049. C99 or compatible
44050. --- Mac OSX, iPhone, Darwin ---
44051. C99 or above
44052. MinGW. Also GCC flags (`DUK_F_GCC`) are enabled now.
44053. This detection is not very reliable.
44054. Support for 48-bit signed integer `duk_tval` with transparent semantics.

44055. GCC and Clang provide endianness defines as built-in predefines, with * leading and trailing double underscores (e.g. __BYTE_ORDER__). See * output of "make gcpredes" and "make clangpredes". Clang doesn't * seem to provide __FLOAT_WORD_ORDER__; assume not mixed endian for clang. * <http://gcc.gnu.org/onlinedocs/cpp/Common-Predefined-Macros.html>
44056. 64-bit type detection is a bit tricky. * * ULLONG_MAX is a standard define. __LONG_LONG_MAX__ and __ULONG_LONG_MAX__ * are used by at least GCC (even if system headers don't provide ULLONG_MAX). * Some GCC variants may provide __LONG_LONG_MAX__ but not __ULONG_LONG_MAX__. * * ULL / LL constants are rejected / warned about by some compilers, even if * the compiler has a 64-bit type and the compiler/system headers provide an * unsupported constant (ULL/LL)! Try to avoid using ULL / LL constants. * As a side effect we can only check that e.g. ULONG_MAX is larger than 32 * bits but can't be sure it is exactly 64 bits. Self tests will catch such * cases.
44057. Convert any input pointer into a "void **", losing a const qualifier. * This is not fully portable because casting through duk_uintptr_t may * not work on all architectures (e.g. those with long, segmented pointers).
44058. Small integers (16 bits or more) can fall back to the 'int' type, but * have a typedef so they are marked "small" explicitly.
44059. **comment:** Most portable, potentially wastes space
label: code-design
44060. PowerPC
44061. --- Solaris ---
44062. **comment:** Technically C99 (C++11) but found in many systems. On some systems * __STDC_LIMIT_MACROS and __STDC_CONSTANT_MACROS must be defined before * including stdint.h (see above).
label: code-design
44063. Missing some obvious constants.
44064. No provider for DUK_USE_DATE_PARSE_STRING(), fall back to ISO 8601 only.
44065. no endian.h or stdint.h
44066. --- Atari ST TOS ---
44067. GCC: assume we have __va_copy() in non-C99 mode.
44068. User forced alignment to 4 or 8.
44069. C99 / C++11 and above: rely on va_copy() which is required.
44070. <http://bellard.org/tcc/tcc-doc.html#SEC9>
44071. Check that architecture is two's complement, standard C allows e.g. * INT_MIN to be -2**31+1 (instead of -2**31).
44072. Based on 'make checkalign' there are no alignment requirements on * Linux MIPS except for doubles, which need align by 4. Alignment * requirements vary based on target though.
44073. AmigaOS on M68k
44074. not configured for DLL build
44075. * You may add overriding #define/#undef directives below for * customization. You of course cannot un-#include or un-typedef * anything; these require direct changes above.
44076. **comment:** cannot detect 64-bit type, not always needed so don't error
label: code-design
44077. --- Generic UNIX ---
44078. std::exception
44079. A few types are assumed to always exist.
44080. Float word order not known, assume not a hybrid.
44081. --- Generic fallback ---
44082. **comment:** --- Generic POSIX ---
label: code-design
44083. no parsing (not an error)
44084. **comment:** ANSI C (various versions) and some implementations require that the * pointer arguments to memset(), memcpy(), and memmove() be valid values * even when byte size is 0 (even a NULL pointer is considered invalid in * this context). Zero-size operations as such are allowed, as long as their * pointer arguments point to a valid memory area. The DUK_MEMSET(), * DUK_MEMCPY(), and DUK_MEMMOVE() macros require this same behavior, i.e.: * (1) pointers must be valid and non-NULL, (2) zero size must otherwise be * allowed. If these are not fulfilled, a macro wrapper is needed. * * <http://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0> * <http://lists.cs.uiuc.edu/pipermail/l1vmdev/2007-October/011065.html> * * Not sure what's the required behavior when a pointer points just past the * end of a buffer, which often happens in practice (e.g. zero size memmoves). * For example, if allocation size is 3, the following pointer would not * technically point to a valid memory byte: * * <-- alloc --> * | 0 | 1 | 2 | * ^- - p=3, points after last valid byte (2)
label: code-design
44085. C++11 or above
44086. SPARC byte order varies so rely on autodetection.
44087. DUK_F_PACKED_TVAL_PROVIDED
44088. <http://www.monkey.org/openbsd/archive/ports/0401/msg00089.html>
44089. VBCC
44090. sigsetjmp() alternative
44091. GCC, Clang also defines __GNUC__ so don't detect GCC if using Clang.
44092. --- SuperH ---
44093. Pre-C99: va_list type is implementation dependent. This replacement * assumes it is a plain value so that a simple assignment will work. * This is not the case on all platforms (it may be a single-array element, * for instance).
44094. **comment:** The best type for an "all around int" in Duktape internals is "at least * 32 bit signed integer" which is most convenient. Same for unsigned type. * Prefer 'int' when large enough, as it is almost always a convenient type.
label: code-design
44095. uclibc
44096. Explicit marker needed; may be 'defined', 'undefined', 'or 'not provided'.
44097. No provider for DUK_USE_DATE_FORMAT_STRING(), fall back to ISO 8601 only.
44098. **comment:** Many platforms are missing fpclassify() and friends, so use replacements * if necessary. The replacement constants (FP_NAN etc) can be anything but * match Linux constants now.
label: code-design
44099. * Checks for config option consistency (DUK_USE_xxx)
44100. autodetect compiler
44101. Rely on C89 headers only; time.h must be here.
44102. These functions don't currently need replacement but are wrapped for * completeness. Because these are used as function pointers, they need * to be defined as concrete C functions (not macros).
44103. --- TI-Nspire ---
44104. **comment:** We need va_copy() which is defined in C99 / C++11, so an awkward * replacement is needed for pre-C99 / pre-C++11 environments. This * will quite likely need portability hacks for some non-C99 * environments.
label: code-design
44105. AmigaOS on M68K or PPC is always big endian.
44106. Feature option forcing.
44107. --- ARM 64-bit ---
44108. * Feature option handling
44109. **comment:** * Wrapper typedefs and constants for integer types, also sanity check types. * * C99 typedefs are quite good but not always available, and we want to avoid * forcibly redefining the C99 typedefs. So, there are Duktape wrappers for * all C99 typedefs and Duktape code should only use these typedefs. Type * detection when C99 is not supported is best effort and may end up detecting * some types incorrectly. * * Pointer sizes are a portability problem: pointers to different types may * have a different size and function pointers are very difficult to manage * portably. * * http://en.wikipedia.org/wiki/C_data_types#Fixed

width_integer_types * * Note: there's an interesting corner case when trying to define minimum * signed integer value constants which leads to the current workaround of * defining e.g. -0x80000000 as (-0x7fffffffL - 1L). See doc/code-issues.txt * for a longer discussion. * * Note: avoid typecasts and computations in macro integer constants as they * can then no longer be used in macro relational expressions (such as * #if DUK_SIZE_MAX < 0xffffffffUL). There is internal code which relies on * being able to compare DUK_SIZE_MAX against a limit.

label: code-design

44110. Not standard but common enough

44111. --- QNX ---

44112. **comment:** Object property allocation layout has implications for memory and code * footprint and generated code size/speed. The best layout also depends * on whether the platform has alignment requirements or benefits from * having mostly aligned accesses.

label: code-design

44113. (v)snprintf() is missing before MSVC 2015. Note that _(v)snprintf() does * NOT NUL terminate on truncation, but Duktape code never assumes that. * http://stackoverflow.com/questions/2915672/snprintf-and-visual-studio-2010

44114. Motorola 68K. Not defined by VBCC, so user must define one of these * manually when using VBCC.

44115. http://msdn.microsoft.com/en-us/library/aa235362(VS.60).aspx

44116. Non-C99 case, still relying on DUK_UINTPTR_MAX, as long as it is not a computed value

44117. http://bellard.org/tcc/tcc-doc.html#SEC7

44118. * Intermediate helper defines

44119. **comment:** Macro hackery to convert e.g. __LINE__ to a string without formatting, * see: http://stackoverflow.com/questions/240353/convert-a-preprocessor-token-to-a-string

label: code-design

44120. Flash player (e.g. Crossbridge)

44121. no endian.h

44122. Shared includes: C89

44123. integer byte order

44124. float word order

44125. * Convert DUK_USE_BYTEORDER, from whatever source, into currently used * internal defines. If detection failed, #error out.

44126. e.g. ptrdiff_t

44127. **comment:** XXX: DUK_NOINLINE, DUK_INLINE, DUK_ALWAYS_INLINE for msvc?

label: code-design

44128. SPARC

44129. VBCC supports C99 so check only for C99 for union initializer support. * Designated union initializers would possibly work even without a C99 check.

44130. C++11 apparently ratified stdint.h

44131. QNX gcc cross compiler seems to define e.g. __LITTLEENDIAN__ or __BIGENDIAN__: * \$ /opt/qnx650/host/linux/x86/usr/bin/i486-pc-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 67:#define __LITTLEENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/mips-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 81:#define __BIGENDIAN__ 1 * \$ /opt/qnx650/host/linux/x86/usr/bin/arm-unknown-nto-qnx6.5.0-gcc -dM -E -</dev/null | grep -ni endian * 70:#define __LITTLEENDIAN__ 1

44132. At least some uclibc versions have broken floating point math. For * example, fpclassify() can incorrectly classify certain NaN formats. * To be safe, use replacements.

44133. Placeholder fix for (detection is wider than necessary): * http://llvm.org/bugs/show_bug.cgi?id=17788

44134. snprintf() is technically not part of C89 but usually available.

44135. --- x32 ---

44136. Basic integer typedefs and limits, preferably from inttypes.h, otherwise * through automatic detection.

44137. **comment:** Only include when compiling Duktape to avoid polluting application build * with a lot of unnecessary defines.

label: code-design

44138. --- VBCC ---

44139. On some systems SIZE_MAX can be smaller than max unsigned 32-bit value * which seems incorrect if size_t is (at least) an unsigned 32-bit type. * However, it doesn't seem useful to error out compilation if this is the * case.

44140. **comment:** On other platforms use layout 2, which requires some padding but * is a bit more natural than layout 3 in ordering the entries. Layout * 3 is currently not used.

label: code-design

44141. VS2005+ should have variadic macros even when they're not C99.

44142. illumos / Solaris

44143. DUK_COMPILING_DUKTAPE

44144. DUK_OPT_FORCE_BYTEORDER

44145. QNX

44146. since gcc-4.5

44147. **comment:** Windows 32-bit and 64-bit are currently the same.

label: code-design

44148. This is optionally used by panic handling to cause the program to segfault * (instead of e.g. abort()) on panic. Valgrind will then indicate the C * call stack leading to the panic.

44149. IEEE float/double typedef.

44150. --- ARM 32-bit ---

44151. **comment:** XXX: This is technically not guaranteed because it's possible to configure * an x86 to require aligned accesses with Alignment Check (AC) flag.

label: code-design

44152. TinyC

44153. **comment:** Note: the funny looking computations for signed minimum 16-bit, 32-bit, and * 64-bit values are intentional as the obvious forms (e.g. -0x80000000L) are * -not- portable. See code-issues.txt for a detailed discussion.

label: code-design

44154. These are necessary wild guesses.

44155. Already provided above

44156. Atari ST TOS. __TOS__ defined by PureC. No platform define in VBCC * apparently, so to use with VBCC user must define __TOS__ manually.

44157. Byte order is big endian but cannot determine IEEE double word order.

44158. --- Flashplayer (Crossbridge) ---

44159. MIPS byte order varies so rely on autodetection.

44160. stdint.h not available

44161. **comment:** The most portable way to figure out local time offset is gmtime(), * but it's not thread safe so use with caution.

label: requirement

44162. Codepoint type. Must be 32 bits or more because it is used also for * internal codepoints. The type is signed because negative codepoints * are used as internal markers (e.g. to mark EOF or missing argument). * (X)UTF-8/CESU-8 encode/decode take and return an unsigned variant to * ensure duk_uint32_t casts back and forth nicely. Almost everything * else uses the signed one.

44163. DUK_USE_VARIADIC_MACROS: required from compilers, so no fill-in.

44164. TOS on M68K is always big endian.

44165. don't use strftime() for now

44166. **comment:** not sure, not needed with C99 anyway

label: requirement

44167. C++ doesn't have standard designated union initializers ({ .foo = 1 }).

44168. --- Emscripten ---

44169. nop

44170. DLL build detection
44171. **comment:** Avoid custom date parsing and formatting for portability.
label: code-design
44172. Workaround for older C++ compilers before including <inttypes.h>, * see e.g.: https://sourceware.org/bugzilla/show_bug.cgi?id=15366
44173. * Compiler autodetection
44174. --- SPARC 32-bit ---
44175. --- Clang ---
44176. On some platforms int is 16-bit but long is 32-bit (e.g. PureC)
44177. **comment:** * Byte order and double memory layout detection ** Endianness detection is a major portability hassle because the macros * and headers are not standardized. There's even variance across UNIX * platforms. Even with "standard" headers, details like underscore count * varies between platforms, e.g. both __BYTE_ORDER and _BYTE_ORDER are used * (Crossbridge has a single underscore, for instance). ** The checks below are structured with this in mind: several approaches are * used, and at the end we check if any of them worked. This allows generic * approaches to be tried first, and platform/compiler specific hacks tried * last. As a last resort, the user can force a specific endianness, as it's * not likely that automatic detection will work on the most exotic platforms. ** Duktape supports little and big endian machines. There's also support * for a hybrid used by some ARM machines where integers are little endian * but IEEE double values use a mixed order (12345678 -> 43218765). This * byte order for doubles is referred to as "mixed endian".
label: code-design
44178. **comment:** Check whether we should use 64-bit integers or not. ** Quite incomplete now. Use 64-bit types if detected (C99 or other detection) * unless they are known to be unreliable. For instance, 64-bit types are * available on VBCC but seem to misbehave.
label: code-design
44179. VS2013+ supports union initializers but there's a bug involving union-inside-struct: * <https://connect.microsoft.com/VisualStudio/feedback/details/805981> * The bug was fixed (at least) in VS2015 so check for VS2015 for now: * <https://blogs.msdn.microsoft.com/vcblog/2015/07/01/c-compiler-front-end-fixes-in-vs2015/> * Manually tested using VS2013, CL reports 18.00.31101, so enable for VS2013 too.
44180. no user declarations
44181. DUK_CONFIG_H_INCLUDED
44182. BCC (Bruce's C compiler): this is a "torture target" for compilation
44183. Generic Unix (includes Cygwin)
44184. There was a curious bug where test-bi-date-canceling.js would fail e.g. * on 64-bit Ubuntu, gcc-4.8.1, -m32, and no -std=c99. Some date computations * using doubles would be optimized which then broke some corner case tests. * The problem goes away by adding 'volatile' to the datetime computations. * Not sure what the actual triggering conditions are, but using this on * non-C99 systems solves the known issues and has relatively little cost * on other platforms.
44185. C99 / C++11 and above: rely on va_copy() which is required. * Omit parenthesis on macro right side on purpose to minimize differences * to direct use.
44186. C99 types
44187. DUK_SIZE_MAX (= SIZE_MAX) is often reliable
44188. --- PowerPC 64-bit ---
44189. See: /opt/qnx650/target/qnx6/usr/include/sys/platform.h
44190. --- SPARC 64-bit ---
44191. On platforms without any alignment issues, layout 1 is preferable * because it compiles to slightly less code and provides direct access * to property keys.
44192. Strict C99 case: DUK_UINTPTR_MAX (= UINTPTR_MAX) should be very reliable
44193. autodetect platform
44194. **comment:** NetBSD 6.0 x86 (at least) has a few problems with pow() semantics, * see test-bug-netbsd-math-pow.js. Use NetBSD specific workaround. * (This might be a wider problem; if so, generalize the define name.)
label: code-design
44195. autodetect architecture
44196. In VBCC (0.0 / 0.0) results in a warning and 0.0 instead of NaN. * In MSVC (VS2010 Express) (0.0 / 0.0) results in a compile error. * Use a computed NaN (initialized when a heap is created at the * latest).
44197. Clang: assume we have __va_copy() in non-C99 mode.
44198. --- Bruce's C compiler ---
44199. Same as 'duk_int_t' but guaranteed to be a 'fast' variant if this * distinction matters for the CPU. These types are used mainly in the * executor where it might really matter.
44200. vbcc + AmigaOS may be missing these
44201. C99 or C++11, no known issues
44202. **comment:** XXX: DUK_LIKELY, DUK_UNLIKELY for msvc?
label: code-design
44203. Initial fix: disable secure CRT related warnings when compiling Duktape * itself (must be defined before including Windows headers). Don't define * for user code including duktape.h.
44204. varargs
44205. * Fill-ins for platform, architecture, and compiler
44206. e.g. strptime
44207. **comment:** Note: PRS and FMT are intentionally left undefined for now. This means * there is no platform specific date parsing/formatting but there is still * the ISO 8601 standard format.
label: code-design
44208. vsnprintf() is technically not part of C89 but usually available.
44209. Rely on autodetection for byte order, alignment, and packed tval.
44210. AmigaOS + M68K seems to have math issues even when using GCC cross * compilation. Use replacements for all AmigaOS versions on M68K * regardless of compiler.
44211. Byte order is little endian but cannot determine IEEE double word order.
44212. C++
44213. MIPS. Related defines: __MIPSEB__, __MIPSEL__, __mips_isa_rev, __LP64__
44214. VBCC is missing the built-ins even in C99 mode (perhaps a header issue).
44215. __MSC_VER
44216. **comment:** Macro for suppressing warnings for potentially unreferenced variables. * The variables can be actually unreferenced or unreferenced in some * specific cases only; for instance, if a variable is only debug printed, * it is unreferenced when debug printing is disabled.
label: code-design
44217. Byte order varies, rely on autodetection.
44218. Use __setjmp() on Apple by default, see GH-55.
44219. TI-Nspire (using Ntless)
44220. --- PowerPC 32-bit ---
44221. Some math functions are C99 only. This is also an issue with some * embedded environments using uclibc where uclibc has been configured * not to provide some functions. For now, use replacements whenever * using uclibc.
44222. Most portable
44223. Boolean values are represented with the platform 'int'.
44224. Shared includes: stdint.h is C99
44225. BSD variant
44226. Old uclibcs have a broken memcpy so use memmove instead (this is overly wide * now on purpose): <http://lists.uclibc.org/pipermail/uclibc-cvs/2008-October/025511.html>
44227. date.h is omitted, and included per platform
44228. * Architecture autodetection
44229. The most portable current time provider is time(), but it only has a * one second resolution.
44230. Convenience, e.g. gcc 4.5.1 == 40501; <http://stackoverflow.com/questions/6031819/emulating-gccs-built-in-unreachable>

44231. defined(DUK_USE_BYTEORDER)
44232. In VBCC (1.0 / 0.0) results in a warning and 0.0 instead of infinity. * Use a computed infinity (initialized when a heap is created at the * latest).
44233. For now, hash part is dropped if and only if 16-bit object fields are used.
44234. MSVC dllexport/dllimport: appropriate __declspec depends on whether we're * compiling Duktape or the application.
44235. **comment:** On Windows, assume we're little endian. Even Itanium which has a * configurable endianness runs little endian in Windows.
 label: code-design
44236. **comment:** Compiler specific hackery needed to force struct size to match alignment, * see e.g. duk_hbuffer.h. * * http://stackoverflow.com/questions/11130109/c-struct-size-alignment * http://stackoverflow.com/questions/10951039/specifying-64-bit-alignment
 label: code-design
44237. Index values must have at least 32-bit signed range.
44238. **comment:** Workaround for GH-323: avoid inlining control when compiling from * multiple sources, as it causes compiler portability trouble.
 label: code-design
44239. byte order
44240. Linux
44241. Based on 'make checkalign' there are no alignment requirements on * Linux SH4, but align by 4 is probably a good basic default.
44242. MSVC preprocessor defines: http://msdn.microsoft.com/en-us/library/b0084kay.aspx * _MSC_FULL_VER includes the build number, but it has at least two formats, see e.g. * BOOST_MSVC_FULL_VER in http://www.boost.org/doc/libs/1_52_0/boost/config/compiler/visualc.hpp
44243. ARM
44244. Fast variants of small integers, again for really fast paths like the * executor.
44245. **comment:** When C99 types are not available, we use heuristic detection to get * the basic 8, 16, 32, and (possibly) 64 bit types. The fast/least * types are then assumed to be exactly the same for now: these could * be improved per platform but C99 types are very often now available. * 64-bit types are not available on all platforms; this is OK at least * on 32-bit platforms. * * This detection code is necessarily a bit hacky and can provide typedefs * and defines that won't work correctly on some exotic platform.
 label: code-design
44246. **comment:** * Cleanups (all statement parsing flows through here). * * Pop label site and reset labels. Reset 'next temp' to value at * entry to reuse temps.
 label: code-design
44247. **comment:** * Reference counting helper macros. The macros take a thread argument * and must thus always be executed in a specific thread context. The * thread argument is needed for features like finalization. Currently * it is not required for INCREF, but it is included just in case. * * Note that 'raw' macros such as DUK_HEAPHDR_GET_REFCOUNT() are not * defined without DUK_USE_REFERENCE_COUNTING, so caller must #ifdef * around them.
 label: code-design
44248. alloc external with size zero
44249. 'i' is the first entry we'll keep
44250. If blen <= 0xffffUL, clen is also guaranteed to be <= 0xffffUL.
44251. Set catcher regs: idx_base+0 = value, idx_base+1 = lj_type.
44252. resend state next time executor is about to run
44253. **comment:** XXX: avoid this check somehow
 label: code-design
44254. Should match Function.prototype.toString()
44255. * Main struct
44256. '/'
44257. * Unwind debugger state. If we unwind while stepping * (either step over or step into), pause execution.
44258. Duktape.modLoaded[] module cache
44259. [obj handler trap]
44260. Not safe to use 'reg_varbind' as assignment expression * value, so go through a temp.
44261. **comment:** size for formatting buffers
 label: code-design
44262. default case exists: go there if no case matches
44263. registers are mutable, non-deletable
44264. '{'
44265. If the debugger is active we need to force an interrupt so that * debugger breakpoints are rechecked. This is important for function * calls caused by side effects (e.g. when doing a DUK_OP_GETPROP), see * GH-303. Only needed for success path, error path always causes a * breakpoint recheck in the executor. It would be enough to set this * only when returning to an Ecmascript activation, but setting the flag * on every return should have no ill effect.
44266. get or set a range of flags; m=first bit number, n=number of bits
44267. [level obj func pc line]
44268. * Ecmascript compliant [[GetProperty]](P), for internal use only. * * If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero * * May cause arbitrary side effects and invalidate (most) duk_tval * pointers.
44269. [arg1 ... argN this]
44270. * Matching order: * * Punctuator first chars, also covers comments, regexps * LineTerminator * Identifier or reserved word, also covers null/true/false literals * NumericLiteral * StringLiteral * EOF * * The order does not matter as long as the longest match is * always correctly identified. There are order dependencies * in the clauses, so it's not trivial to convert to a switch.
44271. Allocator functions.
44272. next does not exist or next is not a letter
44273. * Create and push an error object onto the top of stack. * If a "double error" occurs, use a fixed error instance * to avoid further trouble.
44274. 'enum'
44275. SameValue(NaN, NaN) = true, regardless of NaN sign or extra bits
44276. just making sure
44277. needed for inf: causes mantissa to become zero, * and rounding to be skipped.
44278. String table assert check omitted from 1.x branch * backport.
44279. **comment:** XXX: string not shared because it is conditional
 label: code-design
44280. carry
44281. Initial bytes for markers.
44282. queue back to heap_allocated
44283. XXX: We can't resize the value stack to a size smaller than the * current top, so the order of the resize and adjusting the stack * top depends on the current vs. final size of the value stack. * The operations could be combined to avoid this, but the proper * fix is to only grow the value stack on a function call, and only * shrink it (without throwing if the shrink fails) on function * return.
44284. print provider
44285. form: { DecimalDigits , }, val1 = min count
44286. * Figure out the final, non-bound constructor, to get "prototype" * property.
44287. **comment:** * Note: * * - duk__match_regexp() is required not to longjmp() in ordinary "non-match" * conditions; a longjmp() will terminate the entire matching process. * * - Clearing saved[] is not necessary because backtracking does it * * - Backtracking also rewinds ctx.recursion back to zero, unless an * internal/limit error occurs (which causes a longjmp()) * * - If we supported anchored matches, we would break out here * unconditionally; however, Ecmascript regexps don't have anchored * matches. It might make sense to implement a fast bail-out if * the regexp begins with '^' and sp is not 0: currently we'll just * run through the entire input string, trivially failing the match * at every non-zero offset.
 label: code-design
44288. DUK_TOK_LPAREN

44289. caller ensures; rom objects are never bufferobjects now
44290. Would be nice to bulk clear the allocation, but the context * is 1-2 kilobytes and nothing should rely on it being zeroed.
44291. 1 -> needs a PUTVAR
44292. * Post-increment/decrement will return the original value as its * result value. However, even that value will be coerced using * ToNumber() which is quite awkward. Specific bytecode opcodes * are used to handle these semantics. * * Note that post increment/decrement has a "no LineTerminator here" * restriction. This is handled by duk__expr_lbp(), which forcibly terminates * the previous expression if a LineTerminator occurs before '++'/'--'.
44293. leave out trailing 'unused' elements
44294. **comment:** TODO: implement Proxy object support here
label: requirement
44295. * Multiply + add + carry for 32-bit components using only 16x16->32 * multiplies and carry detection based on unsigned overflow. * * 1st mult, 32-bit: (A*2^16 + B) * 2nd mult, 32-bit: (C*2^16 + D) * 3rd add, 32-bit: E * 4th add, 32-bit: F * * (AC*2^16 + B) * (C*2^16 + D) + E + F * = AC*2^32 + AD*2^16 + BC*2^16 + BD + E + F * = AC*2^32 + (AD + BC)*2^16 + (BD + E + F) * = AC*2^32 + AD*2^16 + BC*2^16 + (BD + E + F)
44296. args start at index 2
44297. * Various Unicode help functions for character classification predicates, * case conversion, decoding, etc.
44298. eat colon
44299. always set; points to a reserved valstack slot
44300. Track number of escapes: necessary for proper keyword * detection.
44301. * Node.js Buffer.prototype.write(string, [offset], [length], [encoding])
44302. IdentifierPart production with IdentifierStart, ASCII, and non-BMP excluded
44303. should never happen but default here
44304. 'env'
44305. Preliminaries, required by setjmp() handler. Must be careful not * to throw an unintended error here.
44306. -> [val]
44307. emitted
44308. resume delete to target
44309. -> [... obj]
44310. success
44311. +0.0
44312. replacer is a mutation risk
44313. parse key and value
44314. **comment:** * Arithmetic operations other than '+' have number-only semantics * and are implemented here. The separate switch-case here means a * "double dispatch" of the arithmetic opcode, but saves code space. * * E5 Sections 11.5, 11.5.1, 11.5.2, 11.5.3, 11.6, 11.6.1, 11.6.2, 11.6.3.
label: code-design
44315. No duk_bw_remove_ensure_slice(), functionality would be identical.
44316. if boolean matches A, skip next inst
44317. ($\geq e 0$) AND ($= f (\text{expt } b (- p 1))$) * * be <- ($\text{expt } b e$) == b^{e-1} * be1 <- (* be b) == ($\text{expt } b (+ e 1)$) == $b^{(e+1)}$ * r <- (* f be1 2) == $2 * f * b^{(e+1)}$ [if $b==2$ -> f * $b^{(e+2)}$] * s <- (* b 2) [if $b==2$ -> 4] * m+ <- be1 == $b^{(e+1)}$ * m- <- be == b^{e-1} * k <- 0 * B <- B * low_ok <- round * high_ok <- round
44318. Perform fixed-format rounding.
44319. If not within EcmaScript range, some integer time calculations * won't work correctly (and some asserts will fail), so bail out * if so. This fixes test-bug-date-insane-setyear.js. There is * a +/- 24h leeway in this range check to avoid a test262 corner * case documented in test-bug-date-timeval-edges.js.
44320. Failed to match the quantifier, restore lexer and parse * opening brace as a literal.
44321. [... val]
44322. 'true'
44323. **comment:** If we don't have a jmpbuf_ptr, there is little we can do * except panic. The caller's expectation is that we never * return. * * With C++ exceptions we now just propagate an uncaught error * instead of invoking the fatal error handler. Because there's * a dummy jmpbuf for C++ exceptions now, this could be changed.
label: code-design
44324. For now shared handler is fine.
44325. may be less, since DELETED entries are NULLed by rehash
44326. Handle single character fills as memset() even when * the fill data comes from a one-char argument.
44327. OK
44328. **comment:** * Bytecode dump/load * * The bytecode load primitive is more important performance-wise than the * dump primitive. * * Unlike most Duktape API calls, bytecode dump/load is not guaranteed to be * memory safe for invalid arguments - caller beware! There's little point * in trying to achieve memory safety unless bytecode instructions are also * validated which is not easy to do with indirect register references etc.
label: code-design
44329. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
44330. maximum loopcount for peephole optimization
44331. cleared before entering finally
44332. flags for duk_hobject_get_own_propdesc() and variants
44333. **comment:** Execute finalizers before freeing the heap, even for reachable * objects, and regardless of whether or not mark-and-sweep is * enabled. This gives finalizers the chance to free any native * resources like file handles, allocations made outside Duktape, * etc. This is quite tricky to get right, so that all finalizer * guarantees are honored. * * XXX: this perhaps requires an execution time limit.
label: code-design
44334. useful for patching jumps later
44335. avoid attempts to add/remove object keys
44336. The get/set pointers could be 16-bit pointer compressed but it * would make no difference on 32-bit platforms because duk_tval is * 8 bytes or more anyway.
44337. * Local defines
44338. **comment:** XXX: compression
label: code-design
44339. already NULLED (by unwind)
44340. Don't allow actual chars after equal sign.
44341. -> [Object defineProperty undefined obj key desc]
44342. whether to use macros or helper function depends on call count
44343. DUK_USE_FASTINT
44344. 7: toISOString
44345. **comment:** XXX: "read only object" ?
label: code-design
44346. no action
44347. **comment:** XXX: There is currently no support for writing buffer object * indexed elements here. Attempt to do so will succeed and * write a concrete property into the buffer object. This should * be fixed at some point but because buffers are a custom feature * anyway, this is relatively unimportant.
label: code-design
44348. must hold DUK_VARARGS
44349. capture is 'undefined', always matches!
44350. **comment:** * XXX: could make this a lot faster if we create the double memory * representation directly. Feasible easily (must be uniform random).
label: code-design
44351. 'raw'
44352. DUK_TOK_RPAREN
44353. request to create catch binding

44354. for object-bound identifiers
44355. **comment:** XXX: compression (as an option)
 label: code-design
44356. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. ** This handling is now identical for C and Ecmascript functions. * C functions always have the 'NEWENV' flag set, so their * environment record initialization is delayed (which is good). ** Delayed creation (on demand) is handled in duk_js_var.c.
44357. **comment:** * Declaration binding instantiation conceptually happens when calling a * function; for us it essentially means that function prologue. The * conceptual process is described in E5 Section 10.5. ** We need to keep track of all encountered identifiers to (1) create an * identifier-to-register map ("varmap"); and (2) detect duplicate * declarations. Identifiers which are not bound to registers still need * to be tracked for detecting duplicates. Currently such identifiers * are put into the varmap with a 'null' value, which is later cleaned up. ** To support functions with a large number of variable and function * declarations, registers are not allocated beyond a certain limit; * after that limit, variables and functions need slow path access. * Arguments are currently always register bound, which imposes a hard * (and relatively small) argument count limit. ** Some bindings in E5 are not configurable (= deletable) and almost all * are mutable (writable). Exceptions are: ** - The 'arguments' binding, established only if no shadowing argument * or function declaration exists. We handle 'arguments' creation * and binding through an explicit slow path environment record. ** - The "name" binding for a named function expression. This is also * handled through an explicit slow path environment record.
 label: code-design
44358. first coerce to a plain value
44359. -> [... func varmap enum]
44360. * Replace lexical environment for global scope ** Create a new object environment for the global lexical scope. * We can't just reset the _Target property of the current one, * because the lexical scope is shared by other threads with the * same (initial) built-ins.
44361. force_global
44362. NB: use 's' as temp on purpose
44363. Not odd, or y == -Infinity
44364. * Global object built-ins
44365. A bunch of helpers (for size optimization) that combine duk__expr()/duk__exprtop() * and result conversions. ** Each helper needs at least 2-3 calls to make it worth while to wrap.
44366. Value stack: these are expressed as pointers for faster stack manipulation. * [valstack, valstack_top[is GC-reachable, [valstack_top, valstack_end[is * not GC-reachable but kept initialized as 'undefined'.
44367. Output a specified number of digits instead of using the shortest * form. Used for toPrecision() and toFixed().
44368. [body formals], formals is comma separated list that needs to be parsed
44369. bytecode execution
44370. thisArg
44371. * Not found
44372. XXX: duk_uarridx_t?
44373. E5 Section 9.4, ToInteger()
44374. force_new
44375. duk_js_call.c is required to restore the stack reserve * so we only need to reset the top.
44376. * __FILE__ / __LINE__ entry, here 'pc' is line number directly. * Sometimes __FILE__ / __LINE__ is reported as the source for * the error (fileName, lineNumber), sometimes not.
44377. avoid dereference after potential callstack realloc
44378. * Pass 2
44379. compensate for eval() call
44380. set values into ret array
44381. * Pick an object from the head (don't remove yet).
44382. fast path
44383. **comment:** Fast exit if indices are identical. This is valid for a non-existent property, * for an undefined value, and almost always for ToString() coerced comparison of * arbitrary values (corner cases where this is not the case include e.g. a an * object with varying ToString() coercion). ** The specification does not prohibit "caching" of values read from the array, so * assuming equality for comparing an index with itself falls into the category of * "caching". ** Also, compareFn may be inconsistent, so skipping a call to compareFn here may * have an effect on the final result. The specification does not require any * specific behavior for inconsistent compare functions, so again, this fast path * is OK.
 label: code-design
44384. **comment:** XXX: this could be a DUK__CONSTP instead
 label: code-design
44385. [... this tracedata sep this]
44386. * After arguments, allocate special registers (like shuffling temps)
44387. [ToObject(this) item1 ... itemN arr item(i)]
44388. The implementation for computing of start_pos and end_pos differs * from the standard algorithm, but is intended to result in the exactly * same behavior. This is not always obvious.
44389. This behavior mostly mimics Node.js now.
44390. **comment:** XXX: typing (duk_hcompiledfunction has duk_uint32_t)
 label: code-design
44391. **comment:** XXX: because we're dealing with 'own' properties of a fresh array, * the array initializer should just ensure that the array has a large * enough array part and write the values directly into array part, * and finally set 'length' manually in the end (as already happens now).
 label: code-design
44392. This may happen even after the fast path check, if exponent is * not balanced (e.g. "0e1"). Remember to respect zero sign.
44393. Encode a fastint from duk_tval ptr, no value stack effects.
44394. Already declared, update value.
44395. 'has'
44396. DUK_TOK_COMMA
44397. 'void'
44398. bufwriter for code
44399. * Abandon array failed, need to decref keys already inserted * into the beginning of new_e_k before unwinding valstack.
44400. * UTF-8 / XUTF-8 / CESU-8 constants
44401. when going backwards, we decrement cpos 'early'; * 'p' may point to a continuation byte of the char * at offset 'cpos', but that's OK because we'll * backtrack all the way to the initial byte.
44402. DUK_TOK_IDENTIFIER
44403. * Regexp recursive matching function. ** Returns 'sp' on successful match (points to character after last matched one), * NULL otherwise. ** The C recursion depth limit check is only performed in this function, this * suffices because the function is present in all true recursion required by * regexp execution.
44404. **comment:** Use recursion_limit to ensure we don't overwrite js_ctx->visiting[] * array so we don't need two counter checks in the fast path. The * slow path has a much larger recursion limit which we'll use if * necessary.
 label: code-design
44405. [... func varmap enum key value this]
44406. 1: toDateString
44407. **comment:** Fill offset handling is more lenient than in Node.js.
 label: code-design
44408. Positive if local time ahead of UTC.
44409. With this check in place fast paths won't need read-only * object checks. This is technically incorrect if there are * setters that cause no writes to ROM objects, but current * built-ins don't have such setters.

44410. save stack top
44411. Insert an empty jump in the middle of code emitted earlier. This is * currently needed for compiling for-in.
44412. Treat like a debugger statement: ignore when not attached.
44413. 'import'
44414. Duktape/C API guaranteed entries (on top of args)
44415. avoid warning (unsigned)
44416. Helper which can be called both directly and with duk_safe_call().
44417. [... template]
44418. -> [buffer]
44419. **comment:** XXX: Here a "slice copy" would be useful.
 label: code-design
44420. Must ensure result is 64-bit (no overflow); a * simple and sufficient fast path is to allow only * 32-bit inputs. Avoid zero inputs to avoid * negative zero issues (-1 * 0 = -0, for instance).
44421. duk_unicode_ids_m_let_noa[]
44422. Don't intercept a DoubleError, we may have caused the initial double * fault and attempting to intercept it will cause us to be called * recursively and exhaust the C stack.
44423. **comment:** * Node.js Buffer.prototype.equals() * Node.js Buffer.prototype.compare() * Node.js Buffer.compare()
 label: requirement
44424. indirect allocs
44425. Lightfuncs are always considered strict.
44426. implies DUK_HOBJECT_IS_BUFFEROBJECT
44427. safe, because matched (NUL causes a break)
44428. * Packed tval sanity
44429. longjmp type
44430. should be first on 64-bit platforms
44431. **comment:** * XXX: duk_uint_fast32_t should probably be used in many places here.
 label: code-design
44432. [... | (crud) errobj]
44433. expensive flag
44434. validation performed by duk_hexval
44435. non-strict equality from here on
44436. allocate catcher and populate it (should be atomic)
44437. since duk_abandon_array_checked() causes a resize, there should be no gaps in keys
44438. **comment:** DUK_TVAL_SET_TVAL_UPDREF() is used a lot in executor, property lookups, * etc, so it's very important for performance. Measure when changing. ** NOTE: the source and destination duk_tval pointers may be the same, and * the macros MUST deal with that correctly.
 label: code-design
44439. if 0, 'str16' used, if > 0, 'strlist16' used
44440. probe sequence (open addressing)
44441. token was preceded by a lineterm
44442. -> [...]
44443. * Store entry state.
44444. Perform an intermediate join when this many elements have been pushed * on the value stack.
44445. duk_unicode_ids_m_let_noabmp[]
44446. * Parse variant 3 or 4. ** For variant 3 (e.g. "for (A in C) D;") the code for A (except the * final property/variable write) has already been emitted. The first * instruction of that code is at pc_v34_lhs; a JUMP needs to be inserted * there to satisfy control flow needs. ** For variant 4, if the variable declaration had an initializer * (e.g. "for (var A = B in C) D;") the code for the assignment * (B) has already been emitted. ** Variables set before entering here: * * pc_v34_lhs: insert a "JUMP L2" here (see doc/compiler.rst example). * reg_temps + 0: iteration target value (written to LHS) * reg_temps + 1: enumerator object
44447. return'
44448. **comment:** * String table algorithm: fixed size string table with array chaining ** The top level string table has a fixed size, with each slot holding * either NULL, string pointer, or pointer to a separately allocated * string pointer list. ** This is good for low memory environments using a pool allocator: the * top level allocation has a fixed size and the pointer lists have quite * small allocation size, which further matches the typical pool sizes * needed by objects, strings, property tables, etc.
 label: code-design
44449. **comment:** XXX: remove this native function and map 'stack' accessor * to the toString() implementation directly.
 label: code-design
44450. * Create a RegExp instance (E5 Section 15.10.7). ** Note: the output stack left by duk_regexp_compile() is directly compatible * with the input here. ** Input stack: [escaped_source bytecode] (both as strings) * Output stack: [RegExp]
44451. **comment:** log level could be popped but that's not necessary
 label: code-design
44452. thread currently running (only one at a time)
44453. numargs
44454. already defined, good
44455. * Currently only allowed only if yielding thread has only * EcmaScript activations (except for the Duktape.Thread.yield() * call at the callstack top) and none of them constructor * calls. ** This excludes the 'entry' thread which will always have * a preventcount > 0.
44456. Copy term name until end or '/'.
44457. -> [... trap handler]
44458. Non-BMP characters within valid UTF-8 range: encode as is. * They'll decode back into surrogate pairs if the escaped * output is decoded.
44459. [... v1 v2 name filename str] -> [... str v2 name filename]
44460. DUK_TOK_DEBUGGER
44461. Init newly allocated slots (only).
44462. a complex (new) atom taints the result
44463. [this value]
44464. See: test-bug-tailcall-preventyield-assert.c.
44465. check_object_coercible
44466. **comment:** XXX: source code property
 label: code-design
44467. **comment:** XXX: Would be nice to share the fast path loop from duk_hex_decode() * and avoid creating a temporary buffer. However, there are some * differences which prevent trivial sharing: ** - Pipe char detection * - EOF detection * - Unknown length of input and output * * The best approach here would be a bufwriter and a reasonably sized * safe inner loop (e.g. 64 output bytes at a time).
 label: code-design
44468. **comment:** XXX: unnecessary, handle in adjust
 label: code-design
44469. [... key trap handler]
44470. **comment:** * XXX: As noted above, a protected API call won't be counted as a * catcher. This is usually convenient, e.g. in the case of a top- * level duk_pcall(), but may not always be desirable. Perhaps add an * argument to treat them as catchers?
 label: code-design
44471. const
44472. even for zero-length string

44473. eat 'if'
44474. 14: getDay
44475. Custom behavior: plain buffer is used as internal buffer * without making a copy (matches Duktape.Buffer).
44476. function: create an arguments object on function call
44477. -> [... target]
44478. * Nice debug log.
44479. **comment:** XXX: copy flags using a mask
 label: code-design
44480. [... func/retval] -> [...]
44481. **comment:** XXX: optimize loops
 label: code-design
44482. up to 36 bit codepoints
44483. no fractions in internal time
44484. ditto
44485. [key] -> []
44486. Track number of escapes; count not really needed but directive * prologues need to detect whether there were any escapes or line * continuations or not.
44487. [buf? res]
44488. **comment:** Duktape 0.11.0 and prior tried to optimize the resize by not * counting the number of actually used keys prior to the resize. * This worked mostly well but also caused weird leak-like behavior * as in: test-bug-object-prop-alloc-unbounded.js. So, now we count * the keys explicitly to compute the new entry part size.
 label: code-design
44489. undefined -> skip (replaced with empty)
44490. eat opening quote on first loop
44491. **comment:** * Array built-ins ** Note that most Array built-ins are intentionally generic and work even * when the 'this' binding is not an Array instance. To ensure this, * Array algorithms do not assume "magical" Array behavior for the "length" * property, for instance. ** XXX: the "Throw" flag should be set for (almost?) all [[Put]] and * [[Delete]] operations, but it's currently false throughout. Go through * all put/delete cases and check throw flag use. Need a new API primitive * which allows throws flag to be specified. ** XXX: array lengths above 2G won't work reliably. There are many places * where one needs a full signed 32-bit range ([-0xffffffff, 0xffffffff], * i.e. -33- bits). Although array 'length' cannot be written to be outside * the unsigned 32-bit range (E5.1 Section 15.4.5.1 throws a RangeError if so) * some intermediate values may be above 0xffffffff and this may not be always * correctly handled now (duk_uint32_t is not enough for all algorithms). ** For instance, push() can legitimately write entries beyond length 0xffffffff * and cause a RangeError only at the end. To do this properly, the current * push() implementation tracks the array index using a 'double' instead of a * duk_uint32_t (which is somewhat awkward). See test-bi-array-push-maxlen.js. ** On using "put" vs. "def" prop * ===== * Code below must be careful to use the appropriate primitive as it matters * for compliance. When using "put" there may be inherited properties in * Array.prototype which cause side effects when values are written. When * using "define" there are no such side effects, and many test262 test cases * check for this (for real world code, such side effects are very rare). * Both "put" and "define" are used in the E5.1 specification; as a rule, * "put" is used when modifying an existing array (or a non-array 'this' * binding) and "define" for setting values into a fresh result array. ** Also note that Array instance 'length' should be writable, but not * enumerable and definitely not configurable: even Duktape code internally * assumes that an Array instance will always have a 'length' property. * Preventing deletion of the property is critical.
 label: code-design
44492. no need to decref
44493. 18: getMinutes
44494. 'exports'
44495. * Unicode letter check.
44496. DUK_USE_MATH_BUILTIN
44497. No field needed when strings are in ROM.
44498. value was undefined/unsupported
44499. eat 'function'
44500. keep in valstack
44501. * Must guarantee all actually used array entries will fit into * new entry part. Add one growth step to ensure we don't run out * of space right away.
44502. step 11
44503. IdentifierPart production with IdentifierStart and ASCII excluded
44504. Internal timetvalue is already NaN, so don't touch it.
44505. flag for later write
44506. GH-303
44507. Popping one element is called so often that when footprint is not an issue, * compile a specialized function for it.
44508. DUK_TOK_SEMICOLON
44509. return value from Duktape.Thread.yield()
44510. No need to re-lookup 'act' at present: no side effects.
44511. Causes a ToNumber() coercion, but doesn't break coercion order since * year is coerced first anyway.
44512. Parts are in local time, convert when setting.
44513. * E5 Section 7.4. If the multi-line comment contains a newline, * it is treated like a single line terminator for automatic * semicolon insertion.
44514. Note: if target_pc1 == i, we'll optimize a jump to itself. * This does not need to be checked for explicitly; the case * is rare and max iter breaks us out.
44515. Backup 'caller' property and update its value.
44516. assume exactly 1 arg, which is why ** is forbidden; arg size still * depends on type though.
44517. Add function object.
44518. finalization
44519. [ToObject(this) item1 ... itemN arr]
44520. 0x50...0x5f
44521. Rule table: first matching rule is used to determine what to do next.
44522. CATCH flag may be enabled or disabled here; it may be enabled if * the statement has a catch block but the try block does not throw * an error.
44523. * Init the heap object
44524. Inherit from ROM-based global object: less RAM usage, less transparent.
44525. Note: 0xff != DUK_BC_C_MAX
44526. COMMA
44527. Node.js return value for noAssert out-of-bounds reads is * usually (but not always) NaN. Return NaN consistently.
44528. **comment:** function must not be tail called
 label: code-design
44529. stage 3: update length (done by caller), decide return code
44530. idx_space
44531. statement does not terminate directive prologue
44532. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined fresh_require exports module mod_func exports fresh_require exports module]
44533. **comment:** XXX: optimize for buffer inputs: no need to coerce to a string * which causes an unnecessary interning.
 label: code-design
44534. * Table for hex decoding ASCII hex digits
44535. function: create new environment when called (see duk_hcompiledfunction)
44536. pop regexp res_obj or match string
44537. P
44538. required if NAMEBINDING set

44539. slow path decode
44540. don't care value, year is mandatory
44541. really 'not applicable' anymore, should not be referenced after this
44542. skip jump conditionally
44543. * Flags parsing (see E5 Section 15.10.4.1).
44544. for updating (all are set to < 0 for virtual properties)
44545. * Run (object) finalizers in the "to be finalized" work list.
44546. 'warn'
44547. DUK_USE_FILE_IO
44548. [... constructor arg1 ... argN final_cons fallback]
44549. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur.
44550. 'Array' object, array length and index exotic behavior
44551. a statement following a label cannot be a source element * (a function declaration).
44552. 24: setMilliseconds
44553. '\xffValue'
44554. **comment:** DUK_USE_DATE_GET_LOCAL_TZOFFSET() needs to be called with a * time value computed from UTC parts. At this point we only * have 'd' which is a time value computed from local parts, so * it is off by the UTC-to-local time offset which we don't know * yet. The current solution for computing the UTC-to-local * time offset is to iterate a few times and detect a fixed * point or a two-cycle loop (or a sanity iteration limit), * see test-bi-date-local-parts.js and test-bi-date-tzoffset-basic-fi.js. * * E5.1 Section 15.9.1.9: * UTC(t) = t - LocalTZA - DaylightSavingTA(t - LocalTZA) * * For NaN/inf, DUK_USE_DATE_GET_LOCAL_TZOFFSET() returns 0.
label: code-design
44555. DUK_TOK_BREAK
44556. We're conceptually between two opcodes; act->pc indicates the next * instruction to be executed. This is usually the correct pc/line to * indicate in Status. (For the 'debugger' statement this now reports * the pc/line after the debugger statement because the debugger opcode * has already been executed.)
44557. DUK_TOK_MOD_EQ
44558. **comment:** * Write to 'length' of an array is a very complex case * handled in a helper which updates both the array elements * and writes the new 'length'. The write may result in an * unconditional RangeError or a partial write (indicated * by a return code). * * Note: the helper has an unnecessary writability check * for 'length', we already know it is writable.
label: code-design
44559. **comment:** * Allocation size for 'curr_alloc' is alloc_size. There is no * automatic NUL terminator for buffers (see above for rationale). * * 'curr_alloc' is explicitly allocated with heap allocation * primitives and will thus always have alignment suitable for * e.g. duk_tval and an IEEE double.
label: code-design
44560. * ctx->prev_token token to process with duk__expr_led() * ctx->curr_token updated by caller
44561. Note: target registers a and a+1 may overlap with DUK_REGP(b). * Careful here.
44562. * Mark unreachable, finalizable objects. * * Such objects will be moved aside and their finalizers run later. They have * to be treated as reachability roots for their properties etc to remain * allocated. This marking is only done for unreachable values which would * be swept later (refzero_list is thus excluded). * * Objects are first marked FINALIZABLE and only then marked as reachability * roots; otherwise circular references might be handled inconsistently.
44563. **comment:** XXX: accept any duk_hbufferobject type as an input also?
label: code-design
44564. Note: DST adjustment is determined using UTC time.
44565. caller must check
44566. 2
44567. **comment:** Fresh require: require.id is left configurable (but not writable) * so that is not easy to accidentally tweak it, but it can still be * done with Object.defineProperty(). * * XXX: require.id could also be just made non-configurable, as there * is no practical reason to touch it.
label: code-design
44568. log this with a normal debug level because this should be relatively rare
44569. * Ecmascript execution, support primitives.
44570. * replace()
44571. out_clamped=NULL, RangeError if outside range
44572. DUK_USE_MARK_AND_SWEEP && DUK_USE_VOLUNTARY_GC
44573. idx_args = idx_func + 2
44574. handle comma and closing bracket
44575. * Avoid a GC if GC is already running. See duk_heap_mem_alloc().
44576. Note: this allows creation of internal strings.
44577. DUK_FLD_DOUBLE
44578. **comment:** * Logging * * Current logging primitive is a sprintf-style log which is convenient * for most C code. Another useful primitive would be to log N arguments * from value stack (like the Ecmascript binding does).
label: code-design
44579. caller and arguments must use the same thrower, [[ThrowTypeError]]
44580. bound function 'length' property is interesting
44581. DUK_REPLACEMENTS_H_INCLUDED
44582. [... error func]
44583. expt 0x000 is zero/subnormal
44584. There is a caller; it MUST be an Ecmascript caller (otherwise it would * match entry level check)
44585. restore PC
44586. allow negative PCs, behave as a no-op
44587. '\xffValue'
44588. * Activation defines
44589. Step 15: insert itemCount elements into the hole made above
44590. No need to replace the 'enum_target' value in stack, only the * enum_target reference. This also ensures that the original * enum target is reachable, which keeps the proxy and the proxy * target reachable. We do need to replace the internal _Target.
44591. XXX: ARRAY_PART for Array prototype?
44592. Coerce value to a number before computing check_length, so that * the field type specific coercion below can't have side effects * that would invalidate check_length.
44593. string is a valid array index
44594. * Fast path: assume no mutation, iterate object property tables * directly; bail out if that assumption doesn't hold.
44595. 'Number'
44596. buffer is behind a pointer, dynamic or external
44597. * in
44598. allow caller to give a const number with the DUK_CONST_MARKER
44599. DUK_USE_JSON_DECNUMBER_FASTPATH
44600. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack (e.g. if * lval is already a string).
44601. XXX: use DUK_HSTRING_FLAG_INTERNAL?
44602. func support for [[HasInstance]] checked in the beginning of the loop
44603. %p
44604. weak refs should be handled here, but no weak refs for * any non-string objects exist right now.
44605. pop key

44606. implementation specific
44607. * Log level check
44608. **comment:** XXX: unify handling with native call.
label: code-design
44609. internal extra elements assumed on function entry, * always added to user-defined 'extra' for e.g. the * duk_check_stack() call.
44610. * Complex atom * * The original code is used as a template, and removed at the end * (this differs from the handling of simple quantifiers). * * NOTE: there is no current solution for empty atoms in complex * quantifiers. This would need some sort of a 'progress' instruction. * * XXX: impose limit on maximum result size, i.e. atom_code_len * atom_copies?
44611. Leading zero is not counted towards precision digits; not * in the integer part, nor in the fraction part.
44612. 39: setYear
44613. **comment:** * ToString() (E5 Section 9.8) * * ==> implemented in the API.
label: requirement
44614. E5.1 Section 15.9.1.6
44615. DUK_USE_JSON_QUOTESTRING_FASTPATH
44616. always in executor
44617. because of valstack init policy
44618. embed: func ptr
44619. local
44620. Sanity check
44621. * Object environment record. * * Binding (target) object is an external, uncontrolled object. * Identifier may be bound in an ancestor property, and may be * an accessor. Target can also be a Proxy which we must support * here.
44622. * Manipulate callstack for the call.
44623. Currently the bytecode executor and executor interrupt * instruction counts are off because we don't execute the * interrupt handler when we're about to exit from the initial * user call into Duktape. * * If we were to execute the interrupt handler here, the counts * would match. You can enable this block manually to check * that this is the case.
44624. **comment:** Note: combining comparison ops must be done carefully because * of incomparable values (NaN): it's not necessarily true that * $(x \geq y) == !(x < y)$. Also, evaluation order matters, and * although it would only seem to affect the compiler this is * actually not the case, because there are also run-time coercions * of the arguments (with potential side effects). * * XXX: can be combined; check code size.
label: code-design
44625. 'typeof'
44626. Execute bytecode until returned or longjmp().
44627. overflow not possible, buffer limits
44628. 'interface'
44629. zero everything unless requested not to do so
44630. E5 Section 15.12.3, main algorithm, step 4.b.ii steps 1-4.
44631. * Node.js Buffer.isBuffer()
44632. DUK_TOK_COLON
44633. B -> target register for next key * C -> enum register
44634. -> [... flags escaped_source bytecode]
44635. backref n -> saved indices [n*2, n*2+1]
44636. value1 -> error object
44637. remove key
44638. **comment:** * String hash computation (interning). * * String hashing is performance critical because a string hash is computed * for all new strings which are candidates to be added to the string table. * However, strings actually added to the string table go through a codepoint * length calculation which dominates performance because it goes through * every byte of the input string (but only for strings added). * * The string hash algorithm should be fast, but on the other hand provide * good enough hashes to ensure both string table and object property table * hash tables work reasonably well (i.e., there aren't too many collisions * with real world inputs). Unless the hash is cryptographic, it's always * possible to craft inputs with maximal hash collisions. * * NOTE: The hash algorithms must match src/dukutil.py:duk_heap_hashstring() * for ROM string support!
label: code-design
44639. **comment:** XXX: should the API call handle this directly, i.e. attempt * to duk_push_hobject(ctx, null) would push a null instead? * (On the other hand 'undefined' would be just as logical, but * not wanted here.)
label: code-design
44640. **comment:** For short strings, use a fixed temp buffer.
label: code-design
44641. label limits
44642. marker values for hash part
44643. DUK_TOK_GE
44644. -> [... closure]
44645. NULL pointer is required to encode to zero, so memset is enough.
44646. else an array initializer element
44647. * E5 Section 7.3 * * A LineTerminatorSequence essentially merges <CR> <LF> sequences * into a single line terminator. This must be handled by the caller.
44648. 35: setUTCMonth
44649. [... retval]
44650. Input shuffling happens before the actual operation, while output * shuffling happens afterwards. Output shuffling decisions are still * made at the same time to reduce branch clutter; output shuffle decisions * are recorded into X_out variables.
44651. 'new' MemberExpression Arguments
44652. assume plain values
44653. duk_regconst_t is unsigned, so use 0 as dummy value (ignored by caller)
44654. [... val obj]
44655. -> [... re_obj input bc saved_buf lastIndex]
44656. **comment:** (advance << 8) + token_type, updated at function end, * init is unnecessary but suppresses "may be used uninitialized" warnings.
label: code-design
44657. first, parse regexp body roughly
44658. Duktape/C (nativefunction) object, exotic 'length'
44659. DUK_HOBJECT_FLAG_EXOTIC_ARRAY varies
44660. 'this' binding
44661. [... val key] -> [... key val]
44662. isAccessor
44663. Cannot be compressed as a heap pointer because may point to * an arbitrary address.
44664. skip opening slash on first loop
44665. transfer flags
44666. DUK_TOK_INTERFACE
44667. may be undefined
44668. accept anything, expect first value (EOF will be * caught by duk_dec_value() below.
44669. iteration statements allow continue
44670. Clamp so that values at 'clamp_top' and above are wiped and won't * retain reachable garbage. Then extend to 'nregs' because we're * returning to an EcmaScript function.
44671. 'while'

44672. **comment:** Checking this here rather than in memory alloc primitives * reduces checking code there but means a failed allocation * will go through a few retries before giving up. That's * fine because this only happens during debugging.
label: code-design

44673. **comment:** * Array abandon check; abandon if: * * new_used / new_size < limit * new_used < limit * new_size || limit is 3 bits fixed point * new_used < limit' / 8 * new_size || *8 * 8*new_used < limit' * new_size || :8 * new_used < limit' * (new_size / 8) * * Here, new_used = a_used, new_size = a_size. * * Note: some callers use approximate values for a_used and/or a_size * (e.g. dropping a '+1' term). This doesn't affect the usefulness * of the check, but may confuse debugging.
label: code-design

44674. Target is before source. Source offset is expressed as * a "before change" offset. Account for the memmove.

44675. object is constructable

44676. regexp support disabled

44677. This should be equivalent to match() algorithm step 8.f.iii.2: * detect an empty match and allow it, but don't allow it twice.

44678. assume keys are compacted

44679. currently implicitly also DUK_USE_DOUBLE_LINKED_HEAP

44680. advance, whatever the current token is; parse next token in regexp context

44681. *ToInt32(), ToUint32(), ToUint16() (E5 Sections 9.5, 9.6, 9.7)

44682. rego to allocate

44683. preallocated temporaries (2) for variants 3 and 4

44684. [regexp input]

44685. function: function object is strict

44686. DUK_TOK_ALSHIFT

44687. **comment:** XXX: what to do if _Formals is not empty but compiler has * optimized it away -- read length from an explicit property * then?
label: code-design

44688. work list for objects whose refcounts are zero but which have not been * "finalized"; avoids recursive C calls when refcounts go to zero in a * chain of objects.

44689. **comment:** context and locale specific rules which cannot currently be represented * in the caseconv bitstream: hardcoded rules in C
label: code-design

44690. * Heap string representation. * * Strings are byte sequences ordinarily stored in extended UTF-8 format, * allowing values larger than the official UTF-8 range (used internally) * and also allowing UTF-8 encoding of surrogate pairs (CESU-8 format). * Strings may also be invalid UTF-8 altogether which is the case e.g. with * strings used as internal property names and raw buffers converted to * strings. In such cases the 'clen' field contains an inaccurate value. * * Ecmascript requires support for 32-bit long strings. However, since each * 16-bit codepoint can take 3 bytes in CESU-8, this representation can only * support about 1.4G codepoint long strings in extreme cases. This is not * really a practical issue.

44691. call is a direct eval call

44692. varname is still reachable

44693. Leave 'setter' on stack

44694. seconds

44695. **comment:** * Ecmascript compiler. * * Parses an input string and generates a function template result. * Compilation may happen in multiple contexts (global code, eval * code, function code). * * The parser uses a traditional top-down recursive parsing for the * statement level, and an operator precedence based top-down approach * for the expression level. The attempt is to minimize the C stack * depth. Bytecode is generated directly without an intermediate * representation (tree), at the cost of needing two passes over each * function. * * The top-down recursive parser functions are named "duk_parse_XXX". * * Recursion limits are in key functions to prevent arbitrary C recursion: * function body parsing, statement parsing, and expression parsing. * * See doc/compiler.rst for discussion on the design. * * A few typing notes: * * - duk_regconst_t: unsigned, no marker value for "none" * - duk_reg_t: signed, < 0 = none * - PC values: duk_int_t, negative values used as markers
label: code-design

44696. Append a "(line NNN)" to the "message" property of any error * thrown during compilation. Usually compilation errors are * SyntaxErrors but they can also be out-of-memory errors and * the like.

44697. **comment:** XXX: any way to detect faster whether something needs to be closed? * We now look up _Callee and then skip the rest.
label: requirement

44698. DUK_USE_VARIADIC_MACROS

44699. * seeeeeeee eeeeefffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H * * s sign bit * eee... exponent field * fff... fraction * * ieee value = 1.ffff... * 2^(e - 1023) (normal) * = 0.ffff... * 2^(-1022) (denormal) * * algorithm v = f * b^e

44700. key is now reachable in the valstack

44701. MakeDate

44702. Request callback should push values for reply to client onto valstack

44703. inserted jump

44704. [enum_target res]

44705. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize or a forced relocating realloc?
label: code-design

44706. For Ecmascript strings, this check can only match for * initial UTF-8 bytes (not continuation bytes). For other * strings all bets are off.

44707. Then reuse the unwound activation; callstack was not shrunk so there is always space

44708. DUK_USE_DEBUGGER_DUMPHEAP || DUK_USE_DEBUGGER_INSPECT

44709. for storing/restoring the varmap binding for catch variable

44710. r <- (2 * f) * b^(e+1)

44711. [...]

44712. **comment:** Careful here and with other duk_put_prop_xxx() helpers: the * target object and the property value may be in the same value * stack slot (unusual, but still conceptually clear).
label: code-design

44713. function is strict

44714. must work for nargs <= 0

44715. Ignore digits beyond a radix-specific limit, but note them * in expt_adj.

44716. don't run finalizers; leave finalizable objects in finalize_list for next round

44717. [this value] -> [value this]

44718. Update 'buffer_length' to be the effective, safe limit which * takes into account the underlying buffer. This value will be * potentially invalidated by any side effect.

44719. 2: toTimeString

44720. **comment:** No need for constants pointer (= same as data). * * When using 16-bit packing alignment to 4 is nice. 'funcs' will be * 4-byte aligned because 'constants' are duk_tvals. For now the * inner function pointers are not compressed, so that 'bytecode' will * also be 4-byte aligned.
label: code-design

44721. * Initialize function state for a zero-argument function

44722. Guaranteed by recursion_limit setup so we don't have to * check twice.

44723. line continuation

44724. IsGenericDescriptor(desc) == true; this means in practice that 'desc' * only has [[Enumerable]] or [[Configurable]] flag updates, which are * allowed at this point.

44725. '-Infinity'

44726. * Resume a thread. * * The thread must be in resumable state, either (a) new thread which hasn't * yet started, or (b) a thread which has previously yielded. This method * must be called from an Ecmascript function. * * Args: * - thread * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.

44727. type tag (public)

44728. **comment:** It'd be nice to do something like this - but it doesn't * work for closures created inside the catch clause.
label: code-design

44729. don't re-enter e.g. during Eval

44730. mp <- b^e

44731. special handling of fmt==NULL
44732. Ensure space for final configuration (idx_retbase + num_stack_rets) * and intermediate configurations.
44733. parse value
44734. **comment:** Note: not honoring round-to-even should work but now generates incorrect * results. For instance, 1e23 serializes to "a000...", i.e. the first digit * equals the radix (10). Scaling stops one step too early in this case. * Don't know why this is the case, but since this code path is unused, it * doesn't matter.
label: code-design
44735. **comment:** The 'strict' flag is copied to get the special [[Get]] of E5.1 * Section 15.3.5.4 to apply when a 'caller' value is a strict bound * function. Not sure if this is correct, because the specification * is a bit ambiguous on this point but it would make sense.
label: code-design
44736. function may call direct eval
44737. **comment:** _Varmap: omitted if function is guaranteed not to do slow path identifier * accesses or if it would turn out to be empty of actual register mappings * after a cleanup. When debugging is enabled, we always need the varmap to * be able to lookup variables at any point.
label: code-design
44738. [array totalLength]
44739. Opcode interval for a Date-based status/peek rate limit check. Only * relevant when debugger is attached. Requesting a timestamp may be a * slow operation on some platforms so this shouldn't be too low. On the * other hand a high value makes Duktape react to a pause request slowly.
44740. -> [... key val val]
44741. [... re_obj input] -> [... re_obj input bc]
44742. valstack slot for 1st token value
44743. side effects, possibly errors
44744. NOTE: Multiple evaluation of 'ptr' in this macro.
44745. Overflow of the execution counter is fine and doesn't break * anything here.
44746. mark-and-sweep: finalized (on previous pass)
44747. curr is accessor, desc is data
44748. [func thisArg arg1 ... argN]
44749. XXX: the insert helpers should ensure that the bytecode result is not * larger than expected (or at least assert for it). Many things in the * bytecode, like skip offsets, won't work correctly if the bytecode is * larger than say 2G.
44750. this function
44751. **comment:** 'tv' becomes invalid
label: code-design
44752. is an actual function (not global/eval code)
44753. Catch attempts to use out-of-range reg/const. Without this * check Duktape 0.12.0 could generate invalid code which caused * an assert failure on execution. This error is triggered e.g. * for functions with a lot of constants and a try-catch statement. * Shuffling or opcode semantics change is needed to fix the issue. * See: test-bug-trycatch-many-constants.js.
44754. 'callee'
44755. Initial stack size satisfies the stack spare constraints so there * is no need to require stack here.
44756. 'byteOffset'
44757. **comment:** duk_handle_call_xxx: constructor call (i.e. called as 'new Foo()')
label: code-design
44758. no operators
44759. Strict: never allow function declarations outside top level.
44760. [... arg1 ... argN envobj]
44761. * Field access
44762. 0x30...0x3f
44763. value1 -> label number, pseudo-type to indicate a continue continuation (for ENDFIN)
44764. true
44765. **comment:** XXX: common reg allocation need is to reuse a sub-expression's temp reg, * but only if it really is a temp. Nothing fancy here now.
label: code-design
44766. only use the highest bit
44767. Function values are handled completely above (including * coercion results):
44768. distinct bignums, easy mistake to make
44769. **comment:** XXX: this is now a very unoptimal implementation -- this can be * made very simple by direct manipulation of the object internals, * given the guarantees above.
label: code-design
44770. Split time value into parts. The time value is assumed to be an internal * one, i.e. finite, no fractions. Possible local time adjustment has already * been applied when reading the time value.
44771. If flags != 0 (strict or SameValue), thr can be NULL. For loose * equals comparison it must be != NULL.
44772. DUK_TOK_INSTANCEOF
44773. time
44774. update entry, allocating if necessary
44775. A structure for 'snapshotting' a point for rewinding
44776. * Ecmascript call
44777. cannot be arguments exotic
44778. **comment:** this is not strictly necessary, but helps debugging
label: code-design
44779. * Prototypes for built-in functions not automatically covered by the * header declarations emitted by genbuiltins.py.
44780. push initial function call to new thread stack; this is * picked up by resume().
44781. 31: setUTCHours
44782. Decode a one-bit flag, and if set, decode a value of 'bits', otherwise return * default value. Return value is signed so that negative marker value can be * used by caller as a "not present" value.
44783. format plus something to avoid just missing
44784. **comment:** XXX: post-checks (such as no duplicate keys)
label: code-design
44785. if property already exists, overwrites silently
44786. **comment:** 'val' is never NaN, so no need to normalize
label: code-design
44787. * Lowercase digits for radix values 2 to 36. Also doubles as lowercase * hex nybble table.
44788. may be reg or const
44789. decl name
44790. rnd_state for duk_util_tinyrandom.c
44791. Note: heap->heap_thread, heap->curr_thread, and heap->heap_object * are on the heap allocated list.
44792. jump is inserted here
44793. misc
44794. 'length' and other virtual properties are not * enumerable, but are included if non-enumerable * properties are requested.
44795. only flag now
44796. [... func this arg1 ... argN]
44797. record previous atom info in case next token is a quantifier
44798. -"-

44799. * Ecmascript compliant [[GetOwnProperty]](P), for internal use only. ** If property is found: * - Fills descriptor fields to 'out_desc' * - If DUK_GETDESC_FLAG_PUSH_VALUE is set, pushes a value related to the * property onto the stack ('undefined' for accessor properties). * - Returns non-zero * * If property is not found: * - 'out_desc' is left in untouched state (possibly garbage) * - Nothing is pushed onto the stack (not even with DUK_GETDESC_FLAG_PUSH_VALUE * set) * - Returns zero ** Notes: ** - Getting a property descriptor may cause an allocation (and hence * GC) to take place, hence reachability and refcount of all related * values matter. Reallocation of value stack, properties, etc may * invalidate many duk_tval pointers (concretely, those which reside * in memory areas subject to reallocation). However, heap object * pointers are never affected (heap objects have stable pointers). * * - The value of a plain property is always reachable and has a non-zero * reference count. ** - The value of a virtual property is not necessarily reachable from * elsewhere and may have a refcount of zero. Hence we push it onto * the valstack for the caller, which ensures it remains reachable * while it is needed. ** - There are no virtual accessor properties. Hence, all getters and * setters are always related to concretely stored properties, which * ensures that the get/set functions in the resulting descriptor are * reachable and have non-zero refcounts. Should there be virtual * accessor properties later, this would need to change.

44800. not enumerable

44801. %f and %lf both consume a 'long'

44802. * Catcher defines

44803. never executed

44804. **comment:** At least 'magic' has a significant impact on function * identity.

label: code-design

44805. 'c'

44806. important to do this *after* pushing, to make the thread reachable for gc

44807. [offset fieldByteLength noAssert], when ftype == DUK__FLD_VARINT

44808. Should not happen.

44809. init pointer fields to null

44810. * toString(), valueOf()

44811. success, fixup pointers

44812. **comment:** XXX: several pointer comparison issues here

label: code-design

44813. insert (DUK__REOP_WIPERANGE, start, count) in reverse order so the order ends up right

44814. Check that value is a duk_hbufferobject and return a pointer to it.

44815. Note: DUK_VALSTACK_INITIAL_SIZE must be >= DUK_VALSTACK_API_ENTRY_MINIMUM * + DUK_VALSTACK_INTERNAL_EXTRA so that the initial stack conforms to spare * requirements.

44816. **comment:** XXX: may need a 'length' filter for forEach()

label: code-design

44817. Note: assumes that duk_util_probe_steps size is 32

44818. guaranteed when building arguments

44819. updates thread state, minimizes its allocations

44820. '{"_inf":true}'

44821. **comment:** Without variadic macros resort to comma expression trickery to handle debug * prints. This generates a lot of harmless warnings. These hacks are not * needed normally because DUK_D() and friends will hide the entire debug log * statement from the compiler.

label: code-design

44822. **comment:** TimeClip() should never be necessary

label: code-design

44823. 0x10-0x1f

44824. E5.1 Section B.2.2, step 7.

44825. Parser part masks.

44826. end switch (tok)

44827. negative -> complex atom

44828. -> [... val root ""]

44829. result array

44830. XXX: fast path for arrays?

44831. -> [... ToObject(this) ToUint32(length)]

44832. Continue checked execution if there are breakpoints or we're stepping. * Also use checked execution if paused flag is active - it shouldn't be * because the debug message loop shouldn't terminate if it was. Step out * is handled by callstack unwind and doesn't need checked execution. * Note that debugger may have detached due to error or explicit request * above, so we must recheck attach status.

44833. 0x0...0x0f

44834. coercion order matters

44835. 'jc'

44836. 1 if property found, 0 otherwise

44837. fixed-format

44838. not strictly necessary because of lookahead '}' above

44839. **comment:** XXX: must be able to represent -len

label: code-design

44840. DUK_USE_DPRINT_COLORS

44841. expect_eof

44842. borrowed reference to catch variable name (or NULL if none)

44843. **comment:** XXX: these could be implemented as macros calling an internal function * directly. * XXX: same issue as with Duktape.fin: there's no way to delete the property * now (just set it to undefined).

label: code-design

44844. **comment:** XXX: set magic directly here? (it could share the c_nargs arg)

label: code-design

44845. * Preliminary activation record and valstack manipulation. * The concrete actions depend on whether we're dealing * with a tail call (reuse an existing activation), a resume, * or a normal call. ** The basic actions, in varying order, are: ** - Check stack size for call handling * - Grow call stack if necessary (non-tail-calls) * - Update current activation (idx_retval) if necessary * (non-tail, non-resume calls) * - Move start of args (idx_args) to valstack bottom * (tail calls) ** Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.

44846. result index for filter()

44847. either nonzero value is ok

44848. bc

44849. * DELVAR ** See E5 Sections: * 11.4.1 The delete operator * 10.2.1.1.5 DeleteBinding (N) [declarative environment record] * 10.2.1.2.5 DeleteBinding (N) [object environment record] ** Variable bindings established inside eval() are deletable (configurable), * other bindings are not, including variables declared in global level. * Registers are always non-deletable, and the deletion of other bindings * is controlled by the configurable flag. ** For strict mode code, the 'delete' operator should fail with a compile * time SyntaxError if applied to identifiers. Hence, no strict mode * run-time deletion of identifiers should ever happen. This function * should never be called from strict mode code!

44850. * Mark refzero_list objects. * * Objects on the refzero_list have no inbound references. They might have * outbound references to objects that we might free, which would invalidate * any references held by the refzero objects. A refzero object might also * be rescued by refcount finalization. Refzero objects are treated as * reachability roots to ensure they (or anything they point to) are not * freed in mark-and-sweep.

44851. Negative zero needs special handling in JX/JC because * it would otherwise serialize to '0', not '-0'.

44852. can't resize below 'top'

44853. round up roughly to next 'grow step'

44854. **comment:** * E5 Section 7.6: ** IdentifierPart: * IdentifierStart * UnicodeCombiningMark * UnicodeDigit * UnicodeConnectorPunctuation * <ZWNJ> [U+200C] * <ZWJ> [U+200D] ** IdentifierPart production has one multi-character production * as part of its IdentifierStart alternative. The '\ character * of an escape

sequence is not matched here, see discussion in * duk_unicode_is_identifier_start(). ** To match non-ASCII characters (codepoints >= 0x80), a very slow * linear range-by-range scan is used. The codepoint is first compared * to the IdentifierStart ranges, and if it doesn't match, then to a * set consisting of code points in IdentifierPart but not in * IdentifierStart. This is done to keep the unicode range data small, * at the expense of speed. ** The ASCII fast path consists of: ** 0x0030 ... 0x0039 ['0' ... '9', UnicodeDigit] * 0x0041 ... 0x005a ['A' ... 'Z', IdentifierStart] * 0x0061 ... 0x007a ['a' ... 'z', IdentifierStart] * 0x0024 ['\$', IdentifierStart] * 0x005f['_', IdentifierStart and * UnicodeConnectorPunctuation] ** UnicodeCombiningMark has no code points <= 0x7f. ** The matching code reuses the "identifier start" tables, and then * consults a separate range set for characters in "identifier part" * but not in "identifier start". These can be extracted with the * 'src/extract_chars.py' script. ** UnicodeCombiningMark -> categories Mn, Mc * UnicodeDigit -> categories Nd * UnicodeConnectorPunctuation -> categories Pc

label: code-design

44855. last array index explicitly initialized, +1

44856. Value would yield 'undefined', so skip key altogether. * Side effects have already happened.

44857. need side effects, not value

44858. * Check function name validity now that we know strictness. * This only applies to function declarations and expressions, * not setter/getter name. * * See: test-dev-strict-mode-boundary.js

44859. Target may be a Proxy or property may be an accessor, so we must * use an actual, Proxy-aware hasprop check here. * * out->holder is NOT set to the actual duk_hobject where the * property is found, but rather the object binding target object.

44860. Note: for srclen=0, src may be NULL

44861. never an empty match, so step 13.c.iii can't be triggered

44862. Outer executor with setjmp/longjmp handling.

44863. if B/C is >= this value, refers to a const

44864. if debugging disabled

44865. if true, the stack already contains the final result

44866. * E5 Section 7.2 specifies six characters specifically as * white space: * * 0009;<control>;Cc;0S;;;;N;CHARACTER TABULATION;;;; * 000B;<control>;Cc;0S;;;;N;LINE TABULATION;;;; * 000C;<control>;Cc;0;WS;;;;N;FORM FEED (FF);;; * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; * FEFF;ZERO WIDTH NO-BREAK SPACE;Cf;0;BN;;;;N;BYTE ORDER MARK;;;; * * It also specifies any Unicode category 'Zs' characters as white * space. These can be extracted with the "src/extract_chars.py" script. * Current result: * * RAW OUTPUT: * ===== * 0020;SPACE;Zs;0;WS;;;;N;;;; * 00A0;NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;NON-BREAKING SPACE;;;; * 1680;OGHAM SPACE MARK;Zs;0;WS;;;;N;;;; * 180E;MONGOLIAN VOWEL SEPARATOR;Zs;0;WS;;;;N;;;; * 2000;EN QUAD;Zs;0;WS;2002;;;;N;;;; * 2001;EM QUAD;Zs;0;WS;2003;;;;N;;;; * 2002;EN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2003;EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2004;THREE-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2005;FOUR-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2006;SIX-PER-EM SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2007;FIGURE SPACE;Zs;0;WS;<noBreak> 0020;;;;N;;;; * 2008;PUNCTUATION SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 2009;THIN SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 200A;HAIR SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 202F;NARROW NO-BREAK SPACE;Zs;0;CS;<noBreak> 0020;;;;N;;;; * 205F;MEDIUM MATHEMATICAL SPACE;Zs;0;WS;<compat> 0020;;;;N;;;; * 3000;IDEOGRAPHIC SPACE;Zs;0;WS;<wide> 0020;;;;N;;;; * * RANGES: * ===== * 0x0020 * 0x00a0 * 0x1680 * 0x180e * 0x2000 ... 0x20a0 * 0x202f * 0x205f * 0x3000 * * A manual decoder (below) is probably most compact for this.

44867. ToNumber(bool) is +1.0 or 0.0. Tagged boolean value is always 0 or 1.

44868. when nothing is running, API calls are in non-strict mode

44869. **comment:** String length is computed here to avoid multiple evaluation * of a macro argument in the calling side.

label: code-design

44870. see duk_js_executor.c

44871. Convert heap string index to a token (reserved words)

44872. **comment:** XXX: refactor and share with other code

label: code-design

44873. order must match constants in genbuiltins.py

44874. Buffer is kept as is, with the fixed/dynamic nature of the * buffer only changed if requested. An external buffer * is converted into a non-external dynamic buffer in a * duk_to_dynamic_buffer() call.

44875. * Ecmascript compliant [[Delete]](P, Throw).

44876. -> [... str]

44877. args go here in parens

44878. 0xb0...0xbf

44879. Write in big endian

44880. only need to guarantee 1 more slot, but allocation growth is in chunks

44881. **comment:** alloc function typedefs in duktape.h

label: code-design

44882. i >= 0 would always be true

44883. **comment:** XXX: inline into a prototype walking loop?

label: code-design

44884. Lightfunc handling by ToObject() coercion.

44885. DUK_BUFOBJ_UINT16ARRAY

44886. 'else'

44887. 'String' object, array index exotic behavior

44888. A -> target reg * B -> object reg/const (may be const e.g. in "foo'[1]") * C -> key reg/const

44889. There is no eval() special handling here: eval() is never * automatically converted to a lightfunc.

44890. **comment:** XXX: identify enumeration target with an object index (not top of stack)

label: code-design

44891. Write unsigned 32-bit integer.

44892. The underlying types for offset/length in duk_hbufferobject is * duk_uint_t; make sure argument values fit and that offset + length * does not wrap.

44893. [body formals source template closure]

44894. no refcount changes

44895. [arg1 ... argN-1 body] -> [body arg1 ... argN-1]

44896. First evaluate LHS fully to ensure all side effects are out.

44897. properties object

44898. Call handling and success path. Success path exit cleans * up almost all state.

44899. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer * stack[4] = regexp match OR match string

44900. object is extensible

44901. * The length comparisons are present to handle * strings like "use strict\u0000foo" as required.

44902. lookup name from an open declarative record's registers

44903. relookup (side effects)

44904. **comment:** XXX: make a common DUK_USE_ option, and allow custom fixed seed?

label: code-design

44905. x <- y * z

44906. * Note: assuming new_a_size == 0, and that entry part contains * no conflicting keys, refcounts do not need to be adjusted for * the values, as they remain exactly the same. ** The keys, however, need to be interned, incref'd, and be * reachable for GC. Any intern attempt may trigger a GC and * claim any non-reachable strings, so every key must be reachable * at all times. ** A longjmp must not occur here, as the new_p allocation would * be freed without these keys being decref'd, hence the messy * decref handling if intern fails.

44907. **comment:** * 'props' contains {key,value,flags} entries, optional array entries, and * an optional hash lookup table for non-array entries in a single 'sliced' * allocation. There are several layout options, which differ slightly in * generated code size/speed and alignment/padding; duk_features.h selects * the layout used. * * Layout 1 (DUK_USE_HOBJECT_LAYOUT_1): * * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_propvalue)

bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_uint8_t) bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xffffffffUL = unused, 0xfffffffleUL = deleted
* * Layout 2 (DUK_USE_HOBJECT_LAYOUT_2): * * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) * e_size * sizeof(duk_hstring)
*) bytes of entry keys (e_next gc reachable) * e_size * sizeof(duk_uint8_t) + pad bytes of entry flags (e_next gc reachable) * a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xfffffffleUL = unused,
0xfffffffleUL = deleted * * Layout 3 (DUK_USE_HOBJECT_LAYOUT_3): * * e_size * sizeof(duk_propvalue) bytes of entry values (e_next gc reachable) *
a_size * sizeof(duk_tval) bytes of (opt) array values (plain only) (all gc reachable) * e_size * sizeof(duk_hstring *) bytes of entry keys (e_next gc reachable) *
h_size * sizeof(duk_uint32_t) bytes of (opt) hash indexes to entries (e_size), * 0xfffffffleUL = unused, 0xfffffffleUL = deleted * e_size * sizeof(duk_uint8_t) bytes
of entry flags (e_next gc reachable) * * In layout 1, the 'e_next' count is rounded to 4 or 8 on platforms * requiring 4 or 8 byte alignment. This ensures proper
alignment * for the entries, at the cost of memory footprint. However, it's * probably preferable to use another layout on such platforms instead. * * In layout 2, the
key and value parts are swapped to avoid padding * the key array on platforms requiring alignment by 8. The flags part * is padded to get alignment for array
entries. The 'e_next' count does * not need to be rounded as in layout 1. * * In layout 3, entry values and array values are always aligned properly, * and assuming
pointers are at most 8 bytes, so are the entry keys. Hash * indices will be properly aligned (assuming pointers are at least 4 bytes). * Finally, flags don't need
additional alignment. This layout provides * compact allocations without padding (even on platforms with alignment * requirements) at the cost of a bit slower
lookups. * * Objects with few keys don't have a hash index; keys are looked up linearly, * which is cache efficient because the keys are consecutive. Larger objects
* have a hash index part which contains integer indexes to the entries part. * * A single allocation reduces memory allocation overhead but requires more * work
when any part needs to be resized. A sliced allocation for entries * makes linear key matching faster on most platforms (more locality) and * skimps on flags size
(which would be followed by 3 bytes of padding in * most architectures if entries were placed in a struct). * * 'props' also contains internal properties distinguished
with a non-BMP * prefix. Often used properties should be placed early in 'props' whenever * possible to make accessing them as fast as possible.

label: code-design

44908. Catches >0x100000000 and negative values.

44909. * Entry points

44910. * Error built-ins

44911. Packed or unpacked tval

44912. * PC-to-line constants

44913. top-down expression parser

44914. E5 Section 15.4.5.1, steps 4.e.i - 4.e.ii

44915. fall through

44916. x

44917. **comment:** XXX: write protect after flag? -> any chance of handling it here?

label: code-design

44918. Term was '!' and is eaten entirely (including dup slashes).

44919. fast path for ASCII

44920. num_stack_res

44921. %lx

44922. * Not found as concrete or virtual

44923. 'Buffer'

44924. [... env target target]

44925. DUK_TOK_LOR

44926. DUK_OP_DECLVAR flags in A; bottom bits are reserved for propdesc flags (DUK_PROPDESC_FLAG_XXX)

44927. Day-of-month is one-based in the API, but zero-based * internally, so fix here. Note that month is zero-based * both in the API and internally.

44928. The coercion order must match the ToPropertyDescriptor() algorithm * so that side effects in coercion happen in the correct order. * (This order also happens to be compatible with duk_def_prop(), * although it doesn't matter in practice.)

44929. triggers garbage digit check below

44930. relevant array index is non-configurable, blocks write

44931. **comment:** the NaN variant we use

label: code-design

44932. * Compact the function template.

44933. PRIMARY EXPRESSIONS

44934. * indexOf(), lastIndexOf()

44935. Sometimes this assert is not true right now; it will be true after * rounding. See: test-bug-numconv-mantissa-assert.js.

44936. Automatic error throwing, retval check.

44937. Fast path for the case where the register * is a number (e.g. loop counter).

44938. always push some string

44939. spare

44940. Note: new_e_next matches pushed temp key count, and nothing can * fail above between the push and this point.

44941. * Heap flags

44942. Object extra properties. * * There are some difference between function templates and functions. * For example, function templates don't have .length and nargs is * normally used to instantiate the functions.

44943. We intentionally ignore the duk_safe_call() return value and only * check the output type. This way we don't also need to check that * the returned value is indeed a string in the success case.

44944. * Arguments exotic behavior not possible for new properties: all * magically bound properties are initially present in the arguments * object, and if they are deleted, the binding is also removed from * parameter map.

44945. get_value

44946. clear array part flag only after switching

44947. !

44948. tv1 points to value storage

44949. Map DUK_HBUFFEROBJECT_ELEM_xxx to prototype object built-in index. * Sync with duk_hbufferobject.h.

44950. * Create "fallback" object to be used as the object instance, * unless the constructor returns a replacement value. * Its internal prototype needs to be set based on "prototype" * property of the constructor.

44951. Slightly modified "Bernstein hash" from: * * http://eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx * * Modifications: string skipping and reverse direction similar to * Lua 5.1.5, and different hash initializer. * * The reverse direction ensures last byte it always included in the * hash which is a good default as changing parts of the string are * more often in the suffix than in the prefix.

44952. ascii fast path: avoid decoding utf-8

44953. just in case

44954. entry_top + 4

44955. m- <- (* m- B)

44956. UTC

44957. no valstack space check

44958. End of input (NUL) goes through slow path and causes SyntaxError.

44959. [... enum_target res trap_result val]

44960. **comment:** XXX: unnecessary '%'s' formatting here, but cannot use * 'msg' as a format string directly.

label: code-design

44961. ptr_curr_pc != NULL only when bytecode executor is active.

44962. Restore entry thread executor curr_pc stack frame pointer.

44963. Storing the entry top is cheaper here to ensure stack is correct at exit, * as there are several paths out.

44964. duk_hobject_set_length_zero(thr, func->h_funcs);

44965. fall thru

44966. see above

44967. * Struct defines

44968. [... name reg/null] -> [...]

44969. mandatory if du.d is a NaN

44970. duk_tval ptr for 'func' on stack (borrowed reference)

44971. For tv1 == tv2, this is a no-op (no explicit check needed).

44972. Use b[] to access the size of the union, which is strictly not * correct. Can't use fixed size unless there's feature detection * for pointer byte size.

44973. local copy to avoid relookups

44974. **comment:** XXX: this has some overlap with object inspection; remove this and make * DumpHeap return lists of heapptrs instead?
label: code-design

44975. Return to the bytecode executor caller which will unwind stacks. * Return value is already on the stack top: [...] retval].

44976. * Hobject Ecmascript [[Class]].

44977. Handle one slow path unit (or finish if we're done).

44978. **comment:** Copy the .buffer property, needed for TypedArray.prototype.subarray(). ** XXX: limit copy only for TypedArray classes specifically?
label: code-design

44979. DUK_HEAPHDR_HAS_FINALIZED may or may not be set.

44980. enough to cover the whole mantissa

44981. DUK_TOK_BXOR_EQ

44982. **comment:** * Hobject enumeration support. ** Creates an internal enumeration state object to be used e.g. with for-in * enumeration. The state object contains a snapshot of target object keys * and internal control state for enumeration. Enumerator flags allow caller * to e.g. request internal/non-enumerable properties, and to enumerate only * "own" properties. ** Also creates the result value for e.g. Object.keys() based on the same * internal structure. ** This snapshot-based enumeration approach is used to simplify enumeration: * non-snapshot-based approaches are difficult to reconcile with mutating * the enumeration target, running multiple long-lived enumerators at the * same time, garbage collection details, etc. The downside is that the * enumerator object is memory inefficient especially for iterating arrays.
label: code-design

44983. repl string scan

44984. [key result] -> [result]

44985. 'xffTracedata'

44986. ... and its 'message' from an instance property

44987. AUTHORS.rst

44988. Potential direct eval call detected, flag the CALL * so that a run-time "direct eval" check is made and * special behavior may be triggered. Note that this * does not prevent 'eval' from being register bound.

44989. prefer jump

44990. **comment:** * Arithmetic, binary, and logical helpers. ** Note: there is no opcode for logical AND or logical OR; this is on * purpose, because the evalution order semantics for them make such * opcodes pretty pointless: short circuiting means they are most * comfortably implemented as jumps. However, a logical NOT opcode * is useful. ** Note: careful with duk_tval pointers here: they are potentially * invalidated by any DECREF and almost any API call. It's still * preferable to work without making a copy but that's not always * possible.
label: code-design

44991. automatic length update

44992. args

44993. 'String'

44994. With ROM-based strings, heap->strs[] and thr->strs[] are omitted * so nothing to initialize for strs[].

44995. * Note: prototype chain is followed BEFORE first comparison. This * means that the instanceof lval is never itself compared to the * rval.prototype property. This is apparently intentional, see E5 * Section 15.3.5.3, step 4.a. ** Also note: * js> (function() {}) instanceof Function * true * js> Function instanceof Function * true ** For the latter, h_proto will be Function.prototype, which is the * built-in Function prototype. Because Function.[[Prototype]] is * also the built-in Function prototype, the result is true.

44996. not a vararg function

44997. Shouldn't happen but check anyway.

44998. E5 Sections 11.8.3, 11.8.5; x <= y --> not (x > y) --> not (y < x)

44999. Used by duk_emit*() calls so that we don't shuffle the loadints that * are needed to handle indirect opcodes.

45000. No strs[] pointer.

45001. **comment:** XXX: assertion that entries >= old_len are already unused
label: code-design

45002. e.g. DUK_OP_PREINCR

45003. eat 'while'

45004. nuke values at idx_rebase to get the first retval (initially * at idx_rcbase) to idx_rebase

45005. input radix

45006. char: use int cast

45007. result

45008. exponent digit

45009. **comment:** This is a cleaner approach and also produces smaller code than * the other alternative. Use duk_require_string() for format * safety (although the source property should always exist).
label: code-design

45010. defprop_flags

45011. s < 2

45012. spaces (nargs - 1) + newline

45013. * Parse inner function

45014. if accessor without getter, return value is undefined

45015. **comment:** * For non-ASCII strings, we need to scan forwards or backwards * from some starting point. The starting point may be the start * or end of the string, or some cached midpoint in the string * cache. ** For "short" strings we simply scan without checking or updating * the cache. For longer strings we check and update the cache as * necessary, inserting a new cache entry if none exists.
label: code-design

45016. XXX: len >= 0x80000000 won't work below because we need to be * able to represent -len.

45017. who resumed us (if any)

45018. out_clamped==NULL -> RangeError if outside range

45019. relookup after possible realloc

45020. [...] regexp_object escaped_source bytecode]

45021. m+ <- (* m+ B)

45022. bytecode opcode (or extraop) for binary ops

45023. forget temp

45024. * Shared assignment expression handling * * args = (opcode << 8) + rbp * * If 'opcode' is DUK_OP_NONE, plain assignment without arithmetic. * Syntactically valid left-hand-side forms which are not accepted as * left-hand-side values (e.g. as in "f() = 1") must NOT cause a * SyntaxError, but rather a run-time ReferenceError. ** When evaluating X <op>= Y, the LHS (X) is conceptually evaluated * to a temporary first. The RHS is then evaluated. Finally, the * <op> is applied to the initial value of RHS (not the value after * RHS evaluation), and written to X. Doing so concretely generates * inefficient code so we'd like to avoid the temporary when possible. * See: <https://github.com/svaarala/duktape/pull/992>. ** The expression value (final LHS value, written to RHS) is * conceptually copied into a fresh temporary so that it won't * change even if the LHS/RHS values change in outer expressions. * For example, it'd be generally incorrect for the expression value * to be the RHS register binding, unless there's a guarantee that it * won't change during further expression evaluation. Using the * temporary concretely produces inefficient bytecode, so we try to * avoid the extra temporary for some known-to-be-safe cases. * Currently the only safe case we detect is a

"top level assignment", * for example "x = y + z;", where the assignment expression value is * ignored. * See: test-dev-assign-expr.js and test-bug-assign-mutate-gh381.js.

45025. TRYCATCH, cannot emit now (not enough info)

45026. Value stack slot limits: these are quite approximate right now, and * because they overlap in control flow, some could be eliminated.

45027. absolute position for digit considered for rounding

45028. -> [... this timeval_new]

45029. TypeError if wrong; class check, see E5 Section 15.10.6

45030. DUK_HOBJECT_FLAG_EXOTIC_DUKFUNC: omitted here intentionally

45031. * Byte order. Important to self check, because on some exotic platforms * there is no actual detection but rather assumption based on platform * defines.

45032. Both inputs are zero; cases where only one is zero can go * through main algorithm.

45033. same thread

45034. 'Boolean'

45035. * Inside one or more 'with' statements fall back to slow path always. * (See e.g. test-stmt-with.js.)

45036. * Init the heap thread

45037. XXX: need a duk_require_func_or_lfunc_coerce()

45038. avoid referencing, invalidated

45039. Don't support "straddled" source now.

45040. * Object compaction. * Compaction is assumed to never throw an error.

45041. [arg1 ... argN this loggerLevel loggerName]

45042. duk_tval intentionally skipped

45043. * Since built-ins are not often extended, compact them.

45044. * Compilation

45045. Push a new ArrayBuffer (becomes view .buffer)

45046. idx_start

45047. Note: we rely on the _Varmap having a bunch of nice properties, like: * - being compacted and unmodified during this process * - not containing an array part * - having correct value types

45048. **comment:** XXX: return indication of "terminalness" (e.g. a 'throw' is terminal)

label: code-design

45049. may be NULL if no constants or inner funcs

45050. Equivalent year for DST calculations outside [1970,2038[range, see * E5 Section 15.9.1.8. Equivalent year has the same leap-year-ness and * starts with the same weekday on Jan 1. * https://bugzilla.mozilla.org/show_bug.cgi?id=351066

45051. empty expressions can be detected conveniently with nud/led counts

45052. There are at most 7 args, but we use 8 here so that also * DUK_DATE_IDX_WEEKDAY gets initialized (to zero) to avoid the potential * for any Valgrind gripes later.

45053. [... val callback thisArg val i obj]

45054. [... this tracedata sep this str1 ... strN]

45055. important chosen base types

45056. Digit count indicates number of fractions (i.e. an absolute * digit index instead of a relative one). Used together with * DUK_N2S_FLAG_FIXED_FORMAT for toFixed().

45057. must also check refzero_list

45058. MEMBER/NEW/CALL EXPRESSIONS

45059. **comment:** XXX: add flag to indicate whether caller cares about return value; this * affects e.g. handling of assignment expressions. This change needs API * changes elsewhere too.

label: code-design

45060. Fast jumps (which avoid longjmp) jump directly to the jump sites * which are always known even while the iteration/switch statement * is still being parsed. A final peephole pass "straightens out" * the jumps.

45061. [... func]

45062. * Function gets no new environment when called. This is the * case for global code, indirect eval code, and non-strict * direct eval code. There is no direct correspondence to the * E5 specification, as global/eval code is not exposed as a * function.

45063. Wrappers for calling standard math library methods. These may be required * on platforms where one or more of the math built-ins are defined as macros * or inline functions and are thus not suitable to be used as function pointers.

45064. **comment:** Throwing an error this deep makes the error rather vague, but * saves hundreds of bytes of code.

label: code-design

45065. This is not strictly necessary, but avoids compiler warnings; e.g. * gcc won't reliably detect that no uninitialized data is read below.

45066. Trigger at zero or below

45067. Custom coercion for API

45068. **comment:** Note: cannot read more than 24 bits without possibly shifting top bits out. * Fixable, but adds complexity.

label: code-design

45069. Almost all global level Function objects are constructable * but not all: Function.prototype is a non-constructable, * callable Function.

45070. source ends before dest starts

45071. * Statement terminator check, including automatic semicolon * handling. After this step, 'curr_tok' should be the first * token after a possible statement terminator.

45072. * Exposed string-to-number API * * Input: [string] * Output: [number] * * If number parsing fails, a NaN is pushed as the result. If number parsing * fails due to an internal error, an InternalError is thrown.

45073. 64-bit OK because always >= 0

45074. reg/const for switch value

45075. **comment:** When formatting an argument to a string, errors may happen from multiple * causes. In general we want to catch obvious errors like a toLogString() * throwing an error, but we don't currently try to catch every possible * error. In particular, internal errors (like out of memory or stack) are * not caught. Also, we expect Error toString() to not throw an error.

label: code-design

45076. Each round of finalizer execution may spawn new finalizable objects * which is normal behavior for some applications. Allow multiple * rounds of finalization, but use a shrinking limit based on the * first round to detect the case where a runaway finalizer creates * an unbounded amount of new finalizable objects. Finalizer rescue * is not supported: the semantics are unclear because most of the * objects being finalized here are already reachable. The finalizer * is given a boolean to indicate that rescue is not possible. * * See discussion in: <https://github.com/svaarala/duktape/pull/473>

45077. no prototype, updated below

45078. unwind to 'resume' caller

45079. 0xff-0x0ff

45080. **comment:** XXX: missing trap result validation for non-configurable target keys * (must be present), for non-extensible target all target keys must be * present and no extra keys can be present. * <http://www.ecma-international.org/ecma-262/6.0/#sec-proxy-object-internal-methods-and-internal-slots-ownpropertykeys>

label: code-design

45081. **comment:** NOTE: lightfuncs are coerced to full functions because * lightfuncs don't fit into a property value slot. This * has some side effects, see test-dev-lightfunc-accessor.js.

label: code-design

45082. IdentityEscape, with dollar added as a valid additional * non-standard escape (see test-regexp-identity-escape-dollar.js). * Careful not to match end-of-buffer (<0) here.

45083. 1st related value (type specific)

45084. The caller is responsible for being sure that bytecode being loaded * is valid and trusted. Invalid bytecode can cause memory unsafe * behavior directly during loading or later during bytecode execution * (instruction validation would be quite complex to implement). * * This signature check is the only sanity check for detecting * accidental invalid inputs. The initial 0xFF byte ensures no * ordinary string will be accepted by accident.

45085. Shared object part
45086. always throw ReferenceError for unresolvable
45087. * Mark roots, hoping that recursion limit is not normally hit. * If recursion limit is hit, run additional reachability rounds * starting from "temproots" until marking is complete. ** Marking happens in two phases: first we mark actual reachability * roots (and run "temproots" to complete the process). Then we * check which objects are unreachable and are finalizable; such * objects are marked as FINALIZABLE and marked as reachability * (and "temproots" is run again to complete the process). ** The heap finalize_list must also be marked as a reachability root. * There may be objects on the list from a previous round if the * previous run had finalizer skip flag.
45088. Shift to sign extend.
45089. '^'
45090. required_desc_flags
45091. **comment:** Stack indices for better readability
 label: code-design
45092. * Try registers
45093. no prototype or class yet
45094. shadowed; update value
45095. E5 Section 8.6.2 + custom classes
45096. **comment:** Lookup current thread; use the stable 'entry_thread' for this to * avoid clobber warnings. Any valid, reachable 'thr' value would be * fine for this, so using 'entry_thread' is just to silence warnings.
 label: code-design
45097. RELATIONAL EXPRESSION
45098. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property existence check right now.
45099. These are just convenience "wiping" of state. Side effects should * not be an issue here: thr->heap and thr->heap->jl have a stable * pointer. Finalizer runs etc capture even out-of-memory errors so * nothing should throw here.
45100. Zero escape (also allowed in non-strict mode)
45101. roughly 64 bytes
45102. XXX: this doesn't actually work properly for tail calls, so * tail calls are disabled when DUK_USE_NONSTD_FUNC_CALLER_PROPERTY * is in use.
45103. * Dragon4 slow path (binary) digit generation. * An extra digit is generated for rounding.
45104. this is a bit approximate (errors out before max is reached); this is OK
45105. **comment:** * Parse a function-body-like expression (FunctionBody or Program * in E5 grammar) using a two-pass parse. The productions appear * in the following contexts: * - function expression * - function statement * - function declaration * - getter in object literal * - setter in object literal * - global code * - eval code * - Function constructor body ** This function only parses the statement list of the body; the argument * list and possible function name must be initialized by the caller. * For instance, for Function constructor, the argument names are originally * on the value stack. The parsing of statements ends either at an EOF or * a closing brace; this is controlled by an input flag. ** Note that there are many differences affecting parsing and even code * generation: * - Global and eval code have an implicit return value generated * by the last statement; function code does not * - Global code, eval code, and Function constructor body end in * an EOF, other bodies in a closing brace ('}') ** Upon entry, 'curr_tok' is ignored and the function will pull in the * first token on its own. Upon exit, 'curr_tok' is the terminating * token (EOF or closing brace).
 label: code-design
45106. When torture not enabled, can just use the same helper because * 'reg' won't get spilled.
45107. Note: actual update happens once write has been completed * without error below. The write should always succeed * from a specification viewpoint, but we may e.g. run out * of memory. It's safer in this order.
45108. Convert buffer to result string.
45109. default prototype (Note: 'thr' must be reachable)
45110. we know these because enum objects are internally created
45111. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
 label: code-design
45112. **comment:** Direct eval requires that there's a current * activation and it is an Ecmascript function. * When Eval is executed from e.g. cooperate API * call we'll need to do an indirect eval instead.
 label: code-design
45113. 1 = num actual 'return values'
45114. should never happen
45115. 'l' verbatim
45116. Regexp tokens
45117. avoid attempt to compact the current object
45118. DUK_TOK_TRUE
45119. idx_bottom and idx_retval are only used for book-keeping of * Ecmascript-initiated calls, to allow returning to an Ecmascript * function properly. They are duk_size_t to match the convention * that value stack sizes are duk_size_t and local frame indices * are duk_idx_t.
45120. **comment:** This loop is optimized for size. For speed, there should be * two separate loops, and we should ensure that memcmp() can be * used without an extra "will searchstring fit" check. Doing * the preconditioning for 'p' and 'p_end' is easy but cpos * must be updated if 'p' is wound back (backward scanning).
 label: code-design
45121. min(incl)
45122. may be equal
45123. traceback depth ensures fits into 16 bits
45124. * Get the values onto the stack first. It would be possible to cover * some normal cases without resorting to the value stack. ** The right hand side could be a light function (as they generally * behave like objects). Light functions never have a 'prototype' * property so E5.1 Section 15.3.5.3 step 3 always throws a TypeError. * Using duk_require_hobject() is thus correct (except for error msg).
45125. DUK_TOK_GET
45126. * Various
45127. Module has now evaluated to a wrapped module function. Force its * .name to match module.name (defaults to last component of resolved * ID) so that it is shown in stack traces too. Note that we must not * introduce an actual name binding into the function scope (which is * usually the case with a named function) because it would affect the * scope seen by the module and shadow accesses to globals of the same name. * This is now done by compiling the function as anonymous and then forcing * its .name without setting a "has name binding" flag.
45128. Characters outside BMP cannot be escape()'d. We could * encode them as surrogate pairs (for codepoints inside * valid UTF-8 range, but not extended UTF-8). Because * escape() and unescape() are legacy functions, we don't.
45129. trailer
45130. * Check whether we need to abandon an array part (if it exists)
45131. [... env callee]
45132. **comment:** These is not 100% because format would need to be non-portable "long long". * Also print out as doubles to catch cases where the "long" type is not wide * enough; the limits will then not be printed accurately but the magnitude * will be correct.
 label: code-design
45133. an Ecmascript function
45134. **comment:** avoid pressure to add/remove strings, invalidation of call data argument, etc.
 label: code-design
45135. end of bytecode
45136. key is 'length', cannot match argument exotic behavior
45137. * First check whether property exists; if not, simple case. This covers * steps 1-4.
45138. * Do-while statement is mostly trivial, but there is special * handling for automatic semicolon handling (triggered by the * DUK__ALLOW_AUTO_SEMI_ALWAYS) flag related to a bug filed at: * * https://bugs.ecmascript.org/show_bug.cgi?id=8 * * See doc/compiler.rst for details.

45139. must be found: was found earlier, and cannot be inherited
45140. Input value should be on stack top and will be coerced and * popped. Refuse to update an Array's 'length' to a value * outside the 32-bit range. Negative zero is accepted as zero.
45141. shift in zeroes
45142. free object and all auxiliary (non-heap) allocs
45143. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Returns NULL for a lightfunc.
45144. bytes in dest
45145. stack[0] = regexp * stack[1] = string
45146. flags
45147. Check for maximum buffer length.
45148. 'Thread'
45149. **comment:** Pointer to bytecode executor's 'curr_pc' variable. Used to copy * the current PC back into the topmost activation when activation * state is about to change (or "syncing" is otherwise needed). This * is rather awkward but important for performance, see execution.rst.
label: code-design
45150. The Quote(value) operation: quote a string. * * Stack policy: [] -> [].
45151. Output #2: last component name
45152. tc1 = false, tc2 = false
45153. 0
45154. Exponent without a sign or with a +/- sign is accepted * by all call sites (even JSON.parse()).
45155. 0x90-0x9f
45156. **comment:** XXX: Shuffling support could be implemented here so that LDINT+LDINTX * would only shuffle once (instead of twice). The current code works * though, and has a smaller compiler footprint.
label: code-design
45157. * Write to entry part
45158. To use the shared helper need the virtual index.
45159. * Slow delete, but we don't care as we're already in a very slow path. * The delete always succeeds: key has no exotic behavior, property * is configurable, and no resize occurs.
45160. LAYOUT 1
45161. For len == 0, i is initialized to len - 1 which underflows. * The condition (i < len) will then exit the for-loop on the * first round which is correct. Similarly, loop termination * happens by i underflowing.
45162. Raw internal valstack access macros: access is unsafe so call site * must have a guarantee that the index is valid. When that is the case, * using these macro results in faster and smaller code than duk_get_tval(). * Both 'ctx' and 'idx' are evaluated multiple times, but only for asserts.
45163. * Node.js Buffer.isEncoding()
45164. array entries are all plain values
45165. **comment:** Buffer values are encoded in (lowercase) hex to make the * binary data readable. Base64 or similar would be more * compact but less readable, and the point of JX/JC * variants is to be as useful to a programmer as possible.
label: code-design
45166. **comment:** XXX: the helper currently assumes stack top contains new * 'length' value and the whole calling convention is not very * compatible with what we need.
label: code-design
45167. Coerce to number before validating pointers etc so that the * number coercions in duk_hbufferobject_validated_write() are * guaranteed to be side effect free and not invalidate the * pointer checks we do here.
45168. Mark-and-sweep interval is relative to combined count of objects and * strings kept in the heap during the latest mark-and-sweep pass. * Fixed point .8 multiplier and .0 adder. Trigger count (interval) is * decreased by each (re)allocation attempt (regardless of size), and each * refzero processed object. * * 'SKIP' indicates how many (re)allocations to wait until a retry if * GC is skipped because there is no thread do it with yet (happens * only during init phases).
45169. XXX: assert 'c' is an enumerator
45170. two special escapes: '\n' and "", other printables as is
45171. for indexOf, ToInteger(undefined) would be 0, i.e. correct, but * handle both indexOf and lastIndexOf specially here.
45172. DUK_USE_AUGMENT_ERROR_THROW
45173. * Exponent notation for non-base-10 numbers isn't specified in Ecmascript * specification, as it never explicitly turns up: non-decimal numbers can * only be formatted with Number.prototype.toString([radix]) and for that, * behavior is not explicitly specified. * * Logical choices include formatting the exponent as decimal (e.g. binary * 100000 as 1e+5) or in current radix (e.g. binary 100000 as 1e+101). * The Dragon4 algorithm (in the original paper) prints the exponent value * in the target radix B. However, for radix values 15 and above, the * exponent separator 'e' is no longer easily parseable. Consider, for * instance, the number "1.faecce+1c".
45174. Template functions are not strictly constructable (they don't * have a "prototype" property for instance), so leave the * DUK_HOBJECT_FLAG_CONSTRUCTABLE flag cleared here.
45175. **comment:** Most practical strings will go here.
label: code-design
45176. non-strict eval: env is caller's env or global env (direct vs. indirect call) * global code: env is is global env
45177. **comment:** XXX: use the exactly same arithmetic function here as in executor
label: code-design
45178. -> [... lval new_rval]
45179. sep (even before first one)
45180. **comment:** XXX: use advancing pointers instead of index macros -> faster and smaller?
label: code-design
45181. 0x7F is special
45182. IEEE requires that zeros compare the same regardless * of their signed, so if both x and y are zeroes, they * are caught above.
45183. Signed integers always map to 4 bytes now.
45184. Resolve Proxy targets until Proxy chain ends. No explicit check for * a Proxy loop: user code cannot create such a loop without tweaking * internal properties directly.
45185. estimate is valid
45186. Note: it's nice if size is 2^N (not 4x4 = 16 bytes on 32 bit)
45187. always true, arg is unsigned
45188. ArrayBuffer: unlike any other argument variant, create * a view into the existing buffer.
45189. DUK_TOK_DO
45190. * Slow path: potentially requires function calls for coercion
45191. 'ArrayBuffer'
45192. Initialize index so that we skip internal control keys.
45193. flags have been already cleared
45194. * ArrayBuffer, DataView, and TypedArray constructors
45195. **comment:** * XXX: for now, indicate that an expensive catch binding * declarative environment is always needed. If we don't * need it, we don't need the const_varname either.
label: code-design
45196. -> [... this timeval_new timeval_new]
45197. DUK_TOK_CONST
45198. **comment:** XXX: unnecessary copying of values? Just set 'top' to * b + c, and let the return handling fix up the stack frame?
label: code-design
45199. Slot C

45200. **comment:** Array is dense and contains only strings, but ASIZE may * be larger than used part and there are UNUSED entries.

label: code-design

45201. At least: ... [err]

45202. wrapped

45203. whole or fraction digit

45204. * Constructor calls

45205. only 'length'

45206. DUK_USE_DEBUGGER_DUMPHEAP

45207. 'toISOString'

45208. 'res' contains expression value

45209. No other escape beginning with a digit in strict mode

45210. **comment:** Use Murmurhash2 directly for short strings, and use "block skipping" * for long strings: hash an initial part and then sample the rest of * the string with reasonably sized chunks. An initial offset for the * sampling is computed based on a hash of the initial part of the string; * this is done to (usually) avoid the case where all long strings have * certain offset ranges which are never sampled. * * Skip should depend on length and bound the total time to roughly * logarithmic. With current values: * * 1M string => 256 * 241 = 61696 bytes (0.06M) of hashing * 1G string => 256 * 16321 = 4178176 bytes (3.98M) of hashing * * XXX: It would be better to compute the skip offset more "smoothly" * instead of having a few boundary values.

label: code-design

45211. [key setter this val key] -> [key retval]

45212. point to start of 'reserved area'

45213. Shared helper for match() steps 3-4, search() steps 3-4.

45214. dummy

45215. Just call the "original" Object.defineProperties() to * finish up.

45216. Although these could be parsed as PatternCharacters unambiguously (here), * E5 Section 15.10.1 grammar explicitly forbids these as PatternCharacters.

45217. line terminator will be handled on next round

45218. * Debug dumping

45219. x in [2**31, 2**32[

45220. Error path.

45221. if a site already exists, nop: max one label site per statement

45222. Flag ORed to err_code to indicate __FILE__ / __LINE__ is not * blamed as source of error for error fileName / lineNumber.

45223. Start in paused state.

45224. **comment:** XXX: much to improve (code size)

label: code-design

45225. [regexp string res_arr]

45226. range conversion with a "skip"

45227. **comment:** Update function min/max line from current token. Needed to improve * function line range information for debugging, so that e.g. opening * curly brace is covered by line range even when no opcodes are emitted * for the line containing the brace.

label: code-design

45228. * Regular expression structs, constants, and bytecode defines.

45229. **comment:** should never be zero, because we (Duktape.Thread.yield) are on the stack

label: code-design

45230. **comment:** XXX: This will now return false for non-numbers, even though they would * coerce to NaN (as a general rule). In particular, duk_get_number() * returns a NaN for non-numbers, so should this function also return * true for non-numbers?

label: code-design

45231. **comment:** The handler is looked up with a normal property lookup; it may be an * accessor or the handler object itself may be a proxy object. If the * handler is a proxy, we need to extend the valstack as we make a * recursive proxy check without a function call in between (in fact * there is no limit to the potential recursion here). * * (For sanity, proxy creation rejects another proxy object as either * the handler or the target at the moment so recursive proxy cases * are not realized now.)

label: code-design

45232. Flag handling currently assumes that flags are consistent. This is OK * because the call sites are now strictly controlled.

45233. DUK_TOK_DEFAULT

45234. [key value] or [key undefined]

45235. Pad significand with "virtual" zero digits so that Dragon4 will * have enough (apparent) precision to work with.

45236. * Set lexer input position and reinitialize lookup window.

45237. nargs

45238. With lightfuncs, act 'func' may be NULL

45239. return '(?:)'

45240. DUK_USE_REFZERO_FINALIZER_TORTURE

45241. next to use, highest used is top - 1

45242. **comment:** XXX: actually single step levels would work just fine, clean up

label: code-design

45243. * DUK_CALL_FLAG_IGNORE_RECLIMIT causes duk_handle_call() to ignore C * recursion depth limit (and won't increase it either). This is * dangerous, but useful because it allows the error handler to run * even if the original error is caused by C recursion depth limit. * * The heap level DUK_HEAP_FLAG_ERRHANDLER_RUNNING is set for the * duration of the error handler and cleared afterwards. This flag * prevents the error handler from running recursively. The flag is * heap level so that the flag properly controls even coroutines * launched by an error handler. Since the flag is heap level, it is * critical to restore it correctly. * * We ignore errors now: a success return and an error value both * replace the original error value. (This would be easy to change.)

45244. **comment:** XXX: remove heap->dbg_exec_counter, use heap->inst_count_interrupt instead?

label: code-design

45245. DUK_USE_JSON_DECSTRING_FASTPATH

45246. The actual detached_cb call is postponed to message loop so * we don't need any special precautions here (just skip to EOM * on the already closed connection).

45247. arg2 would be clobbered so reassign it to a temp.

45248. standard behavior, step 3.f.i

45249. avoid degenerate cases, so that (len - 1) won't underflow

45250. Note: %06d for positive value, %07d for negative value to include * sign and 6 digits.

45251. 'let'

45252. * Halt execution helper

45253. 1110 xxxx; 3 bytes

45254. ch1 = (r_increment << 8) + byte

45255. variable name reg/const, if variable not register-bound

45256. Cast converts magic to 16-bit signed value

45257. * Remove the object from the refzero list. This cannot be done * before a possible finalizer has been executed; the finalizer * may trigger a mark-and-sweep, and mark-and-sweep must be able * to traverse a complete refzero_list.

45258. Argument variants. When the argument is an ArrayBuffer a view to * the same buffer is created; otherwise a new ArrayBuffer is always * created.

45259. entry part size

45260. * Property not found in prototype chain.

45261. function executes in strict mode

45262. 'data' is reachable through every compiled function which * contains a reference.

45263. Caller must trigger recomputation of active breakpoint list. To * ensure stale values are not used if that doesn't happen, clear the * active breakpoint list here.

45264. * Longjmp types, also double as identifying continuation type for a rethrow (in 'finally')

45265. note: long live range
45266. * Object will be kept; queue object back to heap_allocated (to tail)
45267. no change
45268. DUK_USE_ASSERTIONS
45269. force the property to 'undefined' to create a slot for it
45270. Optimized for speed.
45271. **comment:** prevent multiple in-progress detaches
 label: requirement
45272. Set a high interrupt counter; the original executor * interrupt invocation will rewrite before exiting.
45273. DUK_TOK_THIS
45274. index
45275. * Expression parsing: duk__expr_nud(), duk__expr_led(), duk__expr_lbp(), and helpers. * * - duk__expr_nud(): ("null denotation"): process prev_token as a "start" of an expression (e.g. literal) * - duk__expr_led(): ("left denotation"): process prev_token in the "middle" of an expression (e.g. operator) * - duk__expr_lbp(): ("left-binding power"): return left-binding power of curr_token
45276. 3-letter log level strings
45277. **comment:** * Convert char offset to byte offset * * Avoid using the string cache if possible: for ASCII strings byte and * char offsets are equal and for short strings direct scanning may be * better than using the string cache (which may evict a more important * entry). * * Typing now assumes 32-bit string byte/char offsets (duk_uint_fast32_t). * Better typing might be to use duk_size_t.
 label: code-design
45278. Must prevent finalizers which may have arbitrary side effects.
45279. DUK_EXCEPTION_H_INCLUDED
45280. invalidates h_buf pointer
45281. Special case: original qmin was zero so there is nothing * to repeat. Emit an atom copy but jump over it here.
45282. **comment:** Higher footprint, less churn.
 label: code-design
45283. -> [... holder name val new_elem]
45284. NUL term or -1 (EOF), NUL check would suffice
45285. core property functions
45286. 25%, i.e. less than 25% used -> abandon
45287. Allow empty fraction (e.g. "123.")
45288. 0xf0...0xff
45289. don't allow const
45290. this is just beneath bottom
45291. * Duktape.Buffer: constructor
45292. 'Uint8Array'
45293. * Helpers for writing multiple properties
45294. exhaustive
45295. refcounting requires direct heap frees, which in turn requires a dual linked heap
45296. lastIndexOff() needs to be a vararg function because we must distinguish * between an undefined fromIndex and a "not given" fromIndex; indexOf() is * made vararg for symmetry although it doesn't strictly need to be.
45297. [... error]
45298. XXX: share final setting code for value and flags? difficult because * refcount code is different. Share entry allocation? But can't allocate * until array index checked.
45299. Terminating conditions. For fixed width output, we just ignore the * terminating conditions (and pretend that tc1 == tc2 == false). The * the current shortcut for fixed-format output is to generate a few * extra digits and use rounding (with carry) to finish the output.
45300. Return value when returning to this activation (points to caller * reg, not callee reg); index is absolute (only set if activation is * not topmost). * * Note: idx_bottom is always set, while idx_retval is only applicable * for activations below the topmost one. Currently idx_retval for * the topmost activation is considered garbage (and it not initialized * on entry or cleared on return; may contain previous or garbage * values).
45301. Unlike non-obsolete String calls, substr() algorithm in E5.1 * specification will happily coerce undefined and null to strings * ("undefined" and "null").
45302. DecimalEscape, only \0 is allowed, no leading zeroes are allowed
45303. parse new token
45304. 1e9
45305. Negative value checked so that a "time jump" works * reasonably. * * Same interval is now used for status sending and * peeking.
45306. * Misc shared helpers.
45307. don't set 'n' at all, inherited value is used as name
45308. ':'
45309. switch cmd
45310. **comment:** XXX: could add a fast path to process chunks of input codepoints, * but relative benefit would be quite small.
 label: code-design
45311. Copy interrupt counter/init value state to new thread (if any). * It's OK for new_thr to be the same as curr_thr.
45312. User totalLength overrides a computed length, but we'll check * every copy in the copy loop. Note that duk_to_uint() can * technically have arbitrary side effects so we need to recheck * the buffers in the copy loop.
45313. **comment:** XXX: some overlapping code; cleanup
 label: code-design
45314. stack[0] = searchElement * stack[1] = fromIndex * stack[2] = object * stack[3] = length (not needed, but not popped above)
45315. from curr pc
45316. * Maximum size check
45317. isError flag for yield
45318. * callstack_top - 1 --> this function * callstack_top - 2 --> caller (may not exist) * * If called directly from C, callstack_top might be 1. If calling * activation doesn't exist, call must be indirect.
45319. no need to unwind callstack
45320. * Array part
45321. **comment:** * Number should already be in NaN-normalized form, * but let's normalize anyway.
 label: code-design
45322. [array totalLength bufres buf]
45323. When src_size == 0, src_data may be NULL (if source * buffer is dynamic), and dst_data may be NULL (if * target buffer is dynamic). Avoid zero-size memcpy() * with an invalid pointer.
45324. [r1,r2] is the range
45325. * NormalizePropertyDescriptor() related helper. * * Internal helper which validates and normalizes a property descriptor * represented as an EcmaScript object (e.g. argument to defineProperty()). * The output of this conversion is a set of defprop_flags and possibly * some values pushed on the value stack; some subset of: property value, * getter, setter. Caller must manage stack top carefully because the * number of values pushed depends on the input property descriptor. * * The original descriptor object must not be altered in the process.
45326. duk_handle_call / duk_handle_safe_call recursion depth limiting
45327. DUK_HOBJECT_FLAG_NEWENV: handled below
45328. A leading digit is not required in some cases, e.g. accept ".123". * In other cases (JSON.parse()) a leading digit is required. This * is checked for after the loop.
45329. common case, already closed, so skip
45330. ignore millisecond fractions after 3
45331. * unshift()

45332. E5 Section 10.4.3

45333. No built-in functions are constructable except the top * level ones (Number, etc).

45334. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed, * and we're in an infinite loop.

45335. Raw helper for getting a value from the stack, checking its tag. * The tag cannot be a number because numbers don't have an internal * tag in the packed representation.

45336. Bernstein hash init value is normally 5381

45337. * DECLVAR ** See E5 Sections: * 10.4.3 Entering Function Code * 10.5 Declaration Binding Instantiation * 12.2 Variable Statement * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution ** Variable declaration behavior is mainly discussed in Section 10.5, * and is not discussed in the execution semantics (Sections 11-13). ** Conceptually declarations happen when code (global, eval, function) * is entered, before any user code is executed. In practice, register- * bound identifiers are 'declared' automatically (by virtue of being * allocated to registers with the initial value 'undefined'). Other * identifiers are declared in the function prologue with this primitive. ** Since non-register bindings eventually back to an internal object's * properties, the 'prop_flags' argument is used to specify binding * type: * * - Immutable binding: set DUK_PROPDESC_FLAG_WRITABLE to false * - Non-deletable binding: set DUK_PROPDESC_FLAG_CONFIGURABLE to false * - The flag DUK_PROPDESC_FLAG_ENUMERABLE should be set, although it * doesn't really matter for internal objects * * All bindings are non-deletable mutable bindings except: * * - Declarations in eval code (mutable, deletable) * - 'arguments' binding in strict function code (immutable) * - Function name binding of a function expression (immutable) ** Declarations may go to declarative environment records (always * so for functions), but may also go to object environment records * (e.g. global code). The global object environment has special * behavior when re-declaring a function (but not a variable); see * E5.1 specification, Section 10.5, step 5.e. ** Declarations always go to the 'top-most' environment record, i.e. * we never check the record chain. It's not an error even if a * property (even an immutable or non-deletable one) of the same name * already exists. ** If a declared variable already exists, its value needs to be updated * (if possible). Returns 1 if a PUTVAR needs to be done by the caller; * otherwise returns 0.

45338. **comment:** duk_handle_call_xxx: call ignores C recursion limit (for errhandler calls)

label: code-design

45339. expt 0xffff is infinite/NaN

45340. Augment the error if called as a normal function. __FILE__ and __LINE__ * are not desirable in this case.

45341. %s

45342. e.g. "x" < "xx"

45343. **comment:** XXX: for fastints, could use a variant which assumes a double duk_tval * (and doesn't need to check for fastint again).

label: code-design

45344. SameValue

45345. How much stack to require on entry to object/array encode

45346. * Parse code after the clause. Possible terminators are * 'case', 'default', and '}'. ** Note that there may be no code at all, not even an empty statement, * between case clauses. This must be handled just like an empty statement * (omitting seemingly pointless JUMPs), to avoid situations like * test-bug-case-fallthrough.js.

45347. ... name ': ' message

45348. to exit

45349. parenthesis count, 0 = top level

45350. * Useful Unicode codepoints * * Integer constants must be signed to avoid unexpected coercions * in comparisons.

45351. input string (may be a user pointer)

45352. * round_idx points to the digit which is considered for rounding; the * digit to its left is the final digit of the rounded value. If round_idx * is zero, rounding will be performed; the result will either be an empty * rounded value or if carry happens a '1' digit is generated.

45353. Careful with carry condition: * - If carry not added: 0x12345678 + 0 + 0xffffffff = 0x12345677 (< 0x12345678) * - If carry added: 0x12345678 + 1 + 0xffffffff = 0x12345678 (== 0x12345678)

45354. shared helper

45355. * Fast buffer writer with spare management.

45356. success/error path both do this

45357. DUK_TOK_MUL

45358. attempt to change from accessor to data property

45359. allowed ascii whitespace

45360. [... source? filename?]

45361. continue jump

45362. -> [... key val]

45363. Prepare value stack for a method call through an object property. * May currently throw an error e.g. when getting the property.

45364. no need to create environment record now; leave as NULL

45365. * Thread state check and book-keeping.

45366. "/=" and not in regexp mode

45367. for lastIndexOf, result may be -1 (mark immediate termination)

45368. fatal_func should be noreturn, but noreturn declarations on function * pointers has a very spotty support apparently so it's not currently * done.

45369. requested identifier

45370. Function expression. Note that any statement beginning with 'function' * is handled by the statement parser as a function declaration, or a * non-standard function expression/statement (or a SyntaxError). We only * handle actual function expressions (occurring inside an expression) here. ** O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnum().

45371. * JSON.stringify() fast path * * Otherwise supports full JSON, JX, and JC features, but bails out on any * possible side effect which might change the value being serialized. The * fast path can take advantage of the fact that the value being serialized * is unchanged so that we can walk directly through property tables etc.

45372. **comment:** XXX: helper

label: code-design

45373. no return value -> don't replace created value

45374. 'Arguments' object and has arguments exotic behavior (non-strict callee)

45375. [requested_id require require.id resolved_id last_comp Duktape.modLoaded Duktape.modLoaded[id]]

45376. Note: left shift, should mask

45377. 31 to 36

45378. two level break

45379. don't allow an empty match at the end of the string

45380. 'thr' is now reachable

45381. Parse argument list. Arguments are written to temps starting from * "next temp". Returns number of arguments parsed. Expects left paren * to be already eaten, and eats the right paren before returning.

45382. Allow 'Infinity'

45383. Opcode slot C is used in a non-standard way, so shuffling * is not allowed.

45384. * Helpers for managing property storage size

45385. if-digit-else-crl

45386. Safe write calls which will ensure space first.

45387. having this as a separate function provided a size benefit

45388. code emission

45389. **comment:** * obj->props is intentionally left as NULL, and duk_hobject_props.c must deal * with this properly. This is intentional: empty objects consume a minimum * amount of memory. Further, an initial allocation might fail and cause * 'obj' to "leak" (require a mark-and-sweep) since it is not reachable yet.

label: code-design

45390. Term was '.', backtrack resolved name by one component. * q[-1] = previous slash (or beyond start of buffer) * q[-2] = last char of previous component (or beyond start of buffer)

45391. **comment:** This could also be thrown internally (set the error, goto check_longjmp), * but it's better for internal errors to bubble outwards so that we won't * infinite loop in this catchpoint.

label: code-design

45392. * Debugging related API calls

45393. -> [... func this arg1 ... argN _Args length]

45394. 'Float64Array'

45395. 1 1 1 <32 bits> * Encode in two parts to avoid bitencode 24-bit limitation

45396. -> [this]

45397. **comment:** Note: array entries are always writable, so the writability check * above is pointless for them. The check could be avoided with some * refactoring but is probably not worth it.

label: code-design

45398. retval indicates delete failed

45399. formatters always get one-based month/day-of-month

45400. continue matching, set neg_tzoffset flag

45401. bound function chain has already been resolved

45402. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property get right now.

45403. standard JSON; array gaps

45404. Note: array part values are [[Writable]], [[Enumerable]], * and [[Configurable]] which matches the required attributes * here.

45405. 0=before period/exp, * 1=after period, before exp * 2=after exp, allow '+' or '-' * 3=after exp and exp sign

45406. just check 'env'

45407. **comment:** XXX: primitive to make array from valstack slice

label: code-design

45408. **comment:** * Dragon4 setup. * * Convert double from IEEE representation for conversion; * normal finite values have an implicit leading 1-bit. The * slow path algorithm doesn't handle zero, so zero is special * cased here but still creates a valid nc_ctx, and goes * through normal formatting in case special formatting has * been requested (e.g. forced exponential format: 0 -> "0e+0").

label: code-design

45409. consistency requires this

45410. DUK_TOK_CASE

45411. Nothing to initialize, strsf[] is in ROM.

45412. currently 6 characters of lookup are actually needed (duk_lexer.c)

45413. index/length check guarantees

45414. left.x1 -> res.x1

45415. * Case conversion tables generated using src/extract_caseconv.py.

45416. is global code

45417. **comment:** * Macros to set a duk_tval and update refcount of the target (decref the * old value and incref the new value if necessary). This is both performance * and footprint critical; any changes made should be measured for size/speed.

label: code-design

45418. We don't duk_require_stack() here now, but rely on the caller having * enough space.

45419. * Regexp executor. * * Safety: the EcmaScript executor should prevent user from reading and * replacing regexp bytecode. Even so, the executor must validate all * memory accesses etc. When an invalid access is detected (e.g. a 'save' * opcode to invalid, unallocated index) it should fail with an internal * error but not cause a segmentation fault. * * Notes: * * - Backtrack counts are limited to unsigned 32 bits but should * technically be duk_size_t for strings longer than 4G chars. * This also requires a regexp bytecode change.

45420. Note: not a valid stack index if num_stack_args == 0

45421. * Convenience (independent of representation)

45422. no refcounting

45423. duk_handle_ecma_call_setup: setup for a resume()

45424. escape any non-ASCII characters

45425. if 0, 'str' used, if > 0, 'strlist' used

45426. These limits are based on bytecode limits. Max temps is limited * by duk_hcompiledfunction nargs/nregs fields being 16 bits.

45427. * pop(), push()

45428. 'DecEnv'

45429. number of elements guaranteed to be user accessible * (in addition to call arguments) on Duktape/C function entry.

45430. NOTE: The Array special behaviors are NOT invoked by duk_xdef_prop_index() * (which differs from the official algorithm). If no error is thrown, this * doesn't matter as the length is updated at the end. However, if an error * is thrown, the length will be unset. That shouldn't matter because the * caller won't get a reference to the intermediate value.

45431. Torture option to shake out finalizer side effect issues: * make a bogus function call for every finalizable object, * essentially simulating the case where everything has a * finalizer.

45432. [... key]

45433. **comment:** This seems faster than emitting bytes one at a time and * then potentially rewinding.

label: code-design

45434. * Writability check

45435. may be changed by call

45436. DUK_USE_NONSTD_FUNC_CALLER_PROPERTY

45437. DUK_USE_JSON_STRINGIFY_FASTPATH

45438. Update all labels with matching label_id.

45439. guaranteed by duk_to_string()

45440. DUK_USE_TRACEBACKS

45441. **comment:** Formatting function pointers is tricky: there is no standard pointer for * function pointers and the size of a function pointer may depend on the * specific pointer type. This helper formats a function pointer based on * its memory layout to get something useful on most platforms.

label: code-design

45442. ensures callstack_top - 1 >= 0

45443. Count free operations toward triggering a GC but never actually trigger * a GC from a free. Otherwise code which frees internal structures would * need to put in NULLs at every turn to ensure the object is always in * consistent state for a mark-and-sweep.

45444. detached

45445. 'constructor'

45446. disabled for now

45447. idx_replacer

45448. type to represent a straight register reference, with <0 indicating none

45449. continue jump not patched, an INVALID opcode remains there

45450. -> [... enum_target res trap_result]

45451. current (next) array index

45452. fmin() with args -0 and +0 is not guaranteed to return * -0 as EcmaScript requires.

45453. 'const'

45454. all codepoints up to U+10FFFF

45455. **comment:** This is performance critical because it's needed for every DECREF. * Take advantage of the fact that the first heap allocated tag is 8, * so that bit 3 is set for all heap allocated tags (and never set for * non-heap-allocated tags).

label: code-design

45456. if no NaN handling flag, may still be NaN here, but not Inf

45457. thr argument only used for thr->heap, so specific thread doesn't matter

45458. Preshifted << 4. Must use 16-bit entry to allow negative value signaling.

45459. [0x00000000, 0xffffffff]
45460. fall through to error
45461. [... key_obj key key]
45462. Specification stripPrefix maps to DUK_S2N_FLAG_ALLOW_AUTO_HEX_INT. * * Don't autodetect octals (from leading zeroes), require user code to * provide an explicit radix 8 for parsing octal. See write-up from Mozilla: * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt#ECMAScript_5_Removes_Octal_Interpretation
45463. Process messages until we're no longer paused or we peek * and see there's nothing to read right now.
45464. XXX: callstack unwind may now throw an error when closing * scopes; this is a sandboxing issue, described in: * <https://github.com/svaarala/duktape/issues/476>
45465. since no recursive error handler calls
45466. **comment:** XXX: other properties of function instances; 'arguments', 'caller'.
 label: code-design
45467. numeric label_id (-1 reserved as marker)
45468. irrelevant when out->value == NULL
45469. Avoid fake finalization when callstack limit has been reached. * Otherwise a callstack limit error will be created, then refzero'ed.
45470. **comment:** * Helper to sort array index keys. The keys are in the enumeration object * entry part, starting from DUK__ENUM_START_INDEX, and the entry part is dense. * * We use insertion sort because it is simple (leading to compact code,) * works nicely in-place, and minimizes operations if data is already sorted * or nearly sorted (which is a very common case here). It also minimizes * the use of element comparisons in general. This is nice because element * comparisons here involve re-parsing the string keys into numbers each * time, which is naturally very expensive. * * Note that the entry part values are all "true", e.g. * * "1" -> true, "3" -> true, "2" -> true * * so it suffices to only work in the key part without exchanging any keys, * simplifying the sort. * *
http://en.wikipedia.org/wiki/Insertion_sort * * (Compiles to about 160 bytes now as a stand-alone function.)
 label: code-design
45471. The function object is now reachable and refcounts are fine, * so we can pop off all the temporaries.
45472. Care must be taken to avoid pointer wrapping in the index * validation. For instance, on a 32-bit platform with 8-byte * duk_tval the index 0x20000000UL would wrap the memory space * once.
45473. Convert a duk_tval fastint (caller checks) to a 32-bit index.
45474. Insert bytes in the middle of the buffer from a slice already * in the buffer. Source offset is interpreted "before" the operation.
45475. DUK_USE_STRTAB_CHAIN
45476. Original function instance/template had NAMEBINDING. * Must create a lexical environment on loading to allow * recursive functions like 'function foo() { foo(); }'.
45477. **comment:** the message should be a compile time constant without formatting (less risk); * we don't care about assertion text size because they're not used in production * builds.
 label: code-design
45478. assume value is a number
45479. **comment:** combines steps 3, 6; step 7 is not needed
 label: requirement
45480. * Debugger support
45481. **comment:** XXX: spare handling, slow now
 label: code-design
45482. normal and constructor calls have identical semantics
45483. * A token is interpreted as any possible production of InputElementDiv * and InputElementRegExp, see E5 Section 7 in its entirety. Note that * the E5 "Token" production does not cover all actual tokens of the * language (which is explicitly stated in the specification, Section 7.5). * Null and boolean literals are defined as part of both ReservedWord * (E5 Section 7.6.1) and Literal (E5 Section 7.8) productions. Here, * null and boolean values have literal tokens, and are not reserved * words. * * Decimal literal negative/positive sign is -not- part of DUK_TOK_NUMBER. * The number tokens always have a non-negative value. The unary minus * operator in "-1.0" is optimized during compilation to yield a single * negative constant. * * Token numbering is free except that reserved words are required to be * in a continuous range and in a particular order. See genstrings.py.
45484. Same coercion behavior as for Number.
45485. the outer loop will recheck and exit
45486. **comment:** These are macros for now, but could be separate functions to reduce code * footprint (check call site count before refactoring).
 label: code-design
45487. x == -Infinity
45488. [... source? func_template]
45489. other call
45490. lastIndex already set up for next match
45491. * Macros to access the 'props' allocation.
45492. **comment:** * Struct size/alignment if platform requires it * * There are some compiler specific struct padding pragmas etc in use, this * selftest ensures they're correctly detected and used.
 label: code-design
45493. [... source? filename?] (depends on flags)
45494. ToUint16() coercion is mandatory in the E5.1 specification, but * this non-compliant behavior makes more sense because we support * non-BMP codepoints. Don't use CESU-8 because that'd create * surrogate pairs.
45495. match string
45496. [targetBuffer targetStart sourceStart sourceEnd]
45497. x = sign(x) * floor(abs(x)), i.e. truncate towards zero, keep sign
45498. **comment:** XXX: here again finalizer thread is the heap_thread which needs * to be coordinated with finalizer thread fixes.
 label: code-design
45499. at least one opcode emitted
45500. Slice offsets are element (not byte) offsets, which only matters * for TypedArray views, Node.js Buffer and ArrayBuffer have shift * zero so byte and element offsets are the same. Negative indices * are counted from end of slice, crossed indices are allowed (and * result in zero length result), and final values are clamped * against the current slice. There's intentionally no check * against the underlying buffer here.
45501. side effects, in theory (referenced by global env)
45502. * Object.seal() and Object.freeze() (E5 Sections 15.2.3.8 and 15.2.3.9) * * Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. * * Note: virtual (non-concrete) properties which are non-configurable but * writable would pose some problems, but such properties do not currently * exist (all virtual properties are non-configurable and non-writable). * If they did exist, the non-configurability does NOT prevent them from * becoming non-writable. However, this change should be recorded somehow * so that it would turn up (e.g. when getting the property descriptor), * requiring some additional flags in the object.
45503. additional flags which are passed in the same flags argument as property * flags but are not stored in object properties.
45504. **comment:** XXX: Is this INCREF necessary? 'func' is always a borrowed * reference reachable through the value stack? If changed, stack * unwind code also needs to be fixed to match.
 label: code-design
45505. **comment:** XXX: shared decoding of 'b' and 'c'?
 label: code-design
45506. **comment:** The fast path for array property put is not fully compliant: * If one places conflicting number-indexed properties into * Array.prototype (for example, a non-writable Array.prototype[7]) * the fast path will incorrectly ignore them. * * This fast path could be made compliant by falling through * to the slow path if the previous value was UNUSED. This would * also remove the need to check for extensibility. Right now a * non-extensible array is slower than an extensible one as far * as writes are concerned. * * The fast path behavior is documented in more detail here: * tests/ecmascript/test-misc-array-fast-write.js
 label: code-design
45507. **comment:** XXX: because we unwind stacks above, thr->heap->curr_thread is at * risk of pointing to an already freed thread. This was indeed the * case in test-bug-multithread-valgrind.c, until duk_handle_call() * was fixed to restore thr->heap->curr_thread before rethrowing an * uncaught error.

label: code-design

45508. Allow naked fraction (e.g. ".123")
45509. * Rate limit check for sending status update or peeking into * the debug transport. Both can be expensive operations that * we don't want to do on every opcode. * * Making sure the interval remains reasonable on a wide variety * of targets and bytecode is difficult without a timestamp, so * we use a Date-provided timestamp for the rate limit check. * But since it's also expensive to get a timestamp, a bytecode * counter is used to rate limit getting timestamps.
45510. >>> struct.unpack('>d', '4000112233445566'.decode('hex')) * (2.008366013071895)
45511. Other longjmp types are handled by executor before propagating * the error here.
45512. * E5 Section 11.8.6 describes the main algorithm, which uses * [[HasInstance]]. [[HasInstance]] is defined for only * function objects: * * - Normal functions: * E5 Section 15.3.5.3 * - Functions established with Function.prototype.bind(): * E5 Section 15.3.4.5.3 * * For other objects, a TypeError is thrown. * * Limited Proxy support: don't support 'getPrototypeOf' trap but * continue lookup in Proxy target if the value is a Proxy.
45513. * Ecmascript/native function call or lightfunc call
45514. Halt execution and enter a debugger message loop until execution is resumed * by the client. PC for the current activation may be temporarily decremented * so that the 'current' instruction will be shown by the client. This helper * is callable from anywhere, also outside bytecode executor.
45515. computed live
45516. **comment:** Load a function from bytecode. The function object returned here must * match what is created by duk_js_push_closure() with respect to its flags, * properties, etc. * * NOTE: there are intentionally no input buffer length / bound checks. * Adding them would be easy but wouldn't ensure memory safety as untrusted * or broken bytecode is unsafe during execution unless the opcodes themselves * are validated (which is quite complex, especially for indirect opcodes).
45517. * For an external string, the NUL-terminated string data is stored * externally. The user must guarantee that data behind this pointer * doesn't change while it's used.
45518. Note: masking of 'x' is not necessary because of * range check and shifting -> no bits overlapping * the marker should be set.
45519. **comment:** * Object.defineProperty() related helper (E5 Section 15.2.3.6) * * Inlines all [[DefineOwnProperty]] exotic behaviors. * * Note: Ecmascript compliant [[DefineOwnProperty]](P, Desc, Throw) is not * implemented directly, but Object.defineProperty() serves its purpose. * We don't need the [[DefineOwnProperty]] internally and we don't have a * property descriptor with 'missing values' so it's easier to avoid it * entirely. * * Note: this is only called for actual objects, not primitive values. * This must support virtual properties for full objects (e.g. Strings) * but not for plain values (e.g. strings). Lightfuncs, even though * primitive in a sense, are treated like objects and accepted as target * values.
45520. callstack limits
45521. A...P
45522. Module table: * - module.exports: initial exports table (may be replaced by user) * - module.id is non-writable and non-configurable, as the CommonJS * spec suggests this if possible * - module.filename: not set, defaults to resolved ID if not explicitly * set by modSearch() (note capitalization, not .fileName, matches Node.js) * - module.name: not set, defaults to last component of resolved ID if * not explicitly set by modSearch()
45523. String constructor needs to distinguish between an argument not given at all * vs. given as 'undefined'. We're a vararg function to handle this properly.
45524. Explicit length is only needed if it differs from 'nargs'.
45525. Force restart caused by a function return; must recheck * debugger breakpoints before checking line transitions, * see GH-303. Restart and then handle interrupt_counter * zero again.
45526. Decode 'bits' bits from the input stream (bits must be 1...24). * When reading past bitstream end, zeroes are shifted in. The result * is signed to match duk_bd_decode_flagged.
45527. (?:
45528. Must be restored here to handle e.g. yields properly.
45529. The base ID of the current require() function, resolution base
45530. non-Reference deletion is always 'true', even in strict mode
45531. jump to next loop, using reg_v34_iter as iterated value
45532. Scan error: this shouldn't normally happen; it could happen if * string is not valid UTF-8 data, and clen/blen are not consistent * with the scanning algorithm.
45533. max possible
45534. punctuators (unlike the spec, also includes "/" and "=")
45535. DUK_BUFOBJ_FLOAT32ARRAY
45536. [...] ptr]
45537. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property delete right now.
45538. 12: getDate
45539. * Helper for valstack space * * Caller of DUK_ASSERT_VALSTACK_SPACE() estimates the number of free stack entries * required for its own use, and any child calls which are not (a) Duktape API calls * or (b) Duktape calls which involve extending the valstack (e.g. getter call).
45540. 'modLoaded'
45541. * Assertions before
45542. **comment:** * Buffer writer (dynamic buffer only) * * Helper for writing to a dynamic buffer with a concept of a "spare" area * to reduce resizes. You can ensure there is enough space beforehand and * then write for a while without further checks, relying on a stable data * pointer. Spare handling is automatic so call sites only indicate how * much data they need right now. * * There are several ways to write using bufwriter. The best approach * depends mainly on how much performance matters over code footprint. * The key issues are (1) ensuring there is space and (2) keeping the * pointers consistent. Fast code should ensure space for multiple writes * with one ensure call. Fastest inner loop code can temporarily borrow * the 'p' pointer but must write it back eventually. * * Be careful to ensure all macro arguments (other than static pointers like * 'thr' and 'bw_ctx') are evaluated exactly once, using temporaries if * necessary (if that's not possible, there should be a note near the macro). * Buffer write arguments often contain arithmetic etc so this is * particularly important here.
45543. Ordering should not matter (E5 Section 11.8.5, step 3.a) but * preserve it just in case.
45544. for zero size, don't push anything on valstack
45545. Here we'd have the option of decoding unpadded base64 * (e.g. "xxxxyy" instead of "xxxxyy==". Currently not * accepted.
45546. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
45547. * Conceptually, we look for the identifier binding by starting from * 'env' and following to chain of environment records (represented * by the prototype chain). * * If 'env' is NULL, the current activation does not yet have an * allocated declarative environment record; this should be treated * exactly as if the environment record existed but had no bindings * other than register bindings. * * Note: we assume that with the DUK_HOBJECT_FLAG_NEWENV cleared * the environment will always be initialized immediately; hence * a NULL 'env' should only happen with the flag set. This is the * case for: (1) function calls, and (2) strict, direct eval calls.
45548. heap level "stash" object (e.g., various reachability roots)
45549. * For global or eval code this is straightforward. For functions * created with the Function constructor we only get the source for * the body and must manufacture the "function ..." part. * * For instance, for constructed functions (v8): * * > a = new Function("foo", "bar", "print(foo)"); * [Function] * > a.toString() * 'function anonymous(foo,bar){\nprint(foo)\n}' * * Similarly for e.g. getters (v8): * * > x = { get a(foo,bar) { print(foo); } } * { a: [Getter] } * > Object.getOwnPropertyDescriptor(x, 'a').get.toString() * 'function a(foo,bar) { print(foo); }'
45550. c recursion check
45551. XXX: optimize by adding the token numbers directly into the * always interned duk_hstring objects (there should be enough * flag bits free for that)?
45552. Continue parsing after padding, allows concatenated, * padded base64.
45553. Check that 'this' is a duk_hbufferobject and return a pointer to it * (NULL if not).
45554. idx_step is +1 for indexOf, -1 for lastIndexOf
45555. is a setter/getter
45556. env created above to stack top
45557. cannot be e.g. arguments exotic, since exotic 'traits' are mutually exclusive
45558. DUK_TOK_LNOT
45559. XXX: sufficient to check 'strict', assert for 'is function'
45560. check final character validity
45561. * Init lexer context

45562. * Inref and decref functions. ** Decref may trigger immediate refzero handling, which may free and finalize * an arbitrary number of objects. *

45563. Generate pc2line data for an instruction sequence, leaving a buffer on stack top.

45564. a dummy undefined value is pushed to make valstack * behavior uniform for caller

45565. idx_step is +1 for reduce, -1 for reduceRight

45566. PatternCharacter, all excluded characters are matched by cases above

45567. Note: -nargs alone would fail for nargs == 0, this is OK

45568. curr and desc are accessors

45569. * Throw the error in the resumed thread's context; the * error value is pushed onto the resumee valstack. ** Note: the callstack of the target may empty in this case * too (i.e. the target thread has never been resumed). The * value stack will contain the initial function in that case, * which we simply ignore.

45570. For string-to-number, pretend we never have the lowest mantissa as there * is no natural "precision" for inputs. Having lowest_mantissa == 0, we'll * fall into the base cases for both e >= 0 and e < 0.

45571. Number built-in accepts a plain number or a Number object (whose * internal value is operated on). Other types cause TypeError.

45572. Encode to extended UTF-8; 'out' must have space for at least * DUK_UNICODE_MAX_XUTF8_LENGTH bytes. Allows encoding of any * 32-bit (unsigned) codepoint.

45573. func.prototype.constructor = func

45574. Accept any pointer-like value; for 'object' dvalue, read * and ignore the class number.

45575. **comment:** Clone the properties of the ROM-based global object to create a * fully RAM-based global object. Uses more memory than the inherit * model but more compliant.
label: code-design

45576. prev value can be garbage, no decref

45577. **comment:** * For/for-in statement is complicated to parse because * determining the statement type (three-part for vs. a * for-in) requires potential backtracking. ** See the helper for the messy stuff.
label: code-design

45578. Stepped? Step out is handled by callstack unwind.

45579. Copy values, the copy method depends on the arguments. ** Copy mode decision may depend on the validity of the underlying * buffer of the source argument; there must be no harmful side effects * from there to here for copy_mode to still be valid.

45580. fills window

45581. Caller must check character offset to be inside the string.

45582. valstack will be unbalanced, which is OK

45583. conservative

45584. unconditional

45585. DUK_USE_FASTINT && DUK_USE_PACKED_TVAL

45586. * NEXTEXNUM checks whether the enumerator still has unenumerated * keys. If so, the next key is loaded to the target register * and the next instruction is skipped. Otherwise the next instruction * will be executed, jumping out of the enumeration loop.

45587. **comment:** Providing access to e.g. act->lex_env would be dangerous: these * internal structures must never be accessible to the application. * Duktape relies on them having consistent data, and this consistency * is only asserted for, not checked for.
label: code-design

45588. Duktape.modSearch

45589. Set .buffer to the argument ArrayBuffer.

45590. * Parse a function-like expression: ** - function expression * - function declaration * - function statement (non-standard) * - setter/getter ** Adds the function to comp_ctx->curr_func function table and returns the * function number. ** On entry, curr_token points to: ** - the token after 'set' or 'get' for setter/getter expression/declaration/statement * - the token after 'set' or 'get' for setter/getter

45591. * Match loop. ** Try matching at different offsets until match found or input exhausted.

45592. flags: ""

45593. -> [res_obj]

45594. unvalidated

45595. nbytes * <-----> * [... | p | x | x | q] * => [... | q | p | x | x]

45596. eat '{' on entry

45597. This native function is also used for Date.prototype.getTime() * as their behavior is identical.

45598. resume write to target

45599. if new_size < L * old_size, resize without abandon check; L = 3-bit fixed point, e.g. 9 -> 9/8 = 112.5%

45600. Note: we could try to stuff a partial hash (e.g. 16 bits) into the * shared heap header. Good hashing needs more hash bits though.

45601. Else functionality is identical for function call and constructor * call.

45602. inclusive

45603. Check that 'this' is a duk_hbufferobject and return a pointer to it.

45604. utf-8 continuation bytes have the form 10xx xxxx

45605. This is based on V8 EquivalentYear() algorithm (see src/genequivyear.py): * <http://code.google.com/p/v8/source/browse/trunk/src/date.h#146>

45606. 'buf' contains the string to write, 'sz_buf' contains the length * (which may be zero).

45607. 'L'

45608. hobject management functions

45609. Convert day from one-based to zero-based (internal). This may * cause the day part to be negative, which is OK.

45610. 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [26 bits]

45611. There is no need to check the first character specially here * (i.e. reject digits): the caller only accepts valid initial * characters and won't call us if the first character is a digit. * This also ensures that the plain string won't be empty.

45612. '\xffBytecode'

45613. Catch a double-to-int64 cast issue encountered in practice.

45614. * DumpHeap command

45615. **comment:** * Lexer for source files, ToNumber() string conversions, RegExp expressions, * and JSON. ** Provides a stream of Ecmascript tokens from an UTF-8/CESU-8 buffer. The * caller can also rewind the token stream into a certain position which is * needed by the compiler part for multi-pass scanning. Tokens are * represented as duk_token structures, and contain line number information. * Token types are identified with DUK_TOK_* defines. ** Characters are decoded into a fixed size lookup window consisting of * decoded Unicode code points, with window positions past the end of the * input filled with an invalid codepoint (-1). The tokenizer can thus * perform multiple character lookups efficiently and with few sanity * checks (such as access outside the end of the input), which keeps the * tokenization code small at the cost of performance. ** Character data in tokens, such as identifier names and string literals, * is encoded into CESU-8 format on-the-fly while parsing the token in * question. The string data is made reachable to garbage collection by * placing the token-related values in value stack entries allocated for * this purpose by the caller. The characters exist in Unicode code point * form only in the fixed size lookup window, which keeps character data * expansion (of especially ASCII data) low. ** Token parsing supports the full range of Unicode characters as described * in the E5 specification. Parsing has been optimized for ASCII characters * because ordinary Ecmascript code consists almost entirely of ASCII * characters. Matching of complex Unicode codepoint sets (such as in the * IdentifierStart and IdentifierPart productions) is optimized for size, * and is done using a linear scan of a bit-packed list of ranges. This is * very slow, but should never be entered unless the source code actually * contains Unicode characters. ** Ecmascript tokenization is partially context sensitive. First, * additional future reserved words are recognized in strict mode (see E5 * Section 7.6.1.2). Second, a forward slash character ('/') can be * recognized either as starting a RegExp literal or as a division operator, * depending on context. The caller must provide necessary context flags * when requesting a new token. ** Future work: * * * Make line number tracking optional, as it consumes space. * * * Add a feature flag for disabling UTF-8 decoding of input, as most * source code is ASCII. Because of Unicode escapes written in ASCII, * this does not allow Unicode support to be removed from e.g. * duk_unicode_is_identifier_start() nor does it allow removal of CESU-8 * encoding of e.g. string literals. * * * Add a feature flag for disabling Unicode compliance of e.g. identifier * names. This allows for a build more than a kilobyte smaller, because * Unicode ranges needed by duk_unicode_is_identifier_start() and * duk_unicode_is_identifier_part() can be dropped. String literals * should still be allowed to contain escaped Unicode, so this still does * not allow removal of CESU-8 encoding of e.g. string literals. * * * Character lookup tables for codepoints above BMP could be stripped. * * * Strictly speaking, E5 specification requires that source code consists * of 16-bit code units, and if not, must be conceptually converted to * that format first. The current lexer processes Unicode code points * and allows characters outside the

BMP. These should be converted to * surrogate pairs while reading the source characters into the window, * not after tokens have been formed (as is done now). However, the fix * is not trivial because two characters are decoded from one codepoint. *** Optimize for speed as well as size. Large if-else ladders are (at * least potentially) slow.

label: code-design

45616. [... buf loop]

45617. **comment:** * Heap object representation. ** Heap objects are used for Ecmascript objects, arrays, and functions, * but also for internal control like declarative and object environment * records. Compiled functions, native functions, and threads are also * objects but with an extended C struct. ** Objects provide the required Ecmascript semantics and exotic behaviors * especially for property access. ** Properties are stored in three conceptual parts: ** 1. A linear 'entry part' contains ordered key-value-attributes triples * and is the main method of string properties. ** 2. An optional linear 'array part' is used for array objects to store a * (dense) range of [0,N[array indexed entries with default attributes * (writable, enumerable, configurable). If the array part would become * sparse or non-default attributes are required, the array part is * abandoned and moved to the 'entry part'. ** 3. An optional 'hash part' is used to optimize lookups of the entry * part; it is used only for objects with sufficiently many properties * and can be abandoned without loss of information. ** These three conceptual parts are stored in a single memory allocated area. * This minimizes memory allocation overhead but also means that all three * parts are resized together, and makes property access a bit complicated.

label: code-design

45618. $x \leftarrow y + z$

45619. get array index related to string (or return DUK_HSTRING_NO_ARRAY_INDEX); * avoids helper call if string has no array index value.

45620. [... re_obj input bc]

45621. Keep the error as the result (coercing it might fail below, * but we don't catch that now).

45622. * Note: although arguments object variable mappings are only established * for non-strict functions (and a call to a non-strict function created * the arguments object in question), an inner strict function may be doing * the actual property write. Hence the throw_flag applied here comes from * the property write call.

45623. push before advancing to keep reachable

45624. * duk_hthread allocation and freeing.

45625. [... pattern flags escaped_source bytecode]

45626. * Helper for updating callee 'caller' property.

45627. eat 'do'

45628. saves a few instructions to have this wrapper (see comment on duk_heap_mem_alloc)

45629. [source template result]

45630. 25 user flags

45631. also stash it before constructor, * in case we need it (as the fallback value)

45632. omit print

45633. Unary plus is used to force a fastint check, so must include * downgrade check.

45634. DUK_FLD_16BIT

45635. **comment:** should never happen, but be robust

label: code-design

45636. _Formals: omitted if function is guaranteed not to need a (non-strict) arguments object

45637. **comment:** XXX: comes out as signed now

label: code-design

45638. flags

45639. **comment:** If message is undefined, the own property 'message' is not set at * all to save property space. An empty message is inherited anyway.

label: code-design

45640. writable but not deletable

45641. Decode UTF-8 codepoint from a sequence of hex escapes. The * first byte of the sequence has been decoded to 't'. ** Note that UTF-8 validation must be strict according to the * specification: E5.1 Section 15.1.3, decode algorithm step * 4.d.vii.8. URIError from non-shortest encodings is also * specifically noted in the spec.

45642. [... formals]

45643. **comment:** avoid tag 0xffff0, no risk of confusion with negative infinity

label: code-design

45644. See: tests/ecmascript/test-spec-bound-constructor.js

45645. guaranteed to succeed

45646. * Define new property * * Note: this may fail if the holder is not extensible.

45647. used elements (includes DELETED)

45648. **comment:** XXX: since the enumerator may be a memory expensive object, * perhaps clear it explicitly here? If so, break jump must * go through this clearing operation.

label: code-design

45649. \rightarrow [... key val replacer holder key]

45650. as is

45651. Step 1 is not necessary because duk_call_method() will take * care of it.

45652. * Code emission helpers * * Some emission helpers understand the range of target and source reg/const * values and automatically emit shuffling code if necessary. This is the * case when the slot in question (A, B, C) is used in the standard way and * for opcodes the emission helpers explicitly understand (like DUK_OP_CALL). ** The standard way is that: * - slot A is a target register * - slot B is a source register/constant * - slot C is a source register/constant * * If a slot is used in a non-standard way the caller must indicate this * somehow. If a slot is used as a target instead of a source (or vice * versa), this can be indicated with a flag to trigger proper shuffling * (e.g. DUK_EMIT_FLAG_B_IS_TARGET). If the value in the slot is not * register/const related at all, the caller must ensure that the raw value * fits into the corresponding slot so as to not trigger shuffling. The * caller must set a "no shuffle" flag to ensure compilation fails if * shuffling were to be triggered because of an internal error. ** For slots B and C the raw slot size is 9 bits but one bit is reserved for * the reg/const indicator. To use the full 9-bit range for a raw value, * shuffling must be disabled with the DUK_EMIT_FLAG_NO_SHUFFLE_{B,C} flag. * Shuffling is only done for A, B, and C slots, not the larger BC or ABC slots. ** There is call handling specific understanding in the A-B-C emitter to * convert call setup and call instructions into indirect ones if necessary.

45653. **comment:** XXX: make this an internal helper

label: code-design

45654. assumed to bottom relative

45655. intentionally empty

45656. Compute time value from (double) parts. The parts can be either UTC * or local time; if local, they need to be (conceptually) converted into * UTC time. The parts may represent valid or invalid time, and may be * wildly out of range (but may cancel each other and still come out in * the valid Date range).

45657. aaaaabbbcccccc cccccccccc

45658. [[SetPrototypeOf]] standard behavior, E6 9.1.2

45659. **comment:** XXX: could unwind catchstack here, so that call handling * didn't need to do that?

label: code-design

45660. 27: setUTCSeconds

45661. Lightfunc virtual properties are non-configurable, so * reject if match any of them.

45662. **comment:** init is unnecessary but suppresses "may be used uninitialized" warnings

label: code-design

45663. no need to unwind catchstack

45664. 'Proxy' object

45665. 'errCreate'

45666. coerce lval with ToString()

45667. see algorithm

45668. * reverse()

45669. index is above internal buffer length -> property is fully normal
45670. ensure never re-entered until rescue cycle complete
45671. **comment:** order: most frequent to least frequent
 label: code-design
45672. **comment:** not very useful, used for debugging
 label: code-design
45673. catch stack depth
45674. [... res_obj]
45675. * Lexer defines.
45676. A token value. Can be memcpy()'d, but note that slot1/slot2 values are on the valstack. * Some fields (like num, str1, str2) are only valid for specific token types and may have * stale values otherwise.
45677. DUK_BUFOBJ_NODEJS_BUFFER
45678. XXX: if 'len' is low, may want to ensure array part is kept: * the caller is likely to want a dense array.
45679. XXX: version specific array format instead?
45680. * Process incoming debug requests ** Individual request handlers can push temporaries on the value stack and * rely on duk_debug_process_message() to restore the value stack top * automatically.
45681. roughly 2 kB
45682. Must avoid duk_pop() in exit path
45683. basic types from duk_features.h
45684. XXX: byte offset
45685. * instanceof
45686. ecmascript compiler limits
45687. token type
45688. -> [... key val replacer]
45689. call_flags
45690. * Comparisons (x >= y, x > y, x <= y, x < y) ** E5 Section 11.8.5: implement 'x < y' and then use negate and eval_left_first * flags to get the rest.
45691. Negative indices are always within allocated stack but * must not go below zero index.
45692. Executor interrupt default interval when nothing else requires a * smaller value. The default interval must be small enough to allow * for reasonable execution timeout checking but large enough to keep * impact on execution performance low.
45693. * Object internal value ** Returned value is guaranteed to be reachable / incref'd, caller does not need * to incref OR decref. No proxies or accessors are invoked, no prototype walk.
45694. Exponent
45695. general temp register
45696. 0xe0...0xef
45697. Maximum write size: XUTF8 path writes max DUK_UNICODE_MAX_XUTF8_LENGTH, * percent escape path writes max two times CESU-8 encoded BMP length.
45698. [arg1 ... argN this loggerLevel loggerName buffer]
45699. If not present in finalize_list or refzero_list, the pointer * must be either in heap_allocated or the string table.
45700. rc_varname and reg_varbind are ignored here
45701. duplicate zeroing, expect for (possible) NULL inits
45702. get as double to handle huge numbers correctly
45703. enum_index
45704. XXX: clarify on when and where DUK_CONST_MARKER is allowed
45705. * Error helpers
45706. * Data following the header depends on the DUK_HBUFFER_FLAG_DYNAMIC * flag. ** If the flag is clear (the buffer is a fixed size one), the buffer * data follows the header directly, consisting of 'size' bytes. ** If the flag is set, the actual buffer is allocated separately, and * a few control fields follow the header. Specifically: ** - a "void **" pointing to the current allocation * - a duk_size_t indicating the full allocated size (always >= 'size') ** If DUK_HBUFFER_FLAG_EXTERNAL is set, the buffer has been allocated * by user code, so that Duktape won't be able to resize it and won't * free it. This allows buffers to point to e.g. an externally * allocated structure such as a frame buffer. ** Unlike strings, no terminator byte (NUL) is guaranteed after the * data. This would be convenient, but would pad aligned user buffers * unnecessarily upwards in size. For instance, if user code requested * a 64-byte dynamic buffer, 65 bytes would actually be allocated which * would then potentially round upwards to perhaps 68 or 72 bytes.
45707. 'new'
45708. evaluate to plain value, no forced register (temp/bound reg both ok)
45709. Keep heap->finalize_list up-to-date during the list walk. * This has no functional impact, but does matter e.g. for * duk_push_heapptr() asserts when assertions are enabled.
45710. DUK_USE_INTERRUPT_COUNTER
45711. Fast path is triggered for no exponent and also for balanced exponent * and fraction parts, e.g. for "1.23e2" == "123". Remember to respect * zero sign.
45712. Note: copying tv_obj and tv_key to locals to shield against a valstack * resize is not necessary for a property put right now (putprop protects * against it internally).
45713. **comment:** * Parser control values for tokens. The token table is ordered by the * DUK_TOK_XXX defines. ** The binding powers are for lbp() use (i.e. for use in led() context). * Binding powers are positive for typing convenience, and bits at the * top should be reserved for flags. Binding power step must be higher * than 1 so that binding power "lbp - 1" can be used for right associative * operators. Currently a step of 2 is used (which frees one more bit for * flags).
 label: code-design
45714. * Array part ** Note: ordering between array and entry part must match 'abandon array' * behavior in duk_hobject_props.c: key order after an array is abandoned * must be the same.
45715. **comment:** XXX: could be eliminated with valstack adjust
 label: code-design
45716. [key val]
45717. DUK_TOK_CATCH
45718. Note: we rely on duk_safe_call() to fix up the stack for the caller, * so we don't need to pop stuff here. There is no return value; * caller determines rescued status based on object refcount.
45719. * Case conversion helper, with context/local sensitivity. * For proper case conversion, one needs to know the character * and the preceding and following characters, as well as * locale/language.
45720. [... arg1 ... argN envobj argobj]
45721. flags:nonstrict
45722. pass2 allocation handles this
45723. * Call the wrapped module function. ** Use a protected call so that we can update Duktape.modLoaded[resolved_id] * even if the module throws an error.
45724. advance manually
45725. **comment:** XXX: awkward
 label: code-design
45726. target out-of-bounds (but positive)
45727. free some memory
45728. constrained by string length
45729. stable; precalculated for faster lookups
45730. * Thread defines
45731. same handling for identifiers and strings
45732. -> [... lval rval rval.prototype]
45733. 'implements'
45734. '='

45735. XXX: Not sure what the best return value would be in the API. * Return a boolean for now. Note that rc == 0 is success (true).
45736. Current convention is to use duk_size_t for value stack sizes and global indices, * and duk_idx_t for local frame indices.
45737. **comment:** XXX: where to release temp regs in intermediate expressions? * e.g. 1+2+3 -> don't inflate temp register count when parsing this. * that particular expression temp regs can be forced here.
label: code-design
45738. negative top-relative indices not allowed now
45739. 'instanceof'
45740. avoid attempt to compact any objects
45741. expression parsing helpers
45742. return ToObject(this)
45743. one token
45744. flags field: LLLLLLFT, L = label (24 bits), F = flags (4 bits), T = type (4 bits)
45745. not protected, respect reclimit, is a constructor call
45746. DUK_TOK_BNOT
45747. E5 Section 8.6.1
45748. indicates a deleted string; any fixed non-NULL, non-hstring pointer works
45749. val is unsigned so >= 0
45750. **comment:** NOTE: we try to minimize code size by avoiding unnecessary pops, * so the stack looks a bit cluttered in this function. DUK_ASSERT_TOP() * assertions are used to ensure stack configuration is correct at each * step.
label: code-design
45751. must have start and end
45752. Check for breakpoints only on line transition. * Breakpoint is triggered when we enter the target * line from a different line, and the previous line * was within the same function. * * This condition is tricky: the condition used to be * that transition to -or across- the breakpoint line * triggered the breakpoint. This seems intuitively * better because it handles breakpoints on lines with * no emitted opcodes; but this leads to the issue * described in: <https://github.com/svaaraal/duktape/issues/263>.
45753. env and act may be NULL
45754. * Process yield * * After longjmp(), processing continues in bytecode executor longjmp * handler, which will e.g. update thr->resumer to NULL.
45755. -> [... enum key enum_target key]
45756. **comment:** XXX: store 'bcode' pointer to activation for faster lookup?
label: code-design
45757. * Unicode tables
45758. [... obj]
45759. Note: space must cater for both JX and JC.
45760. Needs to be inserted; scan backwards, since we optimize * for the case where elements are nearly in order.
45761. Normal non-bound function.
45762. Eating a left curly; regexp mode is allowed by left curly * based on duk_token_lbp[] automatically.
45763. A validated read() is always a number, so it's write coercion * is always side effect free and won't invalidate pointers etc.
45764. current and previous token for parsing
45765. probe sequence
45766. message is empty -> return name
45767. NaN timevalue: we need to coerce the arguments, but * the resulting internal timestamp needs to remain NaN. * This works but is not pretty: parts and dparts will * be partially uninitialized, but we only write to them.
45768. **comment:** It might seem that replacing 'thr->heap' with just 'heap' below * might be a good idea, but it increases code size slightly * (probably due to unnecessary spilling) at least on x64.
label: code-design
45769. Currently nothing to free; 'data' is a heap object
45770. 'prototype'
45771. 'jx'
45772. default prototype (Note: 'obj' must be reachable)
45773. **comment:** XXX: currently NULL allocations are not supported; remove if later allowed
label: code-design
45774. Unwind the topmost callstack entry before reusing it
45775. internal define property: skip write silently if exists
45776. borrowed, no refcount
45777. allow e.g. '0x0009' and '00077'
45778. DUK_BUFOBJ_DATAVIEW
45779. * Compact an object. Minimizes allocation size for objects which are * not likely to be extended. This is useful for internal and non- * extensible objects, but can also be called for non-extensible objects. * May abandon the array part if it is computed to be too sparse. * * This call is relatively expensive, as it needs to scan both the * entries and the array part. * * The call may fail due to allocation error.
45780. find elements to swap
45781. currently, always the case
45782. * Detected label
45783. **comment:** XXX: an array can have length higher than 32 bits; this is not handled * correctly now.
label: code-design
45784. * Helper macros
45785. **comment:** XXX: Simplify this algorithm, should be possible to come up with * a shorter and faster algorithm by inspecting IEEE representation * directly.
label: code-design
45786. * CheckObjectCoercible() (E5 Section 9.10) * * Note: no API equivalent now.
45787. constants are strings or numbers now
45788. '\n'
45789. Must behave like a no-op with NULL and any pointer returned from * malloc/realloc with zero size.
45790. if (is_regexp)
45791. [thisArg arg1 ... argN func boundFunc]
45792. -Infinity
45793. Undefine local defines
45794. * Top-level include file to be used for all (internal) source files. * * Source files should not include individual header files, as they * have not been designed to be individually included.
45795. * Init built-in strings
45796. * Two's complement arithmetic.
45797. [... constructor arg1 ... argN]
45798. no need to decref previous value
45799. 'finally'
45800. Zero size compare not an issue with DUK_MEMCMP.
45801. [... closure template val]
45802. YIELDED: Ecmascript activation + Duktape.Thread.yield() activation
45803. may be NULL (but only if size is 0)
45804. write to entry part

45805. * Ecmascript bytecode executor. ** Resume execution for the current thread from its current activation. * Returns when execution would return from the entry level activation, * leaving a single return value on top of the stack. Function calls * and thread resumptions are handled internally. If an error occurs, * a longjmp() with type DUK_LJ_TYPE_THROW is called on the entry level * setjmp() jmpbuf. ** Ecmascript function calls and coroutine resumptions are handled * internally (by the outer executor function) without recursive C calls. * Other function calls are handled using duk_handle_call(), increasing * C recursion depth. * * Abrupt completions (= long control transfers) are handled either * directly by reconfiguring relevant stacks and restarting execution, * or via a longjmp. Longjmp-free handling is preferable for performance * (especially Emscripten performance), and is used for: break, continue, * and return. ** For more detailed notes, see doc/execution.rst. ** Also see doc/code-issues.rst for discussion of setjmp(), longjmp(), * and volatile.

45806. * Debug connection message write helpers

45807. We know _Formals is dense and all entries will be in the * array part. GC and finalizers shouldn't affect _Formals * so side effects should be fine.

45808. don't want an intermediate exposed state with func == NULL

45809. **comment:** * (Re)try handling the longjmp. ** A longjmp handler may convert the longjmp to a different type and * "virtually" rethrow by goto'ing to 'check_longjmp'. Before the goto, * the following must be updated: * - the heap 'lj' state * - 'thr' must reflect the "throwing" thread
label: code-design

45810. XXX=

45811. separators: space, space, colon

45812. **comment:** Format of magic, bits: * 0...1: field type; 0:uint8, 1:uint16, 2:uint32, 3=float, 4=double, 5=unused, 6=unused, 7=unused * 3: endianness: 0=little, 1=big * 4: signed: 1=yes, 0=no * 5: typedarray: 1=yes, 0=no
label: code-design

45813. **comment:** Note: reuse 'curr' as a temp propdesc
label: code-design

45814. Slow path

45815. Lightweight function: never bound, so terminate.

45816. unconditionally; is_global==0

45817. Node.js Buffer: return offset + #bytes written (i.e. next * write offset).

45818. string limits

45819. **comment:** function makes one or more slow path accesses
label: code-design

45820. DUK_USE_AUGMENT_ERROR_CREATE

45821. No need for a NULL/zero-size check because new_size > 0)

45822. Assume value stack sizes (in elements) fits into duk_idx_t.

45823. **comment:** Relookup in case duk_js_tointeger() ends up e.g. coercing an object.
label: code-design

45824. '>'

45825. * Debug connection skip primitives

45826. valstack index of 'func' and retval (relative to entry valstack_bottom)

45827. XXX: array internal?

45828. Assertion compatible inside a comma expression, evaluates to void. * Currently not compatible with DUK_USE_PANIC_HANDLER() which may have * a statement block.

45829. avoid key quotes when key is an ASCII Identifier

45830. embed: nothing

45831. **comment:** bit mask which indicates that a regconst is a constant instead of a register
label: code-design

45832. * Memory allocation handling.

45833. current assertion is quite strong: decref's and set to unused

45834. Coerce an duk_ivalue to a 'plain' value by generating the necessary * arithmetic operations, property access, or variable access bytecode. * The duk_ivalue argument ('x') is converted into a plain value as a * side effect.

45835. element type

45836. parsing in "scanning" phase (first pass)

45837. regexp res_obj is at offset 4

45838. Flags for duk_js_compare_helper().

45839. **comment:** * Eval ** Eval needs to handle both a "direct eval" and an "indirect eval". * Direct eval handling needs access to the caller's activation so that its * lexical environment can be accessed. A direct eval is only possible from * Ecmascript code; an indirect eval call is possible also from C code. * When an indirect eval call is made from C code, there may not be a * calling activation at all which needs careful handling.
label: code-design

45840. no fractions

45841. 'Function'

45842. 29: setUTCMinutes

45843. * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written.

45844. accept comma, expect new value

45845. -> [... reviver holder name val]

45846. [... pattern flags escaped_source buffer]

45847. reg

45848. empty match -> bump and continue

45849. regexp execution limits

45850. SCANBUILD: "Dereference of null pointer", normal

45851. catch or with binding is currently active

45852. Note: unbalanced stack on purpose

45853. Get a borrowed duk_tval pointer to the current 'this' binding. Caller must * make sure there's an active callstack entry. Note that the returned pointer * is unstable with regards to side effects.

45854. expt values [0x001,0x7fe] = normal

45855. Receiver: Proxy object

45856. **comment:** * Dump/load helpers, xxx_raw() helpers do no buffer checks
label: code-design

45857. **comment:** * Push readable string summarizing duk_tval. The operation is side effect * free and will only throw from internal errors (e.g. out of memory). * This is used by e.g. property access code to summarize a key/base safely, * and is not intended to be fast (but small and safe).
label: code-design

45858. * Type error thrower, E5 Section 13.2.3.

45859. **comment:** Corner case: see test-numconv-parse-mant-carry.js. We could * just bump the exponent and update bitstart, but it's more robust * to recompute (but avoid rounding twice).
label: code-design

45860. The JA(value) operation: encode array. ** Stack policy: [array] -> [array].

45861. XXX: init or assert catch depth etc -- all values

45862. happens when hash part dropped

45863. 1x heap size

45864. no need for decref/incref because value is a number

45865. **comment:** XXX: this may now fail, and is not handled correctly
label: code-design

45866. [key undefined] -> [key]

45867. * join(), toLocaleString() ** Note: checking valstack is necessary, but only in the per-element loop. ** Note: the trivial approach of pushing all the elements on the value stack * and then calling duk_join() fails when the array contains a large number * of elements. This problem can't be offloaded to duk_join() because the * elements to join must be handled here and have special handling. Current * approach is to do intermediate joins with very large number of elements. * There is no fancy handling; the prefix gets re-joined multiple times.

45868. stack top contains 'false'

45869. LAYOUT 2

45870. If 16-bit hash is in use, stuff it into duk_heapHdr_string flags.

45871. * Other cases, use C recursion. ** If a tail call was requested we ignore it and execute a normal call. * Since Duktape 0.11.0 the compiler emits a RETURN opcode even after * a tail call to avoid test-bug-tailcall-thread-yield-resume.js. ** Direct eval call: (1) call target (before following bound function * chain) is the built-in eval() function, and (2) call was made with * the identifier 'eval'.

45872. (

45873. NOTE! Caller must ensure that any side effects from the * coercions below are safe. If that cannot be guaranteed * (which is normally the case), caller must coerce the * argument using duk_to_number() before any pointer * validations; the result of duk_to_number() always coerces * without side effects here.

45874. **comment:** Compared to duk_handle_call(): * - protected call: never * - ignore recursion limit: never
label: code-design

45875. Note: we ask for one return value from duk_safe_call to get this * error debugging here.

45876. -> [sep ToObject(this) len str]

45877. covered by comparison

45878. Setting "no shuffle A" is covered by the assert, but it's needed * with DUK_USE_SHUFFLE_TORTURE.

45879. **comment:** XXX: .code = err_code disabled, not sure if useful
label: code-design

45880. For now, restrict result array into 32-bit length range.

45881. coerce in-place

45882. DUK_ERR_ASSERTION_ERROR: no macros needed

45883. directive prologue status at entry

45884. * ToNumber() (E5 Section 9.3) ** Value to convert must be on stack top, and is popped before exit. ** See: <http://www.cs.indiana.edu/~burger/FP-Printing-PLDI96.pdf> * http://www.cs.indiana.edu/~burger/fp/index.html ** Notes on the conversion: ** - There are specific requirements on the accuracy of the conversion * through a "Mathematical Value" (MV), so this conversion is not * trivial. ** - Quick rejects (e.g. based on first char) are difficult because * the grammar allows leading and trailing white space. ** - Quick reject based on string length is difficult even after * accounting for white space; there may be arbitrarily many * decimal digits. ** - Standard grammar allows decimal values ("123"), hex values * ("0x123") and infinities * * - Unlike source code literals, ToNumber() coerces empty strings * and strings with only whitespace to zero (not NaN).

45885. [... num]

45886. arrays MUST have a 'length' property

45887. Using an explicit 'ASCII' flag has larger footprint (one call site * only) but is quite useful for the case when there's no explicit * 'clen' in duk_hstring.

45888. **comment:** XXX: try to optimize to 8 (would now be possible, <200 used)
label: code-design

45889. * Arguments handling helpers (argument map mainly). ** An arguments object has exotic behavior for some numeric indices. * Accesses may translate to identifier operations which may have * arbitrary side effects (potentially invalidating any duk_tval * pointers).

45890. Undefined/null are considered equal (e.g. "null == undefined" -> true).

45891. * String value of 'blen+1' bytes follows (+1 for NUL termination * convenience for C API). No alignment needs to be guaranteed * for strings, but fields above should guarantee alignment-by-4 * (but not alignment-by-8).

45892. **comment:** XXX: Could be improved by coercing to a readable duk_tval (especially string escaping)
label: code-design

45893. **comment:** Inner executor, performance critical.
label: code-design

45894. significant from precision perspective

45895. [...] Logger clog logfunc clog]

45896. No assertions for offset or length; in particular, \ * it's OK for length to be longer than underlying \ * buffer. Just ensure they don't wrap when added. \

45897. fits into duk_small_int_t

45898. Caller doesn't need to check exotic proxy behavior (but does so for * some fast paths).

45899. is eval code

45900. * Iterate the bitstream (line diffs) until PC is reached

45901. varmap is already in comp_ctx->curr_func.varmap_idx

45902. * Convert binary digits into an IEEE double. Need to handle * denormals and rounding correctly.

45903. [key setter this val] -> [key retval]

45904. DUK_TOK_LAND

45905. **comment:** * E5 Section 15.3.4.2 places few requirements on the output of * this function: ** - The result is an implementation dependent representation * of the function; in particular ** - The result must follow the syntax of a FunctionDeclaration. * In particular, the function must have a name (even in the * case of an anonymous function or a function with an empty * name). ** - Note in particular that the output does NOT need to compile * into anything useful.
label: code-design

45906. Note that multiple catchstack entries may refer to the same * callstack entry.

45907. (-Number.MAX_VALUE).toString(2).length == 1025, + spare

45908. DUK_HNATIVEFUNCTION_H_INCLUDED

45909. check pointer at end

45910. **comment:** XXX: any way to avoid decoding magic bit; there are quite * many function properties and relatively few with magic values.
label: code-design

45911. chain jumps for 'fall-through' * after a case matches.

45912. duk_unicode_ids_noa[]

45913. **comment:** XXX: error message is a bit misleading: we reached a recursion * limit which is also essentially the same as a C callstack limit * (except perhaps with some relaxed threading assumptions).
label: code-design

45914. reset longjmp

45915. non-strict equality for buffers compares contents

45916. all codepoints up to U+FFFF

45917. recursion tracking happens here only

45918. roughly 128 bytes

45919. default clause matches next statement list (if any)

45920. * E5 Sections 11.8.7, 8.12.6. ** Basically just a property existence check using [[HasProperty]].

45921. new value

45922. MULTIPLICATIVE EXPRESSION

45923. **comment:** * typeof * * E5 Section 11.4.3. ** Very straightforward. The only question is what to return for our * non-standard tag / object types. ** There is an unfortunate string constant define naming problem with * typeof return values for e.g. "Object" and "object"; careful with * the built-in string defines. The LC_XXX defines are used for the * lowercase variants now.
label: code-design

45924. -> [... func this]

45925. **comment:** XXX: should this happen in the callee's activation or after unwinding?
label: code-design

45926. **comment:** * Four possible outcomes: ** 1. A 'finally' in the same function catches the 'return'. * It may continue to propagate when 'finally' is finished, * or it may be neutralized by 'finally' (both handled by * ENDFIN). ** 2. The return happens at the entry level of the bytecode * executor, so return from the executor (in C stack). ** 3. There is a calling (Ecmascript) activation in the call * stack => return to it, in the same executor instance. ** 4. There is no calling activation, and the thread is * terminated. There is always a resumer in this case, * which gets the return value similarly to a 'yield' * (except that the current thread can no longer be * resumed).
- label:** code-design
45927. caller checks
45928. t has high mantissa
45929. * Flags ** Fixed buffer: 0 * Dynamic buffer: DUK_HBUFFER_FLAG_DYNAMIC * External buffer: DUK_HBUFFER_FLAG_DYNAMIC | DUK_HBUFFER_FLAG_EXTERNAL
45930. DUK_TOK_LCURLY
45931. **comment:** * Identifier access and function closure handling. ** Provides the primitives for slow path identifier accesses: GETVAR, * PUTVAR, DELVAR, etc. The fast path, direct register accesses, should * be used for most identifier accesses. Consequently, these slow path * primitives should be optimized for maximum compactness. ** Ecmascript environment records (declarative and object) are represented * as internal objects with control keys. Environment records have a * parent record ("outer environment reference") which is represented by * the implicit prototype for technical reasons (in other words, it is a * convenient field). The prototype chain is not followed in the ordinary * sense for variable lookups. ** See identifier-handling.rst for more details on the identifier algorithms * and the internal representation. See function-objects.rst for details on * what function templates and instances are expected to look like. * Care must be taken to avoid duk_tval pointer invalidation caused by * e.g. value stack or object resizing. ** TODO: properties for function instances could be initialized much more * efficiently by creating a property allocation for a certain size and * filling in keys and values directly (and INCREFing both with "bulk incref" * primitives. ** XXX: duk_hobject_getprop() and duk_hobject_putprop() calls are a bit * awkward (especially because they follow the prototype chain); rework * if "raw" own property helpers are added.
- label:** code-design
45932. If 'day' is NaN, returns NaN.
45933. * New length not lower than old length => no changes needed * (not even array allocation).
45934. * API oriented helpers
45935. [enum_target res key true]
45936. **comment:** XXX: we could save space by using _Target OR _This. If _Target, assume * this binding is undefined. If _This, assumes this binding is _This, and * target is also _This. One property would then be enough.
- label:** code-design
45937. ADDITIVE EXPRESSION
45938. macros
45939. default case does not exist, or no statements present * after default case: finish case evaluation
45940. use temp_next for tracking register allocations
45941. stack[0] = search value * stack[1] = replace value * stack[2] = input string * stack[3] = result buffer
45942. DUK_REGEX_H_INCLUDED
45943. **comment:** XXX: add fastint support?
- label:** requirement
45944. FOUND
45945. * Additional control information is placed into the object itself * as internal properties to avoid unnecessary fields for the * majority of functions. The compiler tries to omit internal * control fields when possible. ** Function templates: ** { * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * } ** Function instances: ** { * length: 2, * prototype: { constructor: <func> }, * caller: <thrower>, * arguments: <thrower>, * name: "func", // declaration, named function expressions * fileName: <debug info for creating nice errors> * _Varmap: { "arg1": 0, "arg2": 1, "varname": 2 }, * _Formals: ["arg1", "arg2"], * _Source: "function func(arg1, arg2) { ... }", * _Pc2line: <debug info for pc-to-line mapping>, * _Varenv: <variable environment of closure>, * _Lexenv: <lexical environment of closure (if differs from _Varenv)> * } ** More detailed description of these properties can be found * in the documentation.
45946. initialize for debug prints, needed if sce==NULL
45947. start of valstack allocation
45948. Allocate a new duk_hbuffer of a certain type and return a pointer to it * (NULL on error). Write buffer data pointer to 'out_bufdata' (only if * allocation successful).
45949. Lookup 'key' from arguments internal 'map', delete mapping if found. * Used in E5 Section 10.6 algorithm for [[Delete]]. Note that the * variable/argument itself (where the map points) is not deleted.
45950. -> [... enum_target res trap handler target]
45951. derived types
45952. Awkward inclusion condition: drop out of compilation if not needed by any * call site: object hash part or probing stringtable.
45953. next instruction to execute (points to 'func' bytecode, stable pointer), NULL for native calls
45954. [offset littleEndian], when DUK_FLD_TYPEDARRAY (regardless of ftype)
45955. Must be a "pointer object", i.e. class "Pointer"
45956. ES6 proxy
45957. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array * stack[4] = regexp res_obj (if is_regexp)
45958. value of dbg_exec_counter when we last did a Date-based check
45959. idx_retval unsigned
45960. no negative sign for zero
45961. **comment:** Two value cycle, see e.g. test-bi-date-tzoffset-basic-fi.js. * In these cases, favor a higher tzoffset to get a consistent * result which is independent of iteration count. Not sure if * this is a generically correct solution.
- label:** code-design
45962. 'thr' is the current thread, as no-one resumes except us and we * switch 'thr' in that case.
45963. lastIndex must be ignored for non-global regexps, but get the * value for (theoretical) side effects. No side effects can * really occur, because lastIndex is a normal property and is * always non-configurable for RegExp instances.
45964. clamp anything above nargs
45965. **comment:** XXX: this could be more compact by accessing the internal properties * directly as own properties (they cannot be inherited, and are not * externally visible).
- label:** code-design
45966. Get/set the current user visible size, without accounting for a dynamic * buffer's "spare" (= usable size).
45967. looping should never happen
45968. Validate byte read/write for virtual 'offset', i.e. check that the * offset, taking into account h->offset, is within the underlying * buffer size. This is a safety check which is needed to ensure * that even a misconfigured duk_hbufferobject never causes memory * unsafe behavior (e.g. if an underlying dynamic buffer changes * after being setup). Caller must ensure 'buf' != NULL.
45969. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.
45970. digit position is absolute, not relative
45971. c
45972. **comment:** Same test with volatiles
- label:** test
45973. There's overlap: the desired end result is that * conceptually a copy is made to avoid "trampling" * of source data by destination writes. We make * an actual temporary copy to handle this case.
45974. **comment:** XXX: just use toString() for now; permitted although not recommended. * nargs==1, so radix is passed to toString().
- label:** code-design
45975. %d; only 16 bits are guaranteed

45976. 'this' value: * E5 Section 6.b.i ** The only (standard) case where the 'this' binding is non-null is when * (1) the variable is found in an object environment record, and * (2) that object environment record is a 'with' block. *

45977. This is important to ensure dynamic buffer data pointer is not * NULL (which is possible if buffer size is zero), which in turn * causes portability issues with e.g. memmove() and memcpy().

45978. Since we already started writing the reply, just emit nothing.

45979. * For top-level objects, 'length' property has the following * default attributes: non-writable, non-enumerable, non-configurable * (E5 Section 15). ** However, 'length' property for Array.prototype has attributes * expected of an Array instance which are different: writable, * non-enumerable, non-configurable (E5 Section 15.4.5.2). ** This is currently determined implicitly based on class; there are * no attribute flags in the init data.

45980. -> [... errhandler undefined(= this) errval]

45981. Reconfigure value stack for return to an Ecmascript function at 'act_idx'.

45982. * First pass. ** Gather variable/function declarations needed for second pass. * Code generated is dummy and discarded.

45983. valid pc range is [0, length[

45984. Current tracedata contains 2 entries per callstack entry.

45985. [... errval]

45986. **comment:** XXX: there is a small risk here: because the ISO 8601 parser is * very loose, it may end up parsing some datetime values which * would be better parsed with a platform specific parser.
label: code-design

45987. * Return (t - LocalTime(t)) in minutes: * * t - LocalTime(t) = t - (t + LocalTZA + DaylightSavingTA(t)) * = -(LocalTZA + DaylightSavingTA(t)) * * where DaylightSavingTA() is checked for time 't'. ** Note that the sign of the result is opposite to common usage, * e.g. for EE(S)T which normally is +2h or +3h from UTC, this * function returns -120 or -180. *

45988. **comment:** XXX: no need for indirect call
label: code-design

45989. XXX: the implementation now assumes "chained" bound functions, * whereas "collapsed" bound functions (where there is ever only * one bound function which directly points to a non-bound, final * function) would require a "collapsing" implementation which * merges argument lists etc here.

45990. The current implementation of localeCompare() is simply a codepoint * by codepoint comparison, implemented with a simple string compare * because UTF-8 should preserve codepoint ordering (assuming valid * shortest UTF-8 encoding). ** The specification requires that the return value must be related * to the sort order: e.g. negative means that 'this' comes before * 'that' in sort order. We assume an ascending sort order.

45991. valstack

45992. not protected, respect reclimit, not constructor

45993. Push the resulting view object and attach the ArrayBuffer.

45994. **comment:** XXX: this should be optimized to be a raw query and avoid valstack * operations if possible.
label: code-design

45995. unbalanced stack

45996. value from hook

45997. * If object has been marked finalizable, move it to the * "to be finalized" work list. It will be collected on * the next mark-and-sweep if it is still unreachable * after running the finalizer.

45998. * Struct defs

45999. nbytes zero size case * <-----> * [... | p | x | x | q] [... | p==q] * => [... | x | x | q] [...]

46000. **comment:** not really necessary
label: code-design

46001. [... target] -> [... target keys]

46002. Leading zero.

46003. ''

46004. Make underlying buffer compact to match DUK_BW_GET_SIZE().

46005. first heap allocated, match bit boundary

46006. **comment:** XXX: more specific error classes?
label: code-design

46007. 25: setUTCMilliseconds

46008. * Intermediate value helpers

46009. Limits

46010. Encoded as surrogate pair, each encoding to 3 bytes for * 6 bytes total. Codepoints above U+10FFFF encode as 6 bytes * too, see duk_unicode_encode_cesu8().

46011. * Setup value stack: clamp to 'nargs', fill up to 'nregs' * * Value stack may either grow or shrink, depending on the * number of func registers and the number of actual arguments. * If nregs >= 0, func wants args clamped to 'nargs'; else it * wants all args (= 'num_stack_args').

46012. [sep ToObject(this) len sep result]

46013. * Variant 2 or 4

46014. Lightfunc, always success.

46015. step 11.c is relevant only if non-strict (checked in 11.c.ii)

46016. XX==

46017. **comment:** A plain buffer coerces to a Duktape.Buffer because it's the * object counterpart of the plain buffer value. But it might * still make more sense to produce an ArrayBuffer here?
label: code-design

46018. Force exponential format. Used for toExponential().

46019. Cannot wrap: each object is at least 8 bytes so count is * at most 1/8 of that.

46020. wipe the capture range made by the atom (if any)

46021. Anything else is not constructable.

46022. attrs in E5 Section 15.3.5.1

46023. * Use the index in the header to find the right starting point

46024. Tolerate act_caller->func == NULL which happens in * some finalization cases; treat like unknown caller.

46025. **comment:** XXX: remove DUK_CALL_FLAG_IGNORE_RECLIMIT flag: there's now the * reclimit bump?
label: code-design

46026. maximum length of standard format tag that we support

46027. buffer size is >= 1

46028. ignore encoding for now

46029. enable manually for dumping

46030. (?=

46031. require to be safe

46032. Parser part count.

46033. for side effects, result ignored

46034. Now we can check offset validity.

46035. Milliseconds between status notify and transport peeks.

46036. Matching separator index is used in the control table

46037. rehash even if old and new sizes are the same to get rid of * DELETED entries.

46038. * Two pass approach to processing the property descriptors. * On first pass validate and normalize all descriptors before * any changes are made to the target object. On second pass * make the actual modifications to the target object. ** Right now we'll just use the same normalize/validate helper * on both passes, ignoring its outputs on the first pass.

46039. * Stringify implementation.

46040. ToNumber() for a double is a no-op.

46041. default: char escape (two chars)

46042. * Hobject allocation. ** Provides primitive allocation functions for all object types (plain object, * compiled function, native function, thread). The object return is not yet * in "heap allocated" list and has a refcount of zero, so caller must careful.

46043. Check actual underlying buffers for validity and that they * cover the copy. No side effects are allowed after the check * so that the validity status doesn't change.

46044. * Special parser for character classes; calls callback for every * range parsed and returns the number of ranges present.

46045. negate result

46046. * Lightfunc

46047. **comment:** * Refzero handling is skipped entirely if (1) mark-and-sweep is * running or (2) execution is paused in the debugger. The objects * are left in the heap, and will be freed by mark-and-sweep or * eventual heap destruction. ** This is necessary during mark-and-sweep because refcounts are also * updated during the sweep phase (otherwise objects referenced by a * swept object would have incorrect refcounts) which then calls here. * This could be avoided by using separate decref macros in * mark-and-sweep; however, mark-and-sweep also calls finalizers which * would use the ordinary decref macros anyway and still call this * function. ** This check must be enabled also when mark-and-sweep support has been * disabled: the flag is also used in heap destruction when running * finalizers for remaining objects, and the flag prevents objects from * being moved around in heap linked lists.

label: code-design

46048. XXX: len >= 0x80000000 won't work below because a signed type * is needed by qsort.

46049. Special handling for year sign.

46050. [arg1 ... argN obj length new_length]

46051. The algorithm in E5.1 Section 15.9.1.12 normalizes month, but * does not normalize the day-of-month (nor check whether or not * it is finite) because it's not necessary for finding the day * number which matches the (year,month) pair. ** We assume that duk_day_from_year() is exact here. ** Without an explicit infinity / NaN check in the beginning, * day_num would be a bogus integer here. ** It's possible for 'year' to be out of integer range here. * If so, we need to return NaN without integer overflow. * This fixes test-bug-setyear-overflow.js.

46052. * Node.js Buffer.prototype.slice([start], [end]) * ArrayBuffer.prototype.slice(begin, [end]) * TypedArray.prototype.slice(begin, [end]) ** The API calls are almost identical; negative indices are counted from end * of buffer, and final indices are clamped (allowing crossed indices). Main * differences: ** - Copy/view behavior; Node.js .slice() and TypedArray .subarray() create * views, ArrayBuffer .slice() creates a copy ** - Resulting object has a different class and prototype depending on the * call (or 'this' argument) ** - TypedArray .subarray() arguments are element indices, not byte offsets

46053. **comment:** When alloc_size == 0 the second allocation may not * actually exist.

label: code-design

46054. statements go here (if any) on next loop

46055. 'yield'

46056. eat trailing comma

46057. pruned

46058. * Declarative environment record. ** Identifiers can never be stored in ancestors and are * always plain values, so we can use an internal helper * and access the value directly with an duk_tval ptr. ** A closed environment is only indicated by it missing * the "book-keeping" properties required for accessing * register-bound variables.

46059. For anything other than an Error instance, we calculate the * error location directly from the current activation if one * exists.

46060. Note: there is no requirement that: 'thr->callstack_preventcount == 1' * like for yield.

46061. lookup results is ignored

46062. **comment:** * ToPrimitive() (E5 Section 9.1) ** ==> implemented in the API.

label: requirement

46063. resume caller must be an ecmascript func

46064. e.g. a = -4, b = 5 --> -4 - 5 + 1 / 5 --> -8 / 5 --> -1 * a = -5, b = 5 --> -5 - 5 + 1 / 5 --> -9 / 5 --> -1 * a = -6, b = 5 --> -6 - 5 + 1 / 5 --> -10 / 5 --> -2

46065. stack contains value (if requested), 'out_desc' is set

46066. no issues with empty memcmp()

46067. [str] -> [substr]

46068. func is NULL for lightfunc

46069. property attributes for identifier (relevant if value != NULL)

46070. Default function to format objects. Tries to use toLogString() but falls * back to toString(). Any errors are propagated out without catching.

46071. don't allow continue

46072. respect reclimit, not constructor

46073. * Notation: double underscore used for internal properties which are not * stored in the property allocation (e.g. '__valstack').

46074. 0x60-0x6f

46075. DUK_BUFOBJ_INT32ARRAY

46076. Caller has already eaten the first char so backtrack one byte.

46077. controls for minimum entry part growth

46078. temp accumulation buffer

46079. Unwind.

46080. * Object built-ins

46081. callstack index

46082. **comment:** XXX: The ES5/5.1/6 specifications require that the key in 'key in obj' * must be string coerced before the internal HasProperty() algorithm is * invoked. A fast path skipping coercion could be safely implemented for * numbers (as number-to-string coercion has no side effects). For ES6 * proxy behavior, the trap 'key' argument must be in a string coerced * form (which is a shame).

label: code-design

46083. take last entry

46084. curr is data, desc is accessor

46085. **comment:** XXX: The incref macro takes a thread pointer but doesn't * use it right now.

label: code-design

46086. An object may be in heap_allocated list with a zero * refcount also if it is a temporary object created by * a finalizer; because finalization now runs inside * mark-and-sweep, such objects will not be queued to * refzero_list and will thus appear here with refcount * zero.

46087. Accept 32-bit decimal integers, no leading zeroes, signs, etc. * Leading zeroes are not accepted (zero index "0" is an exception * handled above).

46088. **comment:** Set: currently a finalizer is disabled by setting it to * undefined; this does not remove the property at the moment. * The value could be type checked to be either a function * or something else; if something else, the property could * be deleted.

label: code-design

46089. * Bitstream decoder

46090. statement id allocation (running counter)

46091. 'public'

46092. single match always

46093. Note: 'curr' refers to 'length' propdesc

46094. Use double parts, they tolerate unnormalized time. ** Note: DUK_DATE_IDX_WEEKDAY is initialized with a bogus value (DUK_PL_TZHOUR) * on purpose. It won't be actually used by duk_bi_date_get_timeval_from_dparts(), * but will make the value initialized just in case, and avoid any * potential for Valgrind issues.

46095. Steps 8-10 have been merged to avoid a "partial" variable.

46096. Used during heap destruction: don't actually run finalizers * because we're heading into forced finalization. Instead, * queue finalizable objects back to the heap_allocated list.

46097. duk_safe_call discipline

46098. **comment:** * Unicode tables containing ranges of Unicode characters in a * packed format. These tables are used to match non-ASCII * characters of complex productions by resorting to a linear * range-by-range comparison. This is very slow, but is expected * to be very rare in practical Ecmascript source code, and thus * compactness is most important. ** The tables are matched using uni_range_match() and the format * is described in src/extract_chars.py.

label: code-design

46099. **comment:** XXX: encoding is ignored now.

label: code-design

46100. use 'undefined' for value automatically

46101. [... obj ...]

46102. **comment:** Function pointers do not always cast correctly to void * * (depends on memory and segmentation model for instance), * so they coerce to NULL.

label: code-design

46103. Buffer length is bounded to 0xffff automatically, avoid compile warning.

46104. Integer division which floors also negative values correctly.

46105. consequence of above

46106. * Self tests to ensure execution environment is sane. Intended to catch * compiler/platform problems which cannot be detected at compile time.

46107. DUK_TAG_NUMBER would logically go here, but it has multiple 'tags'

46108. backtrack (safe)

46109. DUK_FLD_32BIT

46110. * Parse a RegExp token. The grammar is described in E5 Section 15.10. * Terminal constructions (such as quantifiers) are parsed directly here. * * 0xffffffffU is used as a marker for "infinity" in quantifiers. Further, * DUK_MAX_RE_QUANT_DIGITS limits the maximum number of digits that * will be accepted for a quantifier.

46111. **comment:** Note: reuse 'curr'

label: code-design

46112. * Convert and push final string.

46113. evaluate to final form (e.g. coerce GETPROP to code), throw away temp

46114. * Selftest code

46115. use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to the existing one)

46116. duk_tval ptr for 'func' on stack (borrowed reference) or tv_func_copy

46117. * Encoding and decoding basic formats: hex, base64. * * These are in-place operations which may allow an optimized implementation. * * Base-64: <https://tools.ietf.org/html/rfc4648#section-4>

46118. Run mark-and-sweep a few times just in case (unreachable object * finalizers run already here). The last round must rescue objects * from the previous round without running any more finalizers. This * ensures rescued objects get their FINALIZED flag cleared so that * their finalizer is called once more in forced finalization to * satisfy finalizer guarantees. However, we don't want to run any * more finalizer because that'd required one more loop, and so on.

46119. required to keep recursion depth correct

46120. [... buf loop (proplist) (gap) holder ""]

46121. Get default hash part size for a certain entry part size.

46122. Maximum number of digits generated.

46123. [... func buf] -> [... buf]

46124. **comment:** XXX: could share code with duk_js_ops.c, duk_js_compare_helper

label: code-design

46125. Math.pow(+0,y) should be Infinity when y<0. NetBSD pow() * returns -Infinity instead when y is <0 and finite. The * if-clause also catches y == -Infinity (which works even * without the fix).

46126. Free strings in the stringtable and any allocations needed * by the stringtable itself.

46127. e.g. DUK_OP_PREINCV

46128. If neutered must return 0; offset is zeroed during * neutering.

46129. array of declarations: [name1, val1, name2, val2, ...] * valN = (typeN) | (fnum << 8), where fnum is inner func number (0 for vars) * record function and variable declarations in pass 1

46130. breakpoints: [0,breakpoint_count[gc reachable

46131. **comment:** XXX: inefficient loop

label: code-design

46132. * Insert a jump offset at 'offset' to complete an instruction * (the jump offset is always the last component of an instruction). * The 'skip' argument must be computed relative to 'offset', * -without- taking into account the skip field being inserted. * * ... A B C ins X Y Z ... (ins may be a JUMP, SPLIT1/SPLIT2, etc) * => ... A B C ins SKIP X Y Z * * Computing the final (adjusted) skip value, which is relative to the * first byte of the next instruction, is a bit tricky because of the * variable length UTF-8 encoding. See doc/regexp.rst for discussion.

46133. * NYF <int: 5> <int: fatal> <str: msg> <str: filename> <int: linenumber> EOM

46134. Note: not necessary to check p against re_ctx->input_end: * the memory access is checked by duk_inp_get_cp(), while * valid compiled regexps cannot write a saved[] entry * which points to outside the string.

46135. Strings and ROM objects are never placed on the heap allocated list.

46136. [... val root ""] -> [... val val']

46137. **comment:** * JSON built-ins. * * See doc/json.rst. * * Codepoints are handled as duk_uint_fast32_t to ensure that the full * unsigned 32-bit range is supported. This matters to e.g. JX. * * Input parsing doesn't do an explicit end-of-input check at all. This is * safe: input string data is always NUL-terminated (0x00) and valid JSON * inputs never contain plain NUL characters, so that as long as syntax checks * are correct, we'll never read past the NUL. This approach reduces code size * and improves parsing performance, but it's critical that syntax checks are * indeed correct!

label: code-design

46138. stack type fits into 16 bits

46139. 'set'

46140. low to high

46141. required, NULL implies detached

46142. right associative

46143. covers +Infinity

46144. no size check is necessary

46145. r <- (* f 2) * s <- (* (expt b (- e)) 2) == b^(e) * 2 [if b==2 -> b^(1-e)] * m+ <- 1 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round

46146. **comment:** XXX: best behavior for real world compatibility?

label: code-design

46147. 34: setMonth

46148. 20: getSeconds

46149. refcount macros not defined without refcounting, caller must #ifdef now

46150. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object). Return value is never NULL.

46151. DUK_MEMCMP() is guaranteed to return zero (equal) for zero length * inputs so no zero length check is needed.

46152. **comment:** XXX: shared parsing?

label: code-design

46153. Note: reject negative zero.

46154. Delete entry in Duktape.modLoaded[] and rethrow.

46155. already closed

46156. '\xffFormals'

46157. 1

46158. XXX: Multiple tv_func lookups are now avoided by making a local * copy of tv_func. Another approach would be to compute an offset * for tv_func from valstack bottom and recomputing the tv_func * pointer quickly as valstack + offset instead of calling duk_get_tval().

46159. DUK_TOK_NEQ

46160. **comment:** XXX: would be nice to omit this jump when the jump is not * reachable, at least in the obvious cases (such as the case * ending with a 'break'. * * Perhaps duk_parse_stmt() could provide some info on whether * the statement is a "dead end"? * * If implemented, just set pc_prevstmt to -1 when not needed.

label: code-design

46161. no shrink

46162. XXX: duk_to_int() ensures we'll get 8 lowest bits as * as input is within duk_int_t range (capped outside it).
46163. * Ecmascript bytecode
46164. Current require() function
46165. [... errobj]
46166. [... arr]
46167. -> [... escaped_source]
46168. func
46169. ignore fclose() error
46170. DUK_TOK_RETURN
46171. * <https://github.com/svaarala/duktape/issues/127#issuecomment-77863473>
46172. **comment:** Note: not initializing all bytes is normally not an issue: Duktape won't * read or use the uninitialized bytes so valgrind won't issue warnings. * In some special cases a harmless valgrind warning may be issued though. * For example, the DumpHeap debugger command writes out a compiled function's * 'data' area as is, including any uninitialized bytes, which causes a * valgrind warning.
label: code-design
46173. XXX: to util
46174. Byte length would overflow.
46175. register declarations in first pass
46176. implicit this value always undefined for * declarative environment records.
46177. match_caps == 0 without regexps
46178. Value coercion (in stack): ToInteger(), E5 Section 9.4 * API return value coercion: custom
46179. thr->heap->lj.jmpbuf_ptr is checked by duk_err_longjmp() so we don't * need to check that here. If the value is NULL, a panic occurs because * we can't return.
46180. * Object finalizer
46181. **comment:** never shrinks; auto-adds DUK_VALSTACK_INTERNAL_EXTRA, which is generous
label: code-design
46182. -> [... proplist enum_obj key]
46183. elems in source and dest
46184. E5 Section 15.4.5.1, step 4
46185. **comment:** * Handle special cases (NaN, infinity, zero).
label: code-design
46186. DUK_USE_ROM_OBJECTS
46187. stats for current expression being parsed
46188. **comment:** maintain highest 'used' temporary, needed to figure out nregs of function
label: code-design
46189. DUK_IVAL_XXX
46190. **comment:** Serialize uncovered backing buffer as a null; doesn't * really matter as long we're memory safe.
label: code-design
46191. * Error, fatal, and panic handling.
46192. A number can be loaded either through a constant, using * LDINT, or using LDINT+LDINTX. LDINT is always a size win, * LDINT+LDINTX is not if the constant is used multiple times. * Currently always prefer LDINT+LDINTX over a double constant.
46193. guaranteed by string limits
46194. Slot A
46195. terminates expression; e.g. post-increment/-decrement
46196. ignore arguments, return undefined (E5 Section 15.3.4)
46197. Thread state.
46198. g...v
46199. [...] or [... | errobj (M * undefined)] where M = num_stack_rets - 1
46200. omit
46201. always >= 0
46202. Fixed buffer, no zeroing because we'll fill all the data.
46203. Ecma-to-ecma call possible, may or may not be a tail call. * Avoid C recursion by being clever.
46204. terminal type: no depth check
46205. track cpos while scanning
46206. DUK_TOK_SEQ
46207. curr_pc synced by duk_handle_ecma_call_setup()
46208. **comment:** XXX: could accept numbers larger than 32 bits, e.g. up to 53 bits?
label: code-design
46209. accessor flag not encoded explicitly
46210. For now only needed by the debugger.
46211. * Bytecode instruction representation during compilation * * Contains the actual instruction and (optionally) debug info.
46212. remove value
46213. * Forward declarations
46214. mm <- mp
46215. **comment:** LDINTX is not necessarily in FASTINT range, so * no fast path for now. * * XXX: perhaps restrict LDINTX to fastint range, wider * range very rarely needed.
label: code-design
46216. the entries [new_e_next, new_e_size_adjusted[are left uninitialized on purpose (ok, not gc reachable)
46217. rethrow
46218. * Accessor macros for function specific data areas
46219. Argument validation and func/args offset.
46220. day number for Jan 1 since epoch
46221. DUK_HOBJECT_FLAG_NATIVEFUNCTION varies
46222. compact
46223. **comment:** XXX: could clear FINALIZED already here; now cleared in * next mark-and-sweep.
label: code-design
46224. **comment:** XXX: this generates quite large code - perhaps select the error * class based on the code and then just use the error 'name'?
label: code-design
46225. require a short (8-bit) reg/const which fits into bytecode B/C slot
46226. **comment:** Function.prototype.bind() should never let this happen, * ugly error message is enough.
label: code-design
46227. "123." is allowed in some formats
46228. Apply ToNumber() to specified index; if ToInteger(val) in [0,99], add * 1900 and replace value at idx_val.
46229. NB: must accept reserved words as property name
46230. no array part
46231. * Misc
46232. error message doesn't matter, ignored anyway
46233. Resolve 'res' directly into the LHS binding, and use * that as the expression value if safe. If not safe, * resolve to a temp/const and copy to LHS.
46234. * Found existing own (non-inherited) plain property. * Do an access control check and update in place.
46235. Currently nothing to free

46236. **comment:** Convenience copies from heap/vm for faster access.
label: code-design

46237. Reply with tvals pushed by request callback

46238. If out of catchstack, cat = thr->catchstack - 1; * new_cat_top will be 0 in that case.

46239. **comment:** XXX: Could just lookup . toJSON() and continue in fast path, * as it would almost never be defined.
label: code-design

46240. **comment:** * Raw write/read macros for big endian, unaligned basic values. * Caller ensures there's enough space. The macros update the pointer * argument automatically on resizes. The idiom seems a bit odd, but * leads to compact code.
label: code-design

46241. **comment:** * "arguments" and "caller" must be mapped to throwers for strict * mode and bound functions (E5 Section 15.3.5). * * XXX: This is expensive to have for every strict function instance. * Try to implement as virtual properties or on-demand created properties.
label: code-design

46242. 'index'
46243. index is above internal string length -> property is fully normal

46244. LF line terminator; CR LF and Unicode lineterms are handled in slow path

46245. * Misc helpers

46246. expt== -1023 -> bitstart=0 (leading 1); * expt== -1024 -> bitstart=-1 (one left of leading 1), etc

46247. bound chain resolved

46248. base

46249. Line format: <time> <entryLev> <loggerName>: <msg>

46250. NEXTENUM jump slot: executed when enum finished

46251. **comment:** prototype should be last, for readability
label: code-design

46252. **comment:** XXX: direct implementation
label: requirement

46253. Error codes.

46254. multi-character sets not allowed as part of ranges, see * E5 Section 15.10.2.15, abstract operation CharacterRange.

46255. * Property lookup

46256. Constants for duk_hashstring().

46257. roughly 1 kB

46258. advance by one character (code point) and one char_offset

46259. no need to coerce

46260. remove key and value

46261. A regexp token value.

46262. large context; around 2kB now

46263. lexer character window helpers

46264. at index 1

46265. trivial cases

46266. duplicate/invalid key checks; returns 1 if syntax error

46267. Here we could remove references to built-ins, but it may not be * worth the effort because built-ins are quite likely to be shared * with another (unterminated) thread, and terminated threads are also * usually garbage collected quite quickly. Also, doing DECREFs * could trigger finalization, which would run on the current thread * and have access to only some of the built-ins. Garbage collection * deals with this correctly already.

46268. Name will be filled from function expression, not by caller. * This case is used by Function constructor and duk_compile() * API with the DUK_COMPILE_FUNCTION option.

46269. original target at entry_top - 1

46270. not emitted

46271. **comment:** XXX: duk_ssize_t would be useful here
label: code-design

46272. comp_ctx->lex.input and comp_ctx->lex.input_length filled by caller

46273. **comment:** XXX: pre-checks (such as no duplicate keys)
label: code-design

46274. To handle deeper indents efficiently, make use of copies we've * already emitted. In effect we can emit a sequence of 1, 2, 4, * 8, etc copies, and then finish the last run. Byte counters * avoid multiply with gap_len on every loop.

46275. **comment:** XXX: TRYCATCH handling should be reworked to avoid creating * an explicit scope unless it is actually needed (e.g. function * instances or eval is executed inside the catch block). This * rework is not trivial because the compiler doesn't have an * intermediate representation. When the rework is done, the * opcode format can also be made more straightforward.
label: code-design

46276. Because new_size != 0, if condition doesn't need to be * (p != NULL || new_size == 0).

46277. * Init lexer

46278. %'

46279. * This test fails on an exotic ARM target; double-to-uint * cast is incorrectly clamped to -signed- int highest value. * *
<https://github.com/svaara/duktape/issues/336>

46280. DUK_TOK_NUMBER

46281. IEEE exp without bias

46282. 31 bits

46283. call handling

46284. not exposed

46285. No need to check for size of bp_active list, * it's always larger than maximum number of * breakpoints.

46286. **comment:** * Restart execution by reloading thread state. * * Note that 'thr' and any thread configuration may have changed, * so all local variables are suspect and we need to reinitialize. * * The number of local variables should be kept to a minimum: if * the variables are spilled, they will need to be loaded from * memory anyway. * * Any 'goto restart_execution;' code path in opcode dispatch must * ensure 'curr_pc' is synced back to act->curr_pc before the goto * takes place. * * The interpreter must be very careful with memory pointers, as * many pointers are not guaranteed to be 'stable' and may be * reallocated and relocated on-the-fly quite easily (e.g. by a * memory allocation or a property access). * * The following are assumed to have stable pointers: * - the current thread * - the current function * - the bytecode, constant table, inner function table of the * current function (as they are a part of the function allocation) * * The following are assumed to have semi-stable pointers: * - the current activation entry: stable as long as callstack * is not changed (reallocated by growing or shrinking), or * by any garbage collection invocation (through finalizers) * - Note in particular that ANY DECREF can invalidate the * activation pointer, so for the most part a fresh lookup * is required * * The following are not assumed to have stable pointers at all: * - the value stack (registers) of the current thread * - the catch stack of the current thread * * See execution.rst for discussion.
label: code-design

46287. For internal use: get array part value

46288. 10xx xxxx -> invalid

46289. Compiler is responsible for selecting property flags (configurability, * writability, etc).

46290. _Varmap is dense

46291. DUK_TOK_BAND

46292. **comment:** 0x00 ... 0x7f: as is * 0x80: escape generically * 0x81: slow path * 0xa0 ... 0xff: backslash + one char
label: code-design

46293. * Exposed matcher function which provides the semantics of RegExp.prototype.exec(). * * RegExp.prototype.test() has the same semantics as exec() but does not return the * result object (which contains the matching string and capture groups). Currently * there is no separate test() helper, so a temporary result object is

created and * discarded if test() is needed. This is intentional, to save code space. ** Input stack: [... re_obj input] * Output stack: [... result]
46294. * Buffers have no internal references. However, a dynamic * buffer has a separate allocation for the buffer. This is * freed by duk_heap_free_heapheader_raw().
46295. indirect eval
46296. Numbers half-way between integers must be rounded towards +Infinity, * e.g. -3.5 must be rounded to -3 (not -4). When rounded to zero, zero * sign must be set appropriately. E5.1 Section 15.8.2.15. ** Note that ANSI C round() is "round to nearest integer, away from zero", * which is incorrect for negative values. Here we make do with floor().
46297. * Some assertions (E5 Section 13.2).
46298. Shared lenient buffer length clamping helper. No negative indices, no * element/byte shifting.
46299. **comment:** XXX: some code might benefit from DUK__SETTEMP_IFTEMP(ctx,x)
 label: code-design
46300. preincrement and predecrement
46301. Value would normally be omitted, replace with 'null'.
46302. e.g. require('foo'), empty terms not allowed
46303. string is internal
46304. * Parse regexp Disjunction. Most of regexp compilation happens here. ** Handles Disjunction, Alternative, and Term productions directly without * recursion. The only constructs requiring recursion are positive/negative * lookaheads, capturing parentheses, and non-capturing parentheses. ** The function determines whether the entire disjunction is a 'simple atom' * (see doc/regexp.rst discussion on 'simple quantifiers') and if so, * returns the atom character length which is needed by the caller to keep * track of its own atom character length. A disjunction with more than one * alternative is never considered a simple atom (although in some cases * that might be the case). ** Return value: simple atom character length or < 0 if not a simple atom. * Appends the bytecode for the disjunction matcher to the end of the temp * buffer. ** Regexp top level structure is: ** Disjunction = Term* | Term* | Disjunction ** Term = Assertion * | Atom * | Atom Quantifier ** An empty Term sequence is a valid disjunction alternative (e.g. /|||c||/). ** Notes: ** Tracking of the 'simple-ness' of the current atom vs. the entire * disjunction are separate matters. For instance, the disjunction * may be complex, but individual atoms may be simple. Furthermore, * simple quantifiers are used whenever possible, even if the * disjunction as a whole is complex. ** The estimate of whether an atom is simple is conservative now, * and it would be possible to expand it. For instance, captures * cause the disjunction to be marked complex, even though captures * -can- be handled by simple quantifiers with some minor modifications. ** Disjunction 'tainting' as 'complex' is handled at the end of the * main for loop collectively for atoms. Assertions, quantifiers, * and '| tokens need to taint the result manually if necessary. * Assertions cannot add to result char length, only atoms (and * quantifiers) can; currently quantifiers will taint the result * as complex though.
46305. Get a pointer to the current buffer contents (matching current allocation * size). May be NULL for zero size dynamic/external buffer.
46306. success: leave varname in stack
46307. safe
46308. [... closure template len_value]
46309. Note: negative pc values are ignored when patching jumps, so no explicit checks needed
46310. [... func this (crud) retval]
46311. DUK_USE_DATE_PRS_STRTIME
46312. **comment:** Global case is more complex.
 label: code-design
46313. max, excl. varargs marker
46314. should be a safe way to compute this
46315. not an error
46316. prev_token slot2
46317. side effects, perform in-place
46318. **comment:** XXX: turkish / azeri
 label: code-design
46319. current variable environment (may be NULL if delayed)
46320. May happen in some out-of-memory corner cases.
46321. * Store lexer position for a later rewind
46322. -> [... this timeval]
46323. Avoid using RegExp.prototype methods, as they're writable and * configurable and may have been changed.
46324. Index validation is strict, which differs from duk_equals(). * The strict behavior mimics how instanceof itself works, e.g. * it is a TypeError if rval is not a - callable- object. It would * be somewhat inconsistent if rval would be allowed to be * non-existent without a TypeError.
46325. prev_token slot1
46326. Note1
46327. **comment:** When DUK_USE_HEAPPTR16 (and DUK_USE_REFCOUNT16) is in use, the * struct won't align nicely to 4 bytes. This 16-bit extra field * is added to make the alignment clean; the field can be used by * heap objects when 16-bit packing is used. This field is now * conditional to DUK_USE_HEAPPTR16 only, but it is intended to be * used with DUK_USE_REFCOUNT16 and DUK_USE_DOUBLE_LINKED_HEAP; * this only matter to low memory environments anyway.
 label: code-design
46328. DUK_TOK_IMPORT
46329. * Native call.
46330. input offset tracking
46331. -> [... closure template newobj closure]
46332. fixed arg count: value
46333. chain jumps for case * evaluation and checking
46334. Called for handling both a longjmp() with type DUK_LJ_TYPE_YIELD and * when a RETURN opcode terminates a thread and yields to the resumer.
46335. function call
46336. full 3-byte -> 4-char conversions
46337. regexp compilation limits
46338. **comment:** Current PC, accessed by other functions through thr->ptr_to_curr_pc. * Critical for performance. It would be safest to make this volatile, * but that eliminates performance benefits; aliasing guarantees * should be enough though.
 label: code-design
46339. 0x00: slow path * other: as is
46340. **comment:** XXX: this could be optimized; there is only one call site now though
 label: code-design
46341. reject 'in' token (used for for-in)
46342. Data area, fixed allocation, stable data ptrs.
46343. list of conversion specifiers that terminate a format tag; * this is unfortunately guesswork.
46344. Preliminaries for __proto__ and setPrototypeOf (E6 19.1.2.18 steps 1-4); * magic: 0=setter call, 1=Object.setPrototypeOfOf
46345. DUK_LEXER_H_INCLUDED
46346. type bit mask: types which certainly produce 'undefined'
46347. XXX: for threads, compact value stack, call stack, catch stack?
46348. **comment:** Lightfuncs are currently supported by coercing to a temporary * Function object; changes will be allowed (the coerced value is * extensible) but will be lost.
 label: code-design
46349. second, parse flags
46350. * Not found: write to global object (non-strict) or ReferenceError * (strict); see E5 Section 8.7.2, step 3.
46351. strict mode restrictions (E5 Section 12.2.1)
46352. **comment:** Forget the previous allocation, setting size to 0 and alloc to * NULL. Caller is responsible for freeing the previous allocation. * Getting the allocation and clearing it is done in the same API * call to avoid any chance of a realloc.

label: code-design

46353. [regexp]

46354. '\xffFinalizer'

46355. **comment:** The entries can either be register numbers or 'null' values. * Thus, no need to DECREF them and get side effects. DECREF'ing * the keys (strings) can cause memory to be freed but no side * effects as strings don't have finalizers. This is why we can * rely on the object properties not changing from underneath us.

label: code-design

46356. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

46357. For all combinations: +0 < +0, +0 < -0, -0 < +0, -0 < -0, * steps e, f, and g.

46358. output is never longer than input during resolution

46359. **comment:** XXX: make fast paths optional for size minimization?

label: code-design

46360. **comment:** XXX: handling for array part missing now; this doesn't affect * compliance but causes array entry writes using defineProperty() * to always abandon array part.

label: code-design

46361. Fast path the binary case

46362. * Unicode helpers

46363. **comment:** XXX: zero assumption

label: code-design

46364. * Assertions after

46365. r <- r * s <- (* s B) * m+ <- m+ * m- <- m- * k <- (+ k 1)

46366. 'Math'

46367. 1e8: protects against deeply nested inner functions

46368. insert ranges instruction, range count patched in later

46369. * Validate and convert argument property descriptor (an Ecmascript * object) into a set of defprop_flags and possibly property value, * getter, and/or setter values on the value stack. * Lightfunc set/get values are coerced to full Functions.

46370. tc1 = false, tc2 = true

46371. after this, return paths should 'goto finished' for decrement

46372. obj_index

46373. zero-width non-joiner

46374. activation has active breakpoint(s)

46375. Constants for built-in string data depacking.

46376. Need to set curr_token.t because lexing regexp mode depends on current * token type. Zero value causes "allow regexp" mode.

46377. start (incl) and end (excl) of trimmed part

46378. Careful with wrapping: arr_idx upshift may easily wrap, whereas * length downshift won't.

46379. interpret e.g. '09' as '0', not NaN

46380. * splice()

46381. no argument given -> leave components untouched

46382. **comment:** This fast path is pretty marginal in practice. * XXX: candidate for removal.

label: code-design

46383. character represents itself

46384. eat 'return'

46385. **comment:** If thr != NULL, the thr may still be in the middle of * initialization. * XXX: Improve the thread viability test.

label: code-design

46386. [value offset end]

46387. Relies on NULL encoding to zero.

46388. **comment:** XXX: the current implementation works but is quite clunky; it compiles * to almost 1,4kB of x86 code so it needs to be simplified (better approach, * shared helpers, etc). Some ideas for refactoring: * * - a primitive to convert a string into a regexp matcher (reduces matching * code at the cost of making matching much slower) * - use replace() as a basic helper for match() and split(), which are both * much simpler * - API call to get_prop and to_boolean

label: code-design

46389. * Compilation and evaluation

46390. Note: Identifier rejects reserved words

46391. -> [obj trap_result]

46392. * Unicode letter is now taken to be the categories: * * Lu, Ll, Lt, Lm, Lo * * (Not sure if this is exactly correct.) * * The ASCII fast path consists of: * * 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z']

46393. caller ensures

46394. * Not found (in registers or record objects). Declare * to current variable environment.

46395. **comment:** 'this' binding is not needed here

label: requirement

46396. no need to check callable; duk_call() will do that

46397. mark finalizer work list as reachability roots

46398. Initial '{' has been checked and eaten by caller.

46399. '\xffMap'

46400. The directive prologue flag is cleared by default so that it is * unset for any recursive statement parsing. It is only "revived" * if a directive is detected. (We could also make directives only * allowed if 'allow_source_elem' was true.)

46401. 'NaN'

46402. unsigned 31-bit range

46403. DUK_USE_EXPLICIT_NULL_INIT

46404. valgrind whine without this

46405. **comment:** * Helper for calling a user error handler. * * 'thr' must be the currently active thread; the error handler is called * in its context. The valstack of 'thr' must have the error value on * top, and will be replaced by another error value based on the return * value of the error handler. * * The helper calls duk_handle_call() recursively in protected mode. * Before that call happens, no longjmps should happen; as a consequence, * we must assume that the valstack contains enough temporary space for * arguments and such. * * While the error handler runs, any errors thrown will not trigger a * recursive error handler call (this is implemented using a heap level * flag which will "follow" through any coroutines resumed inside the * error handler). If the error handler is not callable or throws an * error, the resulting error replaces the original error (for Duktape * internal errors, duk_error_throw.c further substitutes this error with * a DoubleError which is not ideal). This would be easy to change and * even signal to the caller. * * The user error handler is stored in 'Duktape.errCreate' or * 'Duktape.errThrow' depending on whether we're augmenting the error at * creation or throw time. There are several alternatives to this approach, * see doc/error-objects.rst for discussion. * * Note: since further longjmp()'s may occur while calling the error handler * (for many reasons, e.g. a labeled 'break' inside the handler), the * caller can make no assumptions on the thr->heap->lj state after the * call (this affects especially duk_error_throw.c). This is not an issue * as long as the caller writes to the lj state only after the error handler * finishes.

label: code-design

46406. * Exposed calls

46407. key encountered as a setter

46408. Don't need to sync curr_pc here; duk_new() will do that * when it augments the created error.

46409. one i_step over

46410. Shared prefix for all buffer types.

46411. * Environment record creation and 'arguments' object creation. * Named function expression name binding is handled by the * compiler; the compiled function's parent env will contain * the (immutable) binding already. * * Delayed creation (on demand) is handled in duk_js_var.c

46412. Reject a proxy object as the target because it would need * special handler in property lookups. (ES6 has no such restriction)

46413. must restore reliably before returning

46414. * Basic context initialization. * * Some init values are read from the bytecode header * whose format is (UTF-8 codepoints): * * uint flags * uint nsaved (even, 2n+2 where n = num captures)

46415. return 'res_arr' or 'null'

46416. **comment:** This would be nice, but parsing is faster without resetting the * value slots. The only side effect is that references to temporary * string values may linger until lexing is finished; they're then * freed normally.
label: code-design

46417. break jump

46418. "123"

46419. Note: the only yield-preventing call is Duktape.Thread.yield(), hence check for 1, not 0

46420. range check not necessary because all 4-bit values are mapped

46421. floor(1.15 * (1 << 10))

46422. expect_eof

46423. -- p_insert -- p_curr * v v * | ... | insert | ... | curr

46424. * Stack constants

46425. call: this(fmt(arg)

46426. Combined step 11 (empty string special case) and 14-15.

46427. w/o refcounting

46428. Example: d=3.5, t=0.5 -> ret = (3 + 1) & 0xfe = 4 & 0xfe = 4 * Example: d=4.5, t=0.5 -> ret = (4 + 1) & 0xfe = 5 & 0xfe = 4

46429. Write fully.

46430. Assume IEEE round-to-even, so that shorter encoding can be used * when round-to-even would produce correct result. By removing * this check (and having low_ok == high_ok == 0) the results would * still be accurate but in some cases longer than necessary.

46431. object established using Function.prototype.bind()

46432. * Heap object refcount finalization. * * When an object is about to be freed, all other objects it refers to must * be decref'd. Refcount finalization does NOT free the object or its inner * allocations (mark-and-sweep shares these helpers), it just manipulates * the refcounts. * * Note that any of the decref's may cause a refcount to drop to zero, BUT * it will not be processed inline; instead, because refzero is already * running, the objects will just be queued to refzero list and processed * later. This eliminates C recursion.

46433. output radix

46434. resume check from proxy target

46435. native function properties

46436. 1e8

46437. Lightfunc flags packing and unpacking.

46438. Helpers exposed for internal use

46439. * Hash function duk_util_hashbytes(). * * Currently, 32-bit MurmurHash2. * * Don't rely on specific hash values; hash function may be endianness * dependent, for instance.

46440. **comment:** NULL always indicates alloc failure because * new_alloc_size > 0.
label: code-design

46441. The final object may be a normal function or a lightfunc. * We need to re-lookup tv_func because it may have changed * (also value stack may have been resized). Loop again to * do that; we're guaranteed not to come here again.

46442. **comment:** For objects JSON.stringify() only looks for own, enumerable * properties which is nice for the fast path here. * * For arrays JSON.stringify() uses [[Get]] so it will actually * inherit properties during serialization! This fast path * supports gappy arrays as long as there's no actual inherited * property (which might be a getter etc). * * Since recursion only happens for objects, we can have both * recursion and loop checks here. We use a simple, depth-limited * loop check in the fast path because the object-based tracking * is very slow (when tested, it accounted for 50% of fast path * execution time for input data with a lot of small objects!).
label: code-design

46443. assertion omitted

46444. -> [... obj finalizer]

46445. does not fit into an uchar

46446. need to normalize, may even cancel to 0

46447. Callee/caller are throwers and are not deletable etc. They * could be implemented as virtual properties, but currently * there is no support for virtual properties which are accessors * (only plain virtual properties). This would not be difficult * to change in duk_hobject_props, but we can make the throwers * normal, concrete properties just as easily. * * Note that the specification requires that the *same* thrower * built-in object is used here! See E5 Section 10.6 main * algorithm, step 14, and Section 13.2.3 which describes the * thrower. See test case test-arguments-throwers.js.

46448. used for array, stack, and entry indices

46449. [...] name reg_bind]

46450. Select copy mode. Must take into account element * compatibility and validity of the underlying source * buffer.

46451. invalid value which never matches

46452. Arrays never have other exotic behaviors.

46453. 0xxx xxxx [7 bits]

46454. [...] env target]

46455. Setup builtins from ROM objects. All heaps/threads will share * the same readonly objects.

46456. **comment:** alloc size in elements
label: code-design

46457. * Init remaining result fields * * 'nregs' controls how large a register frame is allocated. * * 'nargs' controls how many formal arguments are written to registers: * r0, ... r(nargs-1). The remaining registers are initialized to * undefined.

46458. note: caller 'sp' is intentionally not updated here

46459. Underlying buffer (refcounted), may be NULL.

46460. actual chars dropped (not just NUL term)

46461. free-form

46462. formals for 'func' (may be NULL if func is a C function)

46463. **comment:** bytes of indent available for copying
label: code-design

46464. * Valstack manipulation for results.

46465. [obj key desc value get set curr_value varname]

46466. * Add _Tracedata to an error on the stack top.

46467. force executor restart to recheck breakpoints; used to handle function returns (see GH-303)

46468. Lexer context. Same context is used for Ecmascript and Regexp parsing.

46469. **comment:** XXX: spilling
label: code-design

46470. Note: rely on index ordering

46471. * Get enumerated keys in an Ecmascript array. Matches Object.keys() behavior * described in E5 Section 15.2.3.14.

46472. NB: duk_lexer_getpoint() is a macro only

46473. dangerous: must only lower (temp_max not updated)

46474. x <- x + y, use t as temp

46475. * Init argument related properties

46476. add BC*2^16
46477. initial stringtable size, must be prime and higher than DUK_UTIL_MIN_HASH_PRIME
46478. char format: use int
46479. Clamping only necessary for 32-bit ints.
46480. maximum size check is handled by callee
46481. -> [... val root val]
46482. Note: start_offset/end_offset can still be < 0 here.
46483. cat_idx catcher is kept, even for finally
46484. "null", "true", and "false" are always reserved words. * Note that "get" and "set" are not!
46485. AssignmentExpression
46486. DUK_TOK_MOD
46487. **comment:** XXX: this is pointless here because pass 1 is throw-away
label: code-design
46488. Possibly truncated; there is no explicit truncation * marker so this is the best we can do.
46489. 'fatal'
46490. * Plain, boring reachable object.
46491. value of re_ctx->captures at start of atom
46492. * Find an existing key from entry part either by linear scan or by * using the hash index (if it exists). * * Sets entry index (and possibly the hash index) to output variables, * which allows the caller to update the entry and hash entries in-place. * If entry is not found, both values are set to -1. If entry is found * but there is no hash part, h_idx is set to -1.
46493. pc_label
46494. 'configurable'
46495. * Object prototype
46496. 'this' binding exists
46497. XXX: direct manipulation, or duk_replace_tval()
46498. fill window
46499. **comment:** * Helper for handling a "bound function" chain when a call is being made. * * Follows the bound function chain until a non-bound function is found. * Prepends the bound arguments to the value stack (at idx_func + 2), * updating 'num_stack_args' in the process. The 'this' binding is also * updated if necessary (at idx_func + 1). Note that for constructor calls * the 'this' binding is never updated by [[BoundThis]]. * * XXX: bound function chains could be collapsed at bound function creation * time so that each bound function would point directly to a non-bound * function. This would make call time handling much easier.
label: code-design
46500. * Constructor arguments are currently somewhat compatible with * (keep it that way if possible): * * http://nodejs.org/api/buffer.html * * Note that the ToBuffer() coercion (duk_to_buffer()) does NOT match * the constructor behavior.
46501. * Emit compiled regexp header: flags, ncaptures * (insertion order inverted on purpose)
46502. * Logging support
46503. [key result]
46504. unchanged by Duktape.Thread.resume()
46505. arguments object would be accessible; note that shadowing * bindings are arguments or function declarations, neither * of which are deletable, so this is safe.
46506. Return 1: got EOM
46507. include removed: duk_internal.h
46508. NOTE: length may be zero
46509. SCANBUILD: NULL pointer dereference, doesn't actually trigger, * asserted above.
46510. Alignment guarantee
46511. !DUK_USE_PREFER_SIZE
46512. Note1: the specification doesn't require matching a time form with * just hours ("HH"), but we accept it here, e.g. "2012-01-02T12Z". * * Note2: the specification doesn't require matching a timezone offset * with just hours ("HH"), but accept it here, e.g. "2012-01-02T03:04:05+02"
46513. * Mark the heap.
46514. silence scan-build warning
46515. varname is popped by above code
46516. Estimating the result size beforehand would be costly, so * start with a reasonable size and extend as needed.
46517. Decode failed.
46518. * Init stringcache
46519. [... constructor arg1 ... argN final_cons]
46520. **comment:** XXX: lithuanian, explicit dot rules
label: code-design
46521. XXX: similar coercion issue as in DUK_TOK_PERIOD
46522. **comment:** Wipe capture range and save old values for backtracking. * * XXX: this typically happens with a relatively small idx_count. * It might be useful to handle cases where the count is small * (say <= 8) by saving the values in stack instead. This would * reduce memory churn and improve performance, at the cost of a * slightly higher code footprint.
label: code-design
46523. source out-of-bounds (but positive)
46524. 'with' binding has no catch clause, so can't be here unless a normal try-catch
46525. second incref for the entry reference
46526. **comment:** XXX: declvar takes an duk_tval pointer, which is awkward and * should be reworked.
label: code-design
46527. Get.
46528. DUK_TOK_PRIVATE
46529. randomized pivot selection
46530. * Check whether the property already exists in the prototype chain. * Note that the actual write goes into the original base object * (except if an accessor property captures the write).
46531. These are not needed when only Duktape.Buffer is supported.
46532. duk_hobject specific fields.
46533. [val] -> []
46534. Map DUK_HBUFFEROBJECT_ELEM_xxx to duk_hobject class number. * Sync with duk_hbufferobject.h and duk_hobject.h.
46535. * Shared exit path
46536. force_exponential
46537. As an initial implementation, write flush after every EOM (and the * version identifier). A better implementation would flush only when * Duktape is finished processing messages so that a flush only happens * after all outbound messages are finished on that occasion.
46538. DUK_INVALID_INDEX won't be accepted as a valid index.
46539. * Heap thread object representation. * * duk_hthread is also the 'context' (duk_context) for exposed APIs * which mostly operate on the topmost frame of the value stack.
46540. DUK_USE_BUFFEROBJECT_SUPPORT
46541. As an initial implementation, read flush after exiting the message * loop. If transport is broken, this is a no-op (with debug logs).
46542. * Figure out how generated digits match up with the mantissa, * and then perform rounding. If mantissa overflows, need to * recompute the exponent (it is bumped and may overflow to * infinity). * * For normal numbers the leading '1' is hidden and ignored, * and the last bit is used for rounding. * * rounding pt * <-----52----->| * 1 x x x x ... x x x |y ==> x x x x ... x x x x * * For denormals, the leading '1' is included in the number, * and the rounding point is different: * * rounding pt * <-52 or less-->| * 1 x x x x ... x x |x y ==> 0 0 ... 1 x x ... x x * * The largest denormals will have a mantissa beginning with * a '1' (the explicit leading bit); smaller denormals will * have leading zero bits. * * If the exponent would become too high, the result becomes * Infinity. If the exponent is so small that the entire

* mantissa becomes zero, the result becomes zero. ** Note: the Dragon4 'k' is off-by-one with respect to the IEEE * exponent. For instance, k==0 indicates that the leading '1' * digit is at the first binary fraction position (0.1xxx...); * the corresponding IEEE exponent would be -1.

46543. string hash

46544. Relookup in case coerce_func() has side effects, e.g. ends up coercing an object

46545. 0 = no update

46546. regexp opcodes

46547. 'function'

46548. recursion limit

46549. utf-8 validation ensures these

46550. count, not including sep

46551. string data is external (duk_hstring_external)

46552. currently paused: talk with debug client until step/resume

46553. duk_concat() coerces arguments with ToString() in correct order

46554. 2 to 11

46555. guaranteed to finish

46556. const flag for B

46557. one i_step added at top of loop

46558. 'res' may be NULL if new allocation size is 0.

46559. **comment:** Shared entry code for many Array built-ins. Note that length is left * on stack (it could be popped, but that's not necessary).

label: code-design

46560. thread

46561. assume PC is at most 32 bits and non-negative

46562. 1 0 <2 bits>

46563. **comment:** * Shared handling for logical AND and logical OR. * * args = (truthval << 8) + rbp * * Truthval determines when to skip right-hand-side. * For logical AND truthval=1, for logical OR truthval=0. ** See doc/compiler.rst for discussion on compiling logical * AND and OR expressions. The approach here is very simplistic, * generating extra jumps and multiple evaluations of truth values, * but generates code on-the-fly with only local back-patching. ** Both logical AND and OR are syntactically left-associated. * However, logical ANDs are compiled as right associative * expressions, i.e. "A && B && C" as "A && (B && C)", to allow * skip jumps to skip over the entire tail. Similarly for logical OR.

label: code-design

46564. NULL obj->p is OK

46565. * In strict mode E5 protects 'eval' and 'arguments' from being * assigned to (or even declared anywhere). Attempt to do so * should result in a compile time SyntaxError. See the internal * design documentation for details. ** Thus, we should never come here, run-time, for strict code, * and name 'eval' or 'arguments'.

46566. Assume that thr->valstack_bottom has been set-up before getting here.

46567. len: 9

46568. * E5 Section 11.4.9

46569. deal with weak references first

46570. duk_push_sprintf constants

46571. negative: dst info not available

46572. topmost element is the result array already

46573. Fast path, handle units with just actual encoding characters.

46574. out of spec, must be configurable

46575. * Debug logging after adjustment.

46576. * Pushers

46577. **comment:** * Note: each API operation potentially resizes the callstack, * so be careful to re-lookup after every operation. Currently * these is no issue because we don't store a temporary 'act' * pointer at all. (This would be a non-issue if we operated * directly on the array part.)

label: code-design

46578. implicit_return_value

46579. XXX: direct valstack write

46580. [... source? filename]

46581. No resize has occurred so temp_desc->e_idx is still OK

46582. lookup for 0x000a above assumes shortest encoding now

46583. XXX: E5.1 Section 11.1.4 coerces the final length through * ToUint32() which is odd but happens now as a side effect of * 'arr_idx' type.

46584. Read tvals from the message and push them onto the valstack, * then call the request callback to process the request.

46585. true, despite side effect resizes

46586. the stack is unbalanced here on purpose; we only rely on the * initial two values: [name this].

46587. [start end str]

46588. source is eval code (not global)

46589. switch initial byte

46590. Note: pc is unsigned and cannot be negative

46591. * Casting

46592. **comment:** * User declarations, e.g. prototypes for user functions used by Duktape * macros. Concretely, if DUK_USE_PANIC_HANDLER is used and the macro * value calls a user function, it needs to be declared for Duktape * compilation to avoid warnings.

label: code-design

46593. Longjmp handling has restored jmpbuf_ptr.

46594. * Round a number upwards to a prime (not usually the nearest one). ** Uses a table of successive 32-bit primes whose ratio is roughly * constant. This keeps the relative upwards 'rounding error' bounded * and the data size small. A simple 'predict-correct' compression is * used to compress primes to one byte per prime. See genhashsizes.py * for details. ** The minimum prime returned here must be coordinated with the possible * probe sequence steps in duk_hobject and duk_heap stringtable.

46595. -> [... varname val this]

46596. processed one or more messages

46597. [... v1(filename) v2(line+flags)]

46598. must have been, since in array part

46599. slower but more compact variant

46600. TypedArray (or other non-ArrayBuffer duk_hbufferobject). * Conceptually same behavior as for an Array-like argument, * with a few fast paths.

46601. **comment:** parameter passing, not thread safe

label: requirement

46602. **comment:** Two temporaries are preallocated here for variants 3 and 4 which need * registers which are never clobbered by expressions in the loop * (concretely: for the enumerator object and the next enumerated value). * Variants 1 and 2 "release" these temps.

label: code-design

46603. Note: decimal number may start with a period, but must be followed by a digit

46604. main reachability roots

46605. DUK_HBUFFER_H_INCLUDED

46606. Inputs: explicit arguments (nargs), +1 for key, +2 for obj_index/nargs passing. * If the value stack does not contain enough args, an error is thrown; this matches * behavior of the other protected call API functions.

46607. [value offset fieldByteLength noAssert], when ftype == DUK_FLD_VARINT

46608. placed in duk_heap_hdr_string

46609. * Main mark-and-sweep function. ** 'flags' represents the features requested by the caller. The current * heap->mark_and_sweep_base_flags is ORed automatically into the flags; * the base flags mask typically prevents certain mark-and-sweep operations * to avoid trouble.

46610. 'private'
46611. toJSON() can also be a lightfunc
46612. Special behavior for 'caller' property of (non-bound) function objects * and non-strict Arguments objects: if 'caller' -value- (!) is a strict * mode function, throw a TypeError (E5 Sections 15.3.5.4, 10.6). * Quite interestingly, a non-strict function with no formal arguments * will get an arguments object -without- special 'caller' behavior! ** The E5.1 spec is a bit ambiguous if this special behavior applies when * a bound function is the base value (not the 'caller' value): Section * 15.3.4.5 (describing bind()) states that [[Get]] for bound functions * matches that of Section 15.3.5.4 ([[Get]] for Function instances). * However, Section 13.3.5.4 has "NOTE: Function objects created using * Function.prototype.bind use the default [[Get]] internal method." * The current implementation assumes this means that bound functions * should not have the special [[Get]] behavior. ** The E5.1 spec is also a bit unclear if the TypeError throwing is * applied if the 'caller' value is a strict bound function. The * current implementation will throw even for both strict non-bound * and strict bound functions. ** See test-dev-strict-func-as-caller-prop-value.js for quite extensive * tests. ** This exotic behavior is disabled when the non-standard 'caller' property * is enabled, as it conflicts with the free use of 'caller'.
46613. move pivot out of the way
46614. final configuration
46615. -1 == not set, -2 == pending (next statement list)
46616. **comment:** temporary, must be signed and 32-bit to hold Unicode code points
 label: code-design
46617. NB: 'length' property is automatically updated by the array setup loop
46618. index for next new key ([0,e_next[are gc reachable)
46619. idx is unsigned, < 0 check is not necessary
46620. indirect opcode follows direct
46621. *toFixed(), toExponential(), toPrecision()
46622. * Helpers for UTF-8 handling ** For bytecode readers the duk_uint32_t and duk_int32_t types are correct * because they're used for more than just codepoints.
46623. follow prototype chain
46624. [value]
46625. XXX: incref by count (here 2 times)
46626. equals
46627. true even with reattach
46628. Fatal error handling, called e.g. when a longjmp() is needed but * lj.jmpbuf_ptr is NULL. fatal_func must never return; it's not * declared as "noreturn" because doing that for typedefs is a bit * challenging portability-wise.
46629. **comment:** XXX: very messy now, but works; clean up, remove unused variables (nominally * used so compiler doesn't complain).
 label: code-design
46630. * Note: type is treated as a field separate from flags, so some masking is * involved in the macros below.
46631. Compute final offset in seconds, positive if local time ahead of * UTC (returned value is UTC-to-local offset). ** difftime() returns a double, so coercion to int generates quite * a lot of code. Direct subtraction is not portable, however. * XXX: allow direct subtraction on known platforms.
46632. **comment:** XXX: this is a very narrow check, and doesn't cover * zeroes, subnormals, infinities, which compare normally.
 label: code-design
46633. Don't allow a zero divisor. Fast path check by * "verifying" with multiplication. Also avoid zero * dividend to avoid negative zero issues (0 / -1 = -0 * for instance).
46634. numeric value of a hex digit (also covers octal and decimal digits)
46635. DUK_TOK_MUL_EQ
46636. DUK_USE_DEBUG
46637. **comment:** XXX: optimize to use direct reads, i.e. avoid * value stack operations.
 label: code-design
46638. 0x00: finish (non-white) * 0x01: continue
46639. %lu
46640. [... arg1 ... argN]
46641. executor interrupt running (used to avoid nested interrupts)
46642. [... func this]
46643. Constants: variable size encoding.
46644. * Variable declarations. ** Unlike function declarations, variable declaration values don't get * assigned on entry. If a binding of the same name already exists, just * ignore it silently.
46645. **comment:** XXX: duk_small_uint_t would be enough for this loop
 label: code-design
46646. no res->strs[]
46647. * Debug dumping of duk_heap.
46648. Detach a debugger if attached (can be called multiple times) * safely.
46649. if 1, doing a string-to-number; else doing a number-to-string
46650. **comment:** Faster alternative: avoid making a temporary copy of tvptr_dst and use * fast incref/decref macros.
 label: code-design
46651. * duk_hbuffer allocation and freeing.
46652. stash to bottom of value stack to keep new_env reachable for duration of eval
46653. step 4.a
46654. Handle a RETURN opcode. Avoid using longjmp() for return handling because * it has a measurable performance impact in ordinary environments and an extreme * impact in Emscripten (GH-342). Return value is on value stack top.
46655. exports
46656. Note: must behave like a no-op with NULL and any pointer * returned from malloc/realloc with zero size.
46657. **comment:** XXX: use duk_is_valid_index() instead?
 label: code-design
46658. eat closing bracket
46659. * Object.isSealed() and Object.isFrozen() (E5 Sections 15.2.3.11, 15.2.3.13) ** Since the algorithms are similar, a helper provides both functions. * Freezing is essentially sealing + making plain properties non-writable. ** Note: all virtual (non-concrete) properties are currently non-configurable * and non-writable (and there are no accessor virtual properties), so they don't * need to be considered here now.
46660. * The switch statement is pretty messy to compile. * See the helper for details.
46661. re_ctx->captures at start and end of atom parsing. * Since 'captures' indicates highest capture number emitted * so far in a DUK_REOP_SAVE, the captures numbers saved by * the atom are: jstart_captures,end_captures].
46662. Loop check using a hybrid approach: a fixed-size visited[] array * with overflow in a loop check object.
46663. first value: comma must not precede the value
46664. actually used, non-NULL keys
46665. r <- (2 * b) * f
46666. [... re_obj input bc saved_buf res_obj]
46667. * Create wrapper object and serialize
46668. t1 <- t1 - s
46669. * Label handling ** Labels are initially added with flags prohibiting both break and continue. * When the statement type is finally uncovered (after potentially multiple * labels), all the labels are updated to allow/prohibit break and continue.
46670. 112.5%, i.e. new size less than 12.5% higher -> fast resize
46671. * URI handling
46672. * ASCII character constants ** C character literals like 'x' have a platform specific value and do * not match ASCII (UTF-8) values on e.g. EBCDIC platforms. So, use * these (admittedly awkward) constants instead. These constants must * also have signed values to avoid unexpected coercions in comparisons. **

<http://en.wikipedia.org/wiki/ASCII>

46673. 32-bit value
46674. string intern table (weak refs)
46675. **comment:** * Shared error message strings * * To minimize code footprint, try to share error messages inside Duktape * code. Modern compilers will do this automatically anyway, this is mostly * for older compilers.
label: code-design
46676. Convert indices to byte offsets.
46677. 'l'
46678. Fast path: source is a TypedArray (or any bufferobject).
46679. Note: object cannot be a finalizable unreachable object, as * they have been marked temporarily reachable for this round, * and are handled above.
46680. 'var'
46681. [args(n) [crud] formals arguments map mappednames]
46682. **comment:** Note: 'this' is not necessarily an Array object. The push() * algorithm is supposed to work for other kinds of objects too, * so the algorithm has e.g. an explicit update for the 'length' * property which is normally "magical" in arrays.
label: code-design
46683. **comment:** * XXX: array indices are mostly typed as duk_uint32_t here; duk_uarridx_t * might be more appropriate.
label: code-design
46684. 'pointer'
46685. Return value would be pointless: because throw_flag==1, we always * throw if the identifier doesn't resolve.
46686. PC points to next instruction, find offending PC, * PC == 0 for native code.
46687. DUK_UNICODE_H_INCLUDED
46688. **comment:** dead code, but ensures portability (see Linux man page notes)
label: code-design
46689. DUK_ALLOW_AUTO_SEMI_ALWAYS workaround
46690. end of valstack allocation (exclusive)
46691. previous value is assumed to be garbage, so don't touch it
46692. * Scan number and setup for Dragon4. * * The fast path case is detected during setup: an integer which * can be converted without rounding, no net exponent. The fast * path could be implemented as a separate scan, but may not really * be worth it: the multiplications for building 'f' are not * expensive when 'f' is small. * * The significand ('f') must contain enough bits of (apparent) * accuracy, so that Dragon4 will generate enough binary output digits. * For decimal numbers, this means generating a 20-digit significand, * which should yield enough practical accuracy to parse IEEE doubles. * In fact, the Ecmascript specification explicitly allows an * implementation to treat digits beyond 20 as zeroes (and even * to round the 20th digit upwards). For non-decimal numbers, the * appropriate number of digits has been precomputed for comparable * accuracy. * * Digit counts: * * [dig_lzero] * | * .+---[dig_prec]---. * || | * 0000123.456789012345678901234567890e+123456 * | | | | * `--+' `----[dig_frac]-----`-+` * | * [dig_whole] [dig_expt] * * dig_frac and dig_expt are -1 if not present * dig_lzero is only computed for whole number part * * Parsing state * * Parsing whole part dig_frac < 0 AND dig_expt < 0 * Parsing fraction part dig_frac >= 0 AND dig_expt < 0 * Parsing exponent part dig_expt >= 0 (dig_frac may be < 0 or >= 0) * * Note: in case we hit an implementation limit (like exponent range), * we should throw an error, NOT return NaN or Infinity. Even with * very large exponent (or significand) values the final result may be * finite, so NaN/Infinity would be incorrect.
46693. **comment:** suppress warning, not used
label: code-design
46694. Buffer/string -> compare contents.
46695. * Update cache entry (allocating if necessary), and move the * cache entry to the first place (in an "LRU" policy).
46696. finished jump
46697. string is ASCII, clen == blen
46698. DUK_USE_HOBJECT_HASH_PART || DUK_USE_STRTAB_PROBE
46699. * Since character data is only generated by decoding the source or by * the compiler itself, we rely on the input codepoints being correct * and avoid a check here. * * Character data can also come here through decoding of Unicode * escapes ("\\udead\\ubef") so all 16-bit unsigned values can be * present, even when the source file itself is strict UTF-8.
46700. **comment:** XXX: more emergency behavior, e.g. find smaller hash sizes etc
label: code-design
46701. * Copy array elements to new array part.
46702. For strings, special form for short lengths.
46703. Setup function properties.
46704. **comment:** * In a fast check we assume old_size equals old_used (i.e., existing * array is fully dense). * * Slow check if: * * (new_size - old_size) / old_size > limit * new_size - old_size > limit * old_size * new_size > (1 + limit) * old_size || limit' is 3 bits fixed point * new_size > (1 + (limit' / 8)) * old_size || * 8 * 8 * new_size > (8 + limit') * old_size || : 8 * new_size > (8 + limit') * (old_size / 8) * new_size > limit" * (old_size / 8) || limit' = 9 -> max 25% increase * arr_idx + 1 > limit" * (old_size / 8) * * This check doesn't work well for small values, so old_size is rounded * up for the check (and the '+ 1' of arr_idx can be ignored in practice): * * arr_idx > limit" * ((old_size + 7) / 8)
label: code-design
46705. **comment:** * A few notes on the algorithm: * - Terms are not allowed to begin with a period unless the term * is either '.' or '..'. This simplifies implementation (and * is within CommonJS modules specification). * - There are few output bound checks here. This is on purpose: * the resolution input is length checked and the output is never * longer than the input. The resolved output is written directly * over the input because it's never longer than the input at any * point in the algorithm. * * - Non-ASCII characters are processed as individual bytes and * need no special treatment. However, U+0000 terminates the * algorithm; this is not an issue because U+0000 is not a * desirable term character anyway.
label: code-design
46706. * Utilities
46707. * Helpers * * The fast path checks are done within a macro to ensure "inlining" * while the slow path actions use a helper (which won't typically be * inlined in size optimized builds).
46708. **comment:** If the platform doesn't support the entire Ecmascript range, we need * to return 0 so that the caller can fall back to the default formatter. * * For now, assume that if time_t is 8 bytes or more, the whole Ecmascript * range is supported. For smaller time_t values (4 bytes in practice), * assumes that the signed 32-bit range is supported. * * XXX: detect this more correctly per platform. The size of time_t is * probably not an accurate guarantee of strftime() supporting or not * supporting a large time range (the full Ecmascript range).
label: code-design
46709. **comment:** XXX: now that pc2line is used by the debugger quite heavily in * checked execution, this should be optimized to avoid value stack * and perhaps also implement some form of pc2line caching (see * future work in debugger.rst).
label: code-design
46710. * Case clause. * * Note: cannot use reg_case as a temp register (for SEQ target) * because it may be a constant.
46711. **comment:** XXX: not sure what the correct semantic details are here, * e.g. handling of missing values (gaps), handling of non-array * trap results, etc. * * For keys, we simply skip non-string keys which seems to be * consistent with how e.g. Object.keys() will process proxy trap * results (ES6, Section 19.1.2.14).
label: code-design
46712. **comment:** XXX: 'copy properties' API call?
label: code-design
46713. * Determine whether to use the constructor return value as the created * object instance or not.
46714. Stack top contains plain value
46715. make current token the previous; need to fiddle with valstack "backing store"
46716. '{_func:true}'
46717. prediction: portable variant using doubles if 64-bit values not available
46718. [... arr val]
46719. Casting convenience.

46720. The stack has a variable shape here, so force it to the * desired one explicitly.
46721. **comment:** XXX: does not work if thr->catchstack is NULL
 label: code-design
46722. Object.getOwnPropertyNames
46723. 'undefined'
46724. **comment:** XXX: with 'caller' property the callstack would need * to be unwound to update the 'caller' properties of * functions in the callstack.
 label: code-design
46725. Overestimate required size; debug code so not critical to be tight.
46726. **comment:** no special formatting
 label: code-design
46727. proplist is very rare
46728. * Handling DUK_OP_ADD this way is more compact (experimentally) * than a separate case with separate argument decoding.
46729. currently processing messages or breakpoints: don't enter message processing recursively (e.g. no breakpoints when processing debugger eval)
46730. DUK_USE_JC
46731. function refers to 'arguments' identifier
46732. don't want an intermediate exposed state with invalid pc
46733. Use loop to minimize code size of relookup after bound function case
46734. 'Undefined'
46735. **comment:** * Variable already declared, ignore re-declaration. * The only exception is the updated behavior of E5.1 for * global function declarations, E5.1 Section 10.5, step 5.e. * This behavior does not apply to global variable declarations.
 label: code-design
46736. **comment:** Avoid fake finalization for the duk__refcount_fake_finalizer function * itself, otherwise we're in infinite recursion.
 label: code-design
46737. **comment:** Borrow is detected based on wrapping which relies on exact 32-bit * types.
 label: code-design
46738. if abandon_array, new_a_size must be 0
46739. Parse enumeration target and initialize enumerator. For 'null' and 'undefined', * INITENUM will creates a 'null' enumerator which works like an empty enumerator * (E5 Section 12.6.4, step 3). Note that INITENUM requires the value to be in a * register (constant not allowed).
46740. pointer is aligned, guaranteed for fixed buffer
46741. See MPUTOBJ comments above.
46742. Pause on the next opcode executed. This is always safe to do even * inside the debugger message loop: the interrupt counter will be reset * to its proper value when the message loop exits.
46743. Ensure there is internal valstack spare before we exit; this may * throw an alloc error. The same guaranteed size must be available * as before the call. This is not optimal now: we store the valstack * allocated size during entry; this value may be higher than the * minimal guarantee for an application.
46744. '\xffPc2line'
46745. **comment:** XXX: better coercion
 label: code-design
46746. guaranteed to finish, as hash is never full
46747. shared flags for a subset of types
46748. nothing to incref
46749. The short string/integer initial bytes starting from 0x60 don't have * defines now.
46750. **comment:** XXX: use a helper for prototype traversal; no loop check here
 label: code-design
46751. FunctionDeclaration: not strictly a statement but handled as such. * * O(depth^2) parse count for inner functions is handled by recording a * lexer offset on the first compilation pass, so that the function can * be efficiently skipped on the second pass. This is encapsulated into * duk_parse_func_like_fnnum().
46752. unpack args
46753. hash lookup
46754. * Cleanup: restore original function, restore valstack state.
46755. don't touch property attributes or hash part
46756. Note: duk_uint8_t type yields larger code
46757. * The error object has been augmented with a traceback and other * info from its creation point -- usually another thread. The * error handler is called here right before throwing, but it also * runs in the resumer's thread. It might be nice to get a traceback * from the resumee but this is not the case now.
46758. Push function object, init flags etc. This must match * duk_js_push_closure() quite carefully.
46759. return arg as-is
46760. NULL accepted
46761. Lightfunc, not blamed now.
46762. 0 = no throw
46763. **comment:** XXX: macros
 label: code-design
46764. Interrupt counter for triggering a slow path check for execution * timeout, debugger interaction such as breakpoints, etc. The value * is valid for the current running thread, and both the init and * counter values are copied whenever a thread switch occurs. It's * important for the counter to be conveniently accessible for the * bytecode executor inner loop for performance reasons.
46765. start value for current countdown
46766. maximum recursion depth for loop detection stacks
46767. us
46768. [... holder name val]
46769. Allow automatic detection of hex base ("0x" or "0X" prefix), * overrides radix argument and forces integer mode.
46770. * Init arguments properties, map, etc.
46771. **comment:** NUL terminator handling doesn't matter here
 label: code-design
46772. **comment:** XXX: error handling is incomplete. It would be cleanest if * there was a setjmp catchpoint, so that all init code could * freely throw errors. If that were the case, the return code * passing here could be removed.
 label: code-design
46773. keep in on valstack, use borrowed ref below
46774. [... Logger clog logfunc clog(=this) msg]
46775. -> [... func this arg1 ... argN _Args]
46776. * Found * * Arguments object has exotic post-processing, see E5 Section 10.6, * description of [[GetOwnProperty]] variant for arguments.
46777. compiler's responsibility
46778. [... func this arg1 ... argN] (not tail call) * [this | arg1 ... argN] (tail call) * * idx_args updated to match
46779. func is NULL for lightfuncs
46780. shared exit path now
46781. Finish the wrapped module source. Force module.filename as the * function .filename so it gets set for functions defined within a * module. This also ensures loggers created within the module get * the module ID (or overridden filename) as their default logger name. * (Note capitalization: .filename matches Node.js while .fileName is * used elsewhere in Duktape.)
46782. **comment:** This limitation would be fixable but adds unnecessary complexity.
 label: code-design
46783. -> [O toString O]
46784. * Finish

46785. -> [timeval this timeval]
46786. cached: valstack_end - valstack (in entries, not bytes)
46787. **comment:** XXX: awkward, especially the bunch of separate output values -> output struct?
 label: code-design
46788. According to E5.1 Section 15.4.4.4 nonexistent trailing * elements do not affect 'length' of the result. Test262 * and other engines disagree, so update idx_last here too.
46789. * Bitstream encoder.
46790. * Once the whole refzero cascade has been freed, check for * a voluntary mark-and-sweep.
46791. Resolve start/end offset as element indices first; arguments * at idx_start/idx_end are element offsets. Working with element * indices first also avoids potential for wrapping.
46792. IdentifierStart production with ASCII and non-BMP excluded
46793. temp object for tracking / detecting duplicate keys
46794. negative
46795. * First create all built-in bare objects on the empty valstack. ** Built-ins in the index range [0,DUK_NUM_BUILTINS-1] have value * stack indices matching their eventual thr->builtins[] index. ** Built-ins in the index range [DUK_NUM_BUILTINS,DUK_NUM_ALL_BUILTINS] * will exist on the value stack during init but won't be placed * into thr->builtins[]. These are objects referenced in some way * from thr->builtins[] roots but which don't need to be indexed by * Duktape through thr->builtins[] (e.g. user custom objects).
46796. DUK_BUFOBJ_INT8ARRAY
46797. DUK_USE_ROM_STRINGS
46798. get pointer offsets for tweaking below
46799. Note: not normalized, but duk_push_number() will normalize
46800. NULL if not an object
46801. * Exposed number-to-string API * * Input: [number] * Output: [string]
46802. XXX: could add fast path for u8 compatible views
46803. Note2
46804. 'resume'
46805. should never happen for a strict callee
46806. **comment:** XXX: This is VERY inefficient now, and should be e.g. a * binary search or perfect hash, to be fixed.
 label: code-design
46807. [... env]
46808. Lookup an identifier name in the current varmap, indicating whether the * identifier is register-bound and if not, allocating a constant for the * identifier name. Returns 1 if register-bound, 0 otherwise. Caller can * also check (out_reg_varbind >= 0) to check whether or not identifier is * register bound. The caller must NOT use out_rc_varname at all unless * return code is 0 or out_reg_varbind is < 0; this is because out_rc_varname * is unsigned and doesn't have a "unused" / none value.
46809. 'export'
46810. DUK_ERR_URI_ERROR: no macros needed
46811. array case is handled comprehensively above
46812. * Bytecode executor call. ** Execute bytecode, handling any recursive function calls and * thread resumptions. Returns when execution would return from * the entry level activation. When the executor returns, a * single return value is left on the stack top. ** The only possible longjmp() is an error (DUK_LJ_TYPE_THROW), * other types are handled internally by the executor.
46813. **comment:** XXX: refactor into an internal helper, pretty awkward
 label: code-design
46814. Allocate a new valstack. ** Note: cannot use a plain DUK_REALLOC() because a mark-and-sweep may * invalidate the original thr->valstack base pointer inside the realloc * process. See doc/memory-management.rst.
46815. Note that this is the same operation for strict and loose equality: * - E5 Section 11.9.3, step 1.c (loose) * - E5 Section 11.9.6, step 4 (strict)
46816. force_no_namebind
46817. DUK_TOK_NEW
46818. * Breakpoint and step state checks
46819. **comment:** XXX: this behavior is quite useless now; it would be nice to be able * to create pointer values from e.g. numbers or strings. Numbers are * problematic on 64-bit platforms though. Hex encoded strings?
 label: code-design
46820. global object doesn't have array part
46821. **comment:** Then evaluate RHS fully (its value becomes the expression value too). * Technically we'd need the side effect safety check here too, but because * we always throw using INVLHS the result doesn't matter.
 label: code-design
46822. **comment:** XXX: specific getter
 label: code-design
46823. [... escape_source bytecode]
46824. First we need to insert a jump in the middle of previously * emitted code to get the control flow right. No jumps can * cross the position where the jump is inserted. See doc/compiler.rst * for discussion on the intricacies of control flow and side effects * for variants 3 and 4.
46825. Cannot simulate individual finalizers because finalize_list only * contains objects with actual finalizers. But simulate side effects * from finalization by doing a bogus function call and resizing the * stacks.
46826. lastIndex is initialized to zero by new RegExp()
46827. **comment:** Note: we could return the hash index here too, but it's not * needed right now.
 label: code-design
46828. >>>
46829. [... closure template formals]
46830. inherit
46831. filename being compiled (ends up in functions' _filename' property)
46832. filter out flags from exptrop rbp_flags here to save space
46833. **comment:** * Debugging macros, DUK_DPRINT() and its variants in particular. ** DUK_DPRINT() allows formatted debug prints, and supports standard * and Duktape specific formatters. See duk_debug_vsnprintf.c for details. ** DUK_D(x), DUK_DD(x), and DUK_DDD(x) are used together with log macros * for technical reasons. They are concretely used to hide 'x' from the * compiler when the corresponding log level is disabled. This allows * clean builds on non-C99 compilers, at the cost of more verbose code. * Examples: ** DUK_D(DUK_DPRINT("foo")); * DUK_DD(DUK_DPRINT("foo")); * DUK_DDD(DUK_DDDPRINT("foo")); * DUK_DDD(DUK_DDDPRINT("foo")); * * This approach is preferable to the old "double parentheses" hack because * double parentheses make the C99 solution worse: __FILE__ and __LINE__ can * no longer be added transparently without going through globals, which * works poorly with threading.
 label: code-design
46834. prop index in 'array part', < 0 if not there
46835. caller handles sign change
46836. **comment:** load factor too low or high, count actually used entries and resize
 label: code-design
46837. * Coercion and fast path processing.
46838. Not strictly necessary because if key == NULL, flag MUST be ignored.
46839. make the new thread reachable
46840. XXX: print built-ins array?
46841. temp reg
46842. * Externs and prototypes
46843. no incref

46844. Test signaling NaN and alias assignment in all endianness combinations.
46845. 1 GB
46846. XXX: fast path for array arguments?
46847. E5.1 Section 15.1.3.3: uriReserved + uriUnescaped + '#'
46848. DUK_USE_PACKED_TVAL
46849. **comment:** * Enumeration semantics come from for-in statement, E5 Section 12.6.4. * If called with 'null' or 'undefined', this opcode returns 'null' as * the enumerator, which is special cased in NEXTENUM. This simplifies * the compiler part
 label: code-design
46850. Insert a reserved area somewhere in the buffer; caller fills it. * Evaluates to a (duk_uint_t *) pointing to the start of the reserved * area for convenience.
46851. values for the state field
46852. How large a loop detection stack to use
46853. pop enum
46854. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)
46855. <
46856. [... closure template undefined]
46857. **comment:** * To determine whether to use an optimized Ecmascript-to-Ecmascript * call, we need to know whether the final, non-bound function is an * Ecmascript function. * * This is now implemented so that we start to do an ecma-to-ecma call * setup which will resolve the bound chain as the first thing. If the * final function is not eligible, the return value indicates that the * ecma-to-ecma call is not possible. The setup will overwrite the call * target at DUK__REGP(idx) with the final, non-bound function (which * may be a lightfunc), and fudge arguments if necessary. * * XXX: If an ecma-to-ecma call is not possible, this initial call * setup will do bound function chain resolution but won't do the * "effective this binding" resolution which is quite confusing. * Perhaps add a helper for doing bound function and effective this * binding resolution - and call that explicitly? Ecma-to-ecma call * setup and normal function handling can then assume this prestep has * been done by the caller.
 label: code-design
46858. dynamic buffer ops
46859. not found in 'curr', next in prototype chain; impose max depth
46860. 13: getUTCDate
46861. **comment:** Truncate to 16 bits here, so that a computed hash can be compared * against a hash stored in a 16-bit field.
 label: code-design
46862. Buffer has virtual properties similar to string, but indexed values * are numbers, not 1-byte buffers/strings which would perform badly.
46863. [... this name]
46864. * Date/time parsing helper. * * Parse a datetime string into a time value. We must first try to parse * the input according to the standard format in E5.1 Section 15.9.1.15. * If that fails, we can try to parse using custom parsing, which can * either be platform neutral (custom code) or platform specific (using * existing platform API calls). * * Note in particular that we must parse whatever toString(), toUTCString(), * and toISOString() can produce; see E5.1 Section 15.9.4.2. * * Returns 1 to allow tail calling. * * There is much room for improvement here with respect to supporting * alternative datetime formats. For instance, V8 parses '2012-01-01' as * UTC and '2012/01/01' as local time.
46865. **comment:** XXX: check flags
 label: requirement
46866. ASCII (and EOF) fast path -- quick accept and reject
46867. 0xc0...0xcf
46868. '-' as a range indicator
46869. -> [... enum_target res handler undefined target]
46870. [arg1 ... argN this loggerLevel loggerName 'fmt' arg]
46871. "123.456"
46872. Helper for creating the arguments object and adding it to the env record * on top of the value stack. This helper has a very strict dependency on * the shape of the input stack.
46873. If an implicit return value is needed by caller, it must be * initialized to 'undefined' because we don't know whether any * non-empty (where "empty" is a continuation type, and different * from an empty statement) statements will be executed. * * However, since 1st pass is a throwaway one, no need to emit * it here.
46874. 'false'
46875. Some internal code now assumes that all duk_uint_t values * can be expressed with a duk_size_t.
46876. data property or accessor without getter
46877. * Args validation
46878. DUK__TAG_NUMBER is intentionally first, as it is the default clause in code * to support the 8-byte representation. Further, it is a non-heap-allocated * type so it should come before DUK__TAG_STRING. Finally, it should not break * the tag value ranges covered by case-clauses in a switch-case.
46879. * String built-ins
46880. * Clear (reachable) flags of refzero work list.
46881. **comment:** * Number should already be in NaN-normalized form, but let's * normalize anyway.
 label: code-design
46882. -> [... obj key trap handler]
46883. catchstack
46884. * C call recursion depth check, which provides a reasonable upper * bound on maximum C stack size (arbitrary C stack growth is only * possible by recursive handle_call / handle_safe_call calls).
46885. Call sites don't need the result length so it's not accumulated.
46886. non-zero: make copy
46887. Return non-zero (true) if we have a good reason to believe * the notify was delivered; if we're still attached at least * a transport error was not indicated by the transport write * callback. This is not a 100% guarantee of course.
46888. fromPresent = false
46889. Check that there's room to push one value.
46890. comma check
46891. backtrack to last output slash (dups already eliminated)
46892. **comment:** Fastint range is signed 48-bit so longest value is $-2^{47} = -140737488355328$ * (16 chars long), longest signed 64-bit value is $-2^{63} = -9223372036854775808$ * (20 chars long). Alloc space for 64-bit range to be safe.
 label: code-design
46893. Flags for call handling.
46894. jump to end
46895. * RegExp built-ins
46896. allocation size
46897. no -> decref members, then free
46898. If any non-Array value had enumerable virtual own * properties, they should be serialized here. Standard * types don't.
46899. XXX: C recursion limit if proxies are allowed as handler/target values
46900. try part
46901. 'caller' must only take on 'null' or function value
46902. sign doesn't matter when writing
46903. **comment:** XXX: use macros for the repetitive tval/refcount handling.
 label: code-design
46904. **comment:** Builtin-objects; may or may not be shared with other threads, * threads existing in different "compartments" will have different * built-ins. Must be stored on a per-thread basis because there * is no intermediate structure for a thread group / compartment. * This takes quite a lot of space, currently $43 \times 4 = 172$

bytes on * 32-bit platforms. ** In some cases the builtins array could be ROM based, but it's * sometimes edited (e.g. for sandboxing) so it's better to keep * this array in RAM.

label: code-design

46905. **comment:** unused

label: code-design

46906. * Other file level defines

46907. Fast path for numbers (one of which may be a fastint)

46908. maximum length for a SKIP-1 diffstream: 35 bits per entry, rounded up to bytes

46909. [enum_target enum res]

46910. orig value

46911. **comment:** XXX use get_tval ptr, more efficient

label: code-design

46912. See comments in duk_pcall().

46913. The 'left' value must not be a register bound variable * because it may be mutated during the rest of the expression * and E5.1 Section 11.2.1 specifies the order of evaluation * so that the base value is evaluated first. * See: test-bug-nested-prop-mutate.js.

46914. A bit tricky overflow test, see doc/code-issues.rst.

46915. key must not already exist in entry part

46916. Digit generation

46917. reset function state (prepare for pass 2)

46918. +0 = catch

46919. avoid side effects!

46920. **comment:** NOTE: "get" and "set" are not officially ReservedWords and the lexer * currently treats them always like ordinary identifiers (DUK_TOK_GET * and DUK_TOK_SET are unused). They need to be detected based on the * identifier string content.

label: code-design

46921. **comment:** XXX: regetting the pointer may be overkill - we're writing * to a side-effect free array here.

label: code-design

46922. * Because buffer values may be looped over and read/written * from, an array index fast path is important.

46923. duk_small_uint_fast_t c = DUK_DEC_C(ins);

46924. For all duk_hbufferobjects, get the plain buffer inside * without making a copy. This is compatible with Duktape 1.2 * but means that a slice/view information is ignored and the * full underlying buffer is returned. ** If called as a constructor, a new Duktape.Buffer object * pointing to the same plain buffer is created below.

46925. Copy values by index reads and writes. Let virtual * property handling take care of coercion.

46926. **comment:** No coercions or other side effects, so safe

label: requirement

46927. 0x20-0x2f

46928. may become sparse...

46929. Pointers may be NULL for a while when 'buf' size is zero and before any * ENSURE calls have been made. Once an ENSURE has been made, the pointers * are required to be non-NUL so that it's always valid to use memcpy() and * memmove(), even for zero size.

46930. * Create and throw an Ecmascript error object based on a code and a message. ** Used when we throw errors internally. Ecmascript generated error objects * are created by Ecmascript code, and the throwing is handled by the bytecode * executor.

46931. XXXXXX--

46932. lightfuncs are treated like objects and not coerced

46933. jump to end or else part

46934. **comment:** Registers 'bc' and 'bc + 1' are written in longjmp handling * and if their previous values (which are temporaries) become * unreachable -and- have a finalizer, there'll be a function * call during error handling which is not supported now (GH-287). * Ensure that both 'bc' and 'bc + 1' have primitive values to * guarantee no finalizer calls in error handling. Scrubbing also * ensures finalizers for the previous values run here rather than * later. Error handling related values are also written to 'bc' * and 'bc + 1' but those values never become unreachable during * error handling, so there's no side effect problem even if the * error value has a finalizer.

label: code-design

46935. no_block

46936. The original value needs to be preserved for filter(), hence * this funny order. We can't re-get the value because of side * effects.

46937. for duk_error_augment.c

46938. just one 'int' for C++ exceptions

46939. '~~'

46940. reset voluntary gc trigger count

46941. If the separator is a RegExp, make a "clone" of it. The specification * algorithm calls [[Match]] directly for specific indices; we emulate this * by tweaking lastIndex and using a "force global" variant of duk_regexp_match() * which will use global-style matching even when the RegExp itself is non-global.

46942. * Defines for JSON, especially duk_bi_json.c.

46943. * Fast path tables

46944. -> [... handler trap]

46945. DUK_TOK_ALSHIFT_EQ

46946. Note: assumes 'data' is always a fixed buffer

46947. * Stringtable entry for fixed size stringtable

46948. sometimes stack and array indices need to go on the stack

46949. hash size ratio goal, must match genhashsizes.py

46950. * Heap structure. ** Heap contains allocated heap objects, interned strings, and built-in * strings for one or more threads.

46951. Require a lot of stack to force a value stack grow/shrink.

46952. **comment:** XXX: lastIndex handling produces a lot of asm

label: code-design

46953. DUK_DEBUG_H_INCLUDED

46954. input size is good output starting point

46955. XXX: Current putvar implementation doesn't have a success flag, * add one and send to debug client?

46956. **comment:** must be able to emit code, alloc consts, etc.

label: code-design

46957. **comment:** Should never happen but avoid infinite loop just in case.

label: code-design

46958. unchanged from Duktape.Thread.yield()

46959. value1 -> return value, pseudo-type to indicate a return continuation (for ENDFIN)

46960. Negative offsets cause a RangeError.

46961. side effect free

46962. main expression parser function

46963. Always overflow.

46964. may happen if size is very close to 2^32-1

46965. fail: restore saves

46966. DUK_USE_DATE_PRS_GETDATE

46967. needed for stepping

46968. **comment:** XXX: could map/decode be unified with duk_unicode_support.c code? * Case conversion needs also the character surroundings though.

label: code-design

46969. clen == blen -> pure ascii

46970. * The attempt to allocate may cause a GC. Such a GC must not attempt to resize * the stringtable (though it can be swept); finalizer execution and object * compaction must also be postponed to avoid the pressure to add strings to the * string table. Call site must prevent these.

46971. current setjmp() catchpoint

46972. Result is undefined.

46973. [... proplist]

46974. This may throw an error though not for valid E5 strings.

46975. * Marking functions for heap types: mark children recursively

46976. * Any catch bindings ("catch (e)") also affect identifier binding. * * Currently, the varmap is modified for the duration of the catch * clause to ensure any identifier accesses with the catch variable * name will use slow path.

46977. XXX: set with duk_hobject_set_length() when tracedata is filled directly

46978. **comment:** XXX: optimize by creating array into correct size directly, and * operating on the array part directly; values can be memcpy()'d from * value stack directly as long as refcounts are increased.
label: code-design

46979. Slow path.

46980. 'compile'

46981. **comment:** XXX: does not work if thr->catchstack is allocated but lowest pointer
label: code-design

46982. DUK_TOK_RCURLY

46983. **comment:** XXX: check .caller writability?
label: code-design

46984. replacement handler

46985. $2^{31-1} \approx 2G$ properties

46986. part of string

46987. Bytecode instructions: endian conversion needed unless * platform is big endian.

46988. DUK_TOK_EXTENDS

46989. XXX: optimize reconfig valstack operations so that resize, clamp, and setting * top are combined into one pass.

46990. fastint downgrade check for return values

46991. **comment:** bytes of indent still needed
label: code-design

46992. DUK_JS_COMPILER_H_INCLUDED

46993. nop

46994. Restore the previous setjmp catcher so that any error in * error handling will propagate outwards rather than re-enter * the same handler. However, the error handling path must be * designed to be error free so that sandboxing guarantees are * reliable, see e.g. <https://github.com/svaarala/duktape/issues/476>.

46995. Double error

46996. * Valstack spare check

46997. This would be pointless: unexpected type and lightfunc would both return NULL

46998. For internal use: get non-accessor entry value

46999. trailing comma followed by rcurly

47000. Prevent mark-and-sweep for the pending finalizers, also prevents * refzero handling from moving objects away from the heap_allocated * list. (The flag meaning is slightly abused here.)

47001. [... value this_binding]

47002. DUK_TOK_INCREMENT

47003. * Shared handling of binary operations * * args = (is_extrap << 16) + (opcode << 8) + rbp

47004. insert code for matching the remainder - infinite or finite

47005. **comment:** XXX: shared api error strings, and perhaps even throw code for rare cases?
label: code-design

47006. Although there are writable virtual properties (e.g. plain buffer * and buffer object number indices), they are handled before we come * here.

47007. * Fast paths

47008. **comment:** XXX: static alloc is OK until separate chaining stringtable * resizing is implemented.
label: code-design

47009. truncate towards zero

47010. 37: setUTCFullYear

47011. **comment:** Function name: missing/undefined is mapped to empty string, * otherwise coerce to string.
label: code-design

47012. * Assert helpers

47013. resume lookup from target

47014. **comment:** XXX: assume these?
label: code-design

47015. When creating built-ins, some of the built-ins may not be set * and we want to tolerate that when throwing errors.

47016. standard prototype

47017. **comment:** XXX: consolidated algorithm step 15.f -> redundant?
label: code-design

47018. flags: "gm"

47019. no need to decref, as previous value is 'undefined'

47020. may need unwind

47021. catch part will catch

47022. -> [... name index]

47023. * Node.js Buffer: toString([encoding], [start], [end])

47024. * Ecmascript compiler.

47025. Allow fraction part

47026. explicit 'this' binding, see GH-164

47027. Initial bytecode size allocation.

47028. XXX: patch initial size afterwards?

47029. already zero

47030. late lookup, avoid side effects

47031. Return value: $<0 \iff x < y \iff 0 \iff x == y \iff >0 \iff x > y$

47032. match against rule part/sep bits

47033. 'd' and 'res' agree here

47034. no value, pseudo-type to indicate a normal continuation (for ENDFIN)

47035. Eval code doesn't need an automatic .prototype object.

47036. The coercion potentially invokes user .valueOf() and .toString() * but can't result in a function value because [[DefaultValue]] would * reject such a result: test-dev-json-stringify-coercion-1.js.

47037. never here

47038. **comment:** Macro for creating flag initializer from a class number. * Unsigned type cast is needed to avoid warnings about coercing * a signed integer to an unsigned one; the largest class values * have the highest bit (bit 31) set which causes this.
label: code-design

47039. Note that 'l' and 'r' may cross, i.e. $r < l$

47040. For internal use: get non-accessor entry value and attributes

label: code-design

47106. Unwind catchstack entries referring to the callstack entry we're reusing

47107. **comment:** XXX: many operations actually want toforcedtemp() -- brand new temp?

label: code-design

47108. e.g. DUK_OP_POSTINCR

47109. 'Uint8ClampedArray'

47110. No copy, leave zero bytes in the buffer. There's no * ambiguity with Float32/Float64 because zero bytes also * represent 0.0.

47111. no typedef

47112. [... func_template]

47113. Note: any entries above the callstack top are garbage and not zeroed. * Also topmost activation idx_retval is garbage (not zeroed), and must * be ignored.

47114. [arg1 ... argN obj length]

47115. Previous value of 'func' caller, restored when unwound. Only in use * when 'func' is non-strict.

47116. When doing string-to-number, lowest_mantissa is always 0 so * the exponent check, while incorrect, won't matter.

47117. If the value has a prototype loop, it's critical not to * throw here. Instead, assume the value is not to be * augmented.

47118. **comment:** unused with some debug level combinations

label: code-design

47119. Line number range for function. Needed during debugging to * determine active breakpoints.

47120. **comment:** slow init, do only for slow path cases

label: code-design

47121. DUK_BUFOBJ_DUKTAPE_BUFFER

47122. positive

47123. **comment:** XXX: Here we have a nasty dependency: the need to manipulate * the callstack means that catchstack must always be unwound by * the caller before unwinding the callstack. This should be fixed * later.

label: code-design

47124. e.g. DUK_OP_PREINCP

47125. DUK_JS_H_INCLUDED

47126. DUK_USE_DATE_NOW_WINDOWS

47127. !DUK_SINGLE_FILE

47128. this is added to checks to allow for Duktape * API calls in addition to function's own use

47129. [... key_obj key]

47130. - A is a source register (it's not a write target, but used * to identify the target object) but can be shuffled. * - B cannot be shuffled normally because it identifies a range * of registers, the emitter has special handling for this * (the "no shuffle" flag must not be set). * - C is a non-register number and cannot be shuffled, but * never needs to be.

47131. is_setget

47132. flags always encodes to 1 byte

47133. Ensure space for 1:1 output plus one escape.

47134. [... obj key value]

47135. * Byte-based matching would be possible for case-sensitive * matching but not for case-insensitive matching. So, we * match by decoding the input and bytecode character normally. ** Bytecode characters are assumed to be already canonicalized. * Input characters are canonicalized automatically by * duk_inp_get_cp() if necessary. ** There is no opcode for matching multiple characters. The * regexp compiler has trouble joining strings efficiently * during compilation. See doc/regexp.rst for more discussion.

47136. C array of duk_labelinfo

47137. **comment:** XXX: double evaluation for 'tv' argument.

label: code-design

47138. with stack depth (affects identifier lookups)

47139. [... Duktape.modSearch resolved_id last_comp fresh_require exports module]

47140. * Detect recursive invocation

47141. **comment:** known to be 32-bit

label: code-design

47142. * Directive prologue tracking.

47143. Handle lightfuncs through object coercion for now.

47144. exponent 0

47145. Identifier found in registers (always non-deletable) * or declarative environment record and non-configurable.

47146. duk_handle_ecma_call_setup: setup for a tail call

47147. match: keep wiped/resaved values

47148. E5 Section 11.7.2, steps 7 and 8

47149. **comment:** * Since there are no references in the catcher structure, * unwinding is quite simple. The only thing we need to * look out for is popping a possible lexical environment * established for an active catch clause.

label: code-design

47150. * String cache. ** Provides a cache to optimize indexed string lookups. The cache keeps * track of (byte offset, char offset) states for a fixed number of strings. * Otherwise we'd need to scan from either end of the string, as we store * strings in (extended) UTF-8.

47151. heap thread, used internally and for finalization

47152. Don't allow a constant for the object (even for a number etc), as * it goes into the 'A' field of the opcode.

47153. For now all calls except Ecma-to-Ecma calls prevent a yield.

47154. If tv1==tv2 this is a NOP, no check is needed

47155. Dispatch loop.

47156. DUK_TOK_SUB

47157. Shared Windows helpers.

47158. This is a pretty awkward control flow, but we need to recheck the * key coercion here.

47159. catches EOF and invalid digits

47160. If thr->ptr_curr_pc is set, sync curr_pc to act->pc. Then NULL * thr->ptr_curr_pc so that it's not accidentally used with an incorrect * activation when side effects occur. If we end up not making the * call we must restore the value.

47161. **comment:** XXX: better macro for DUK_TVAL_IS_UNDEFINED_OR_NULL(tv)

label: code-design

47162. * Call the constructor function (called in "constructor mode").

47163. minimum prime

47164. set by atom case clauses

47165. [... error func fileName]

47166. reset callstack limit

47167. right paren eaten

47168. '['

47169. idx_desc

47170. E5 Section 11.7.1, steps 7 and 8

47171. coerced value is updated to value stack even when RangeError thrown

47172. Note: special DUK_EMIT_FLAG_B_IS_TARGETSOURCE * used to indicate that B is both a source and a * target register. When shuffled, it needs to be * both input and output shuffled.

47173. * Number is special because it doesn't have a specific * tag in the 8-byte representation.

47174. Align 'p' to 4; the input data may have arbitrary alignment. * End of string check not needed because blen >= 16.

47175. Expression_opt
47176. Exact halfway, round to even.
47177. * Arguments check
47178. **comment:** Very minimal inlining to handle common idioms '!0' and '!1', * and also boolean arguments like '!false' and '!true'.
label: code-design
47179. name is empty -> return message
47180. * Rescue or free.
47181. grow by at most one
47182. [A B C D E F G H] rel_index = 2, del_count 3, item count 3 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)
47183. **comment:** XXX: helper, rely on Boolean.prototype as being non-writable, non-configurable
label: code-design
47184. Note: proxy handling must happen before key is string coerced.
47185. DUK_HBUFFEROBJECT_H_INCLUDED
47186. [... res]
47187. Note: unconditional throw
47188. DUK_ERRMSG_H_INCLUDED
47189. Neutered buffer, zero length seems * like good behavior here.
47190. no mark-and-sweep gc
47191. * DUK_BSWAP macros
47192. literals (E5 Section 7.8), except null, true, false, which are treated * like reserved words (above).
47193. loggerName
47194. will result in undefined
47195. DUK_TOK_QUESTION
47196. * Delayed initialization only occurs for 'NEWENV' functions.
47197. Allow automatic detection of octal base, overrides radix * argument and forces integer mode.
47198. Load constants onto value stack but don't yet copy to buffer.
47199. [hobject props enum(props) key desc]
47200. 'Float32Array'
47201. eat comma
47202. range
47203. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT32
47204. Since stack indices are not reliable, we can't do anything useful * here. Invoke the existing setjmp catcher, or if it doesn't exist, * call the fatal error handler.
47205. [A B C D E F G H] rel_index = 2, del_count 3, item count 4 * -> [A B F G H] (conceptual intermediate step) * -> [A B . . . F G H] (placeholder marked) * [A B C D E F G H] (actual result at this point)
47206. Commands and notifies initiated by Duktape.
47207. DUK_TOK_GT
47208. * Stringtable
47209. offset is now target of the pending split (right after jump)
47210. U+03A3 = GREEK CAPITAL LETTER SIGMA
47211. always 0 args
47212. exit bytecode executor by rethrowing an error to caller
47213. Note: e_next indicates the number of gc-reachable entries * in the entry part, and also indicates the index where the * next new property would be inserted. It does *not* indicate * the number of non-NULL keys present in the object. That * value could be counted separately but requires a pass through * the key list.
47214. initialize built-ins - either by copying or creating new ones
47215. Note: thr->catchstack_top may be 0, so that cat < thr->catchstack * initially. This is OK and intended.
47216. SCANBUILD: scan-build complains here about assigned value * being garbage or undefined. This is correct but operating * on undefined values has no ill effect and is ignored by the * caller in the case where this happens.
47217. IsAccessorDescriptor(desc) == true
47218. avoid issues with relocation
47219. avoid recursive string table call
47220. res.x1 -> res.x2
47221. -> [... x y fn]
47222. 'aint' result as complex -- this is conservative, * as lookaheads do not backtrack.
47223. LHS is already side effect free
47224. **comment:** XXX: hack, remove when const lookup is not O(n)
label: code-design
47225. Here the specification requires correct array index enumeration * which is a bit tricky for sparse arrays (it is handled by the * enum setup code). We now enumerate ancestors too, although the * specification is not very clear on whether that is required.
47226. object has an array part (a_size may still be 0)
47227. 0x20...0x2f
47228. eat dup slashes
47229. Get the current data pointer (caller must ensure buf != NULL) as a * duk_uint8_t ptr.
47230. initial estimate based on format string
47231. enable exotic behaviors last
47232. **comment:** if 'src < src_end_safe', safe to read 4 bytes
label: requirement
47233. **comment:** Straightforward algorithm, makes fewer compiler assumptions.
label: code-design
47234. [obj key desc]
47235. **comment:** Convenience for some situations; the above macros don't allow NULLs * for performance reasons.
label: code-design
47236. [... env callee varmap key val]
47237. currently required
47238. * Yield the current thread. * * The thread must be in yieldable state: it must have a resumer, and there * must not be any yield-preventing calls (native calls and constructor calls, * currently) in the thread's call stack (otherwise a resume would not be * possible later). This method must be called from an Ecmascript function. * * Args: * - value * - isError (defaults to false) * * Note: yield and resume handling is currently asymmetric.
47239. **comment:** No need to length check string: it will never exceed even * the 16-bit length maximum.
label: code-design
47240. (flags<<32) + (line), flags = 0
47241. insert range count
47242. XXX: Set 32-bit result (but must then handle signed and * unsigned results separately).
47243. [... new_glob new_env]
47244. Count actually used (non-NULL, non-DELETED) entries.
47245. no refcount check
47246. Parse statements until a closing token (EOF or '}') is found.
47247. duk_hthread_callstack_unwind() will decrease this on unwind
47248. Note: array dump will include elements beyond * 'length'.

47249. Note: currently the catch binding is handled without a register * binding because we don't support dynamic register bindings (they * must be fixed for an entire function). So, there is no need to * record regbases etc.

47250. Shared helper to provide `toString()` and `valueOf()`. Checks 'this', gets * the primitive value to stack top, and optionally coerces with `ToString()`.

47251. size stored in `duk_heap` unused flag bits

47252. `genbuiltins.py` ensures

47253. `> [... val]`

47254. CR LF again a special case

47255. Note: computes with 'idx' in assertions, so caller beware. * 'idx' is preincremented, i.e. '1' on first call, because it * is more convenient for the caller.

47256. Full, aligned 4-byte reads.

47257. `stack[0] = start * stack[1] = end * stack[2] = ToObject(this) * stack[3] = ToUint32(length) * stack[4] = result array`

47258. * Zero the struct, and start initializing roughly in order

47259. `stack[0...nargs-1] = unshift args (vararg) * stack[nargs] = ToObject(this) * stack[nargs+1] = ToUint32(length)`

47260. `duk_unicode_idp_m_ids_noa[]`

47261. mark-and-sweep is currently running

47262. **comment:** * Error codes: defined in `duktape.h` ** Error codes are used as a shorthand to throw exceptions from inside * the implementation. The appropriate Ecmascript object is constructed * based on the code. Ecmascript code throws objects directly. The error * codes are defined in the public API header because they are also used * by calling code.

label: code-design

47263. `DUK_API_INTERNAL_H_INCLUDED`

47264. `'xffTarget'`

47265. FUNCTION EXPRESSIONS

47266. `0x50-0x5f`

47267. Note: there is no standard formatter for function pointers

47268. default: no change

47269. internal type tag

47270. * Ecmascript bytecode executor.

47271. **comment:** `0 = DUK_HOBJECT_CLASS_UNUSED`

label: code-design

47272. `p` points to digit part ('%xy', `p` points to 'x')

47273. the compressed pointer is zeroed which maps to NULL, so nothing to do.

47274. 'count' is more or less comparable to normal trigger counter update * which happens in memory block (re)allocation.

47275. bottom of current frame

47276. Shared offset/length coercion helper.

47277. E5 Section 15.10.2.11

47278. 'writable'

47279. These functions have trouble working as lightfuncs. * Some of them have specific asserts and some may have * additional properties (e.g. `'require.id'` may be written).

47280. Two lowest bits of opcode are used to distinguish * variants. Bit 0 = inc(0)/dec(1), bit 1 = pre(0)/post(1).

47281. internal spare

47282. value (error) is at stack top

47283. input byte length

47284. gap (if empty string, NULL)

47285. [args [crud] arguments]

47286. lightfunc

47287. higher value conserves memory; also note that linear scan is cache friendly

47288. **comment:** XXX: to be implemented?

label: code-design

47289. **comment:** XXX: convert to fixed buffer?

label: code-design

47290. **comment:** Debug macro to print all parts and dparts (used manually because of debug level).

label: code-design

47291. Assume arrays are dense in the fast path.

47292. cannot happen: strings are not put into refzero list (they don't even have the next/prev pointers)

47293. * Value stack top handling

47294. Set property slot to an empty state. Careful not to invoke * any side effects while using `desc.e_idx` so that it doesn't * get invalidated by a finalizer mutating our object.

47295. `x <= y --> not (x > y) --> not (y < x)`

47296. [key setter this val]

47297. little endian

47298. Typing: use `duk_small_(u)int_fast_t` when decoding small * opcode fields (op, A, B, C) and `duk_(u)int_fast_t` when * decoding larger fields (e.g. BC which is 18 bits). Use * unsigned variant by default, signed when the value is used * in signed arithmetic. Using variable names such as 'a', 'b', * 'c', 'bc', etc makes it easier to spot typing mismatches.

47299. * Create required objects: * - 'arguments' object: array-like, but not an array * - 'map' object: internal object, tied to 'arguments' * - 'mappedNames' object: temporary value used during construction

47300. * Misc util stuff

47301. Contract, either: * - Push string to value stack and return 1 * - Don't push anything and return 0

47302. Node.js return value for failed writes is offset + #bytes * that would have been written.

47303. NaN sign bit is platform specific with unpacked, un-normalized NaNs

47304. **comment:** XXX: union would be more correct

label: code-design

47305. * Allocate an `duk_hobject`. ** The allocated object has no allocation for properties; the caller may * want to force a resize if a desired size is known. ** The allocated object has zero reference count and is not reachable. * The caller MUST make the object reachable and increase its reference * count before invoking any operation that might require memory allocation.

47306. **comment:** XXX: Stack discipline is annoying, could be changed in numconv.

label: code-design

47307. 'stack'

47308. Also check for the `refzero_list`; must not be there unless it is * being finalized when `duk_push_heapptr()` is called. ** Corner case: similar to `finalize_list`.

47309. * Heap compiled function (Ecmascript function) representation. ** There is a single data buffer containing the Ecmascript function's * bytecode, constants, and inner functions.

47310. Behavior for `nargs < 2` is implementation dependent: currently we'll * set a NaN time value (matching V8 behavior) in this case.

47311. ".123"

47312. currently used -> new size

47313. caller is responsible for ensuring this

47314. must be signed

47315. uppercase

47316. **comment:** This variant is needed by `String.prototype.split()`; it needs to perform * global-style matching on a cloned RegExp which is potentially non-global.

label: code-design

47317. **comment:** XXX: optimize this filling behavior later

label: code-design

47318. replaced

47319. **comment:** * Binary bitwise operations use different coercions (ToInt32, ToUInt32) * depending on the operation. We coerce the arguments first using * ToInt32(), and then cast to an 32-bit value if necessary. Note that * such casts must be correct even if there is no native 32-bit type * (e.g., duk_int32_t and duk_uint32_t are 64-bit). ** E5 Sections 11.10, 11.7.1, 11.7.2, 11.7.3

label: code-design

47320. [... regexp input] -> [res_obj]

47321. Mostly API and built-in method related

47322. E5 Sections 15.9.3.1, B.2.4, B.2.5

47323. alloc and init

47324. prediction corrections for prime list (see genhashsizes.py)

47325. Make _Target and _Handler non-configurable and non-writable. * They can still be forcibly changed by C code (both user and * Duktape internal), but not by ECMAScript code.

47326. DUK_USE_REGEXPSUPPORT

47327. the DUK_TOK_RCURLY is eaten by duk_parse_stmts()

47328. * Interrupt counter fixup (for development only).

47329. Note: recursive call

47330. maximum bytecode length in instructions

47331. get const for value at valstack top

47332. value1 -> label number, pseudo-type to indicate a break continuation (for ENDFIN)

47333. line tracking

47334. first call

47335. * All done

47336. undefined is accepted

47337. * Wrap up

47338. this case can no longer occur because refcount is unsigned

47339. automatic semi will be inserted

47340. Special shuffling for INITGET/INITSET, where slot C * identifies a register pair and cannot be shuffled * normally. Use an indirect variant instead.

47341. reachable through activation

47342. [... filename &comp_stk]

47343. **comment:** XXX: faster initialization (direct access or better primitives)

label: code-design

47344. Loop structure ensures that we don't compute t1^2 unnecessarily * on the final round, as that might create a bignum exceeding the * current DUK_BI_MAX_PARTS limit.

47345. **comment:** * Check for invalid backreferences; note that it is NOT an error * to back-reference a capture group which has not yet been introduced * in the pattern (as in \1(foo)/); in fact, the backreference will * always match! It IS an error to back-reference a capture group * which will never be introduced in the pattern. Thus, we can check * for such references only after parsing is complete.

label: code-design

47346. +1 for rounding

47347. note: any entries above the callstack top are garbage and not zeroed

47348. must be, since env was created when catcher was created

47349. side effects (currently none though)

47350. DUK_BUILTIN_PROTOS_H_INCLUDED

47351. * Find a matching label catcher or 'finally' catcher in * the same function. ** A label catcher must always exist and will match unless * a 'finally' captures the break/continue first. It is the * compiler's responsibility to ensure that labels are used * correctly.

47352. shadowed, ignore

47353. 'Uint16Array'

47354. -> [... enum_target res]

47355. * Round-up limit. ** For even values, divides evenly, e.g. 10 -> roundup_limit=5. ** For odd values, rounds up, e.g. 3 -> roundup_limit=2. * If radix is 3, 0/3 -> down, 1/3 -> down, 2/3 -> up.

47356. +1 = one retval

47357. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. ** XXX: this is an overkill for some paths, so optimize this later * (or maybe switch to a stack arguments model entirely).

label: code-design

47358. '%p' and NULL is portable, but special case it * anyway to get a standard NULL marker in logs.

47359. [... obj key val]

47360. **comment:** XXX: for Object.keys() we should check enumerability of key

label: code-design

47361. idx_value may be < 0 (no value), set and get may be NULL

47362. numeric value (character, count)

47363. Note: first character is checked against this. But because * IdentifierPart includes all IdentifierStart characters, and * the first character (if unescaped) has already been checked * in the if condition, this is OK.

47364. assume 'var' has been eaten

47365. empty separator can always match

47366. * Close environment record(s) if they exist. ** Only variable environments are closed. If lex_env != var_env, it * cannot currently contain any register bound declarations. ** Only environments created for a NEWENV function are closed. If an * environment is created for e.g. an eval call, it must not be closed.

47367. coerce to string

47368. * Not found in registers, proceed to the parent record. * Here we need to determine what the parent would be, * if 'env' was not NULL (i.e. same logic as when initializing * the record). ** Note that environment initialization is only deferred when * DUK_HOBJECT_HAS_NEWENV is set, and this only happens for: * - Function code * - Strict eval code ** We only need to check _Lexenv here; _Varenv exists only if it * differs from _Lexenv (and thus _Lexenv will also be present).

47369. no check

47370. **comment:** XXX: optimize to 16 bytes

label: code-design

47371. * Endian conversion

47372. nrets

47373. * Second (and possibly third) pass. ** Generate actual code. In most cases the need for shuffle * registers is detected during pass 1, but in some corner cases * we'll only detect it during pass 2 and a third pass is then * needed (see GH-115).

47374. embed: integer value

47375. 0=base, 1=esc, 2=class, 3=class+esc

47376. XXX: proper flags?

47377. mark-and-sweep flags automatically active (used for critical sections)

47378. heap string indices are autogenerated in duk_strings.h

47379. some assertions

47380. array writes autoincrement length

47381. **comment:** * Special behavior in E5.1. * Note that even though parents == 0, the conflicting property * may be an inherited property (currently our global object's * prototype is Object.prototype). Step 5.e first operates on * the existing property (which is potentially in an ancestor) * and then defines a new property in

the global object (and * never modifies the ancestor). ** Also note that this logic would become even more complicated * if the conflicting property might be a virtual one. Object * prototype has no virtual properties, though. ** XXX: this is now very awkward, rework.

label: code-design

47382. Avoid zero copy with an invalid pointer. If obj->p is NULL, * the 'new_a' pointer will be invalid which is not allowed even * when copy size is zero.

47383. reserved words: keywords

47384. The #ifdef clutter here handles the JX/JC enable/disable * combinations properly.

47385. [... key val]

47386. **comment:** * Check whether or not we have an error handler. ** We must be careful of not triggering an error when looking up the * property. For instance, if the property is a getter, we don't want * to call it, only plain values are allowed. The value, if it exists, * is not checked. If the value is not a function, a TypeError happens * when it is called and that error replaces the original one.

label: code-design

47387. **comment:** XXX: 'res' setup can be moved to function body level; in fact, two 'res' * intermediate values suffice for parsing of each function. Nesting is needed * for nested functions (which may occur inside expressions).

label: code-design

47388. **comment:** This seems to waste a lot of stack frame entries, but good compilers * will compute these as needed below. Some of these initial flags are * also modified in the code below, so they can't all be removed.

label: code-design

47389. [... env callee varmap]

47390. duk_push_(u)int() is guaranteed to support at least (un)signed 32-bit range

47391. [body formals source template]

47392. User callback did not return source code, so module loading * is finished: just update modLoaded with final module.exports * and we're done.

47393. **comment:** XXX: make active breakpoints actual copies instead of pointers?

label: code-design

47394. XXX: this bound function resolution also happens elsewhere, * move into a shared helper.

47395. **comment:** XXX: non-callable . toJSON() doesn't need to cause an abort * but does at the moment, probably not worth fixing.

label: code-design

47396. num args starting from idx_argbase

47397. caller must change active thread, and set thr->resumer to NULL

47398. check that the valstack has space for the final amount and any * intermediate space needed; this is unoptimal but should be safe

47399. unary plus

47400. * Shortcut for accessing global object properties

47401. * slice()

47402. may be changed

47403. step 3.e: replace 'Desc.[[Value]]'

47404. Captures which are undefined have NULL pointers and are returned * as 'undefined'. The same is done when saved[] pointers are insane * (this should, of course, never happen in practice).

47405. Only direct eval inherits strictness from calling code * (E5.1 Section 10.1.1).

47406. Get a (possibly canonicalized) input character from current sp. The input * itself is never modified, and captures always record non-canonicalized * characters even in case-insensitive matching.

47407. **comment:** * Update idx_retval of current activation. ** Although it might seem this is not necessary (bytecode executor * does this for Ecmascript-to-Ecmascript calls; other calls are * handled here), this turns out to be necessary for handling yield * and resume. For them, an Ecmascript-to-native call happens, and * the Ecmascript call's idx_retval must be set for things to work.

label: code-design

47408. [... varmap]

47409. * Type checking

47410. * ===== Duktape authors * ===== Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. ** Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. ** The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang\u00f3f \u00c1llango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e1e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6ttsche (<https://github.com/jaseg>) * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6man * * Doug Sanden * * Josh Engebretson (<https://github.com/JoshEngebretson>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * * Seo Sanghyeon (<https://github.com/sanxiny>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstruchtrup> * * Michael Drake (<https://github.com/tlsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9s Vadla Ravn\u00e1n\u00e9s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.

47411. Grow array part for a new highest array index.

47412. no need to mask idx portion

47413. remove hash entry (no decref)

47414. index

47415. exclusive endpoint

47416. Append bytes from a slice already in the buffer.

47417. this also increases refcount by one

47418. whole, won't clip

47419. [source template]

47420. Cannot overlap.

47421. DUK__L0() cannot be a digit, because the loop doesn't terminate if it is

47422. apparently no hint?

47423. **comment:** DUK_USE_REGEXP_CANON_WORKAROUND

label: code-design

47424. stack[0] = separator (string or regexp) * stack[1] = limit * stack[2] = input string * stack[3] = result array

47425. XXX: clumsy sign extend and masking of 16 topmost bits

47426. Note: no key on stack

47427. [... retval]

47428. DUK_USE_JX

47429. num args

47430. end of _Formals

47431. This would be pointless: we'd return NULL for both lightfuncs and * unexpected types.

47432. 2 when called by debugger

47433. * Top level wrappers

47434. DUK_INTERNAL_H_INCLUDED

47435. Report next pc/line to be executed.

47436. Non-object argument is simply int coerced, matches * V8 behavior (except for "null", which we coerce to * 0 but V8 TypeErrors).
47437. * Build function fixed size 'data' buffer, which contains bytecode, * constants, and inner function references. * * During the building phase 'data' is reachable but incomplete. * Only incref's occur during building (no refzero or GC happens), * so the building process is atomic.
47438. fixed offsets in valstack
47439. * Math built-ins
47440. **comment:** Temporary structure used to pass a stack allocated region through * duk_safe_call().
 label: code-design
47441. check that there is space for at least one new entry
47442. 0xa0...0xaf
47443. eval left argument first
47444. -> [... closure template]
47445. 1 << 52
47446. must match genhashsizes.py
47447. setter and/or getter may be NULL
47448. * Encoding helpers * * Some of the typing is bytecode based, e.g. slice sizes are unsigned 32-bit * even though the buffer operations will use duk_size_t.
47449. current thread
47450. **comment:** Called on a read/write error: NULL all callbacks except the detached * callback so that we never accidentally call them after a read/write * error has been indicated. This is especially important for the transport * I/O callbacks to fulfill guaranteed callback semantics.
 label: code-design
47451. Emit 3 bytes and backtrack if there was padding. There's * always space for the whole 3 bytes so no check needed.
47452. Duktape.Thread.yield() should prevent
47453. encode \xffBar as _Bar if no quotes are applied, this is for * readable internal keys.
47454. LICENSE.txt
47455. [... varname val]
47456. string is 'eval' or 'arguments'
47457. * The default property attributes are correct for all * function valued properties of built-in objects now.
47458. Heap udata, used for allocator functions but also for other heap * level callbacks like pointer compression, etc.
47459. IdentifierStart production with ASCII excluded
47460. nothing to mark
47461. **comment:** XXX: function properties
 label: code-design
47462. prepend regexp to valstack 0 index
47463. act_caller->func may be NULL in some finalization cases, * just treat like we don't know the caller.
47464. convenience
47465. Previous value doesn't need refcount changes because its ownership * is transferred to prev_caller.
47466. * Pop built-ins from stack: they are now INCREF'd and * reachable from the builtins[] array or indirectly * through builtins[].
47467. [obj key value desc]
47468. stmt has non-empty value
47469. -> [... holder name val val ToString(i)]
47470. stage 2: delete configurable entries above target length
47471. When LHS is not register bound, always go through a * temporary. No optimization for top level assignment.
47472. keep key reachable for GC etc; guaranteed not to fail
47473. generate mantissa with a single leading whole number digit
47474. * Complex case conversion helper which decodes a bit-packed conversion * control stream generated by unicode/extract_caseconv.py. The conversion * is very slow because it runs through the conversion data in a linear * fashion to save space (which is why ASCII characters have a special * fast path before arriving here). * * The particular bit counts etc have been determined experimentally to * be small but still sufficient, and must match the Python script * (src/extract_caseconv.py). * * The return value is the case converted codepoint or -1 if the conversion * results in multiple characters (this is useful for regexp Canonicalization * operation). If 'buf' is not NULL, the result codepoint(s) are also * appended to the hbuffer. * * Context and locale specific rules must be checked before consulting * this function.
47475. Stored in duk_heapdr unused flags.
47476. **comment:** * Augmenting errors at their creation site and their throw site. * * When errors are created, traceback data is added by built-in code * and a user error handler (if defined) can process or replace the * error. Similarly, when errors are thrown, a user error handler * (if defined) can process or replace the error. * * Augmentation and other processing at error creation time is nice * because an error is only created once, but it may be thrown and * rethrown multiple times. User error handler registered for processing * an error at its throw site must be careful to handle rethrowing in * a useful manner. * * Error augmentation may throw an internal error (e.g. alloc error). * * Ecmascript allows throwing any values, so all values cannot be * augmented. Currently, the built-in augmentation at error creation * only augments error values which are Error instances (= have the * built-in Error.prototype in their prototype chain) and are also * extensible. User error handlers have no limitations in this respect.
 label: code-design
47477. **comment:** * Emit initializers in sets of maximum max_init_values. * Corner cases such as single value initializers do not have * special handling now. * * Elided elements must not be emitted as 'undefined' values, * because such values would be enumerable (which is incorrect). * Also note that trailing elisions must be reflected in the * length of the final array but cause no elements to be actually * inserted.
 label: code-design
47478. * Initialize built-in objects. Current thread must have a valstack * and initialization errors may longjmp, so a setjmp() catch point * must exist.
47479. terminator
47480. DUK_TOK_EQUALSIGN
47481. * Comparison
47482. DUK_JSON_H_INCLUDED
47483. may have side effects
47484. increases key refcount
47485. on first round, skip
47486. **comment:** * Ecmascript specification algorithm and conversion helpers. * * These helpers encapsulate the primitive Ecmascript operation * semantics, and are used by the bytecode executor and the API * (among other places). Note that some primitives are only * implemented as part of the API and have no "internal" helper. * (This is the case when an internal helper would not really be * useful; e.g. the operation is rare, uses value stack heavily, * etc.) * * The operation arguments depend on what is required to implement * the operation: * * - If an operation is simple and stateless, and has no side * effects, it won't take an duk_hthread argument and its * arguments may be duk_tval pointers (which are safe as long * as no side effects take place). * * - If complex coercions are required (e.g. a "ToNumber" coercion) * or errors may be thrown, the operation takes an duk_hthread * argument. This also implies that the operation may have * arbitrary side effects, invalidating any duk_tval pointers. * * - For operations with potential side effects, arguments can be * taken in several ways: * * a) as duk_tval pointers, which makes sense if the "common case" * can be resolved without side effects (e.g. coercion); the * arguments are pushed to the valstack for coercion if * necessary * * b) as duk_tval values * * c) implicitly on value stack top * * d) as indices to the value stack * * Future work: * * - Argument styles may not be the most sensible in every case now. * * - In-place coercions might be useful for several operations, if * in-place coercion is OK for the bytecode executor and the API.
 label: code-design
47487. Note: may be triggered even if minimal new_size would not reach the limit, * plan limit accordingly (taking DUK_VALSTACK_GROW_STEP into account).
47488. **comment:** A little tricky string approach to provide the flags string. * This depends on the specific flag values in duk_regexp.h, * which needs to be asserted for. In practice this doesn't * produce more compact code than the easier approach in use.
 label: code-design
47489. Note: intentionally use approximations to shave a few instructions: * a_used = old_used (accurate: old_used + 1) * a_size = arr_idx (accurate: arr_idx + 1)
47490. Slot C is used in a non-standard fashion (range of regs), * emitter code has special handling for it (must not set the * "no shuffle" flag).

47491. Note: if atom were to contain e.g. captures, we would need to * re-match the atom to get correct captures. Simply quantifiers * do not allow captures in their atom now, so this is not an issue.

47492. * Criteria for augmenting: * * - augmentation enabled in build (naturally) * - error value internal prototype chain contains the built-in * Error prototype object (i.e. 'val instanceof Error') * * Additional criteria for built-in augmenting: * * - error value is an extensible object

47493. **comment:** XXX: solve into closed form (smaller code)
label: code-design

47494. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * If the final target function cannot be handled by an ecma-to-ecma * call, return to the caller with a return value indicating this case. * The bound chain is resolved and the caller can resume with a plain * function call.

47495. must be signed for loop structure

47496. is_extra

47497. **comment:** * Memory calls: relative to heap, GC interaction, but no error throwing. * * XXX: Currently a mark-and-sweep triggered by memory allocation will run * using the heap->heap_thread. This thread is also used for running * mark-and-sweep finalization; this is not ideal because it breaks the * isolation between multiple global environments. * * Notes: * * - DUK_FREE() is required to ignore NULL and any other possible return * value of a zero-sized alloc/realloc (same as ANSI C free()). * * - There is no DUK_REALLOC_ZEROED because we don't assume to know the * old size. Caller must zero the reallocated memory. * * - DUK_REALLOC_INDIRECT() must be used when a mark-and-sweep triggered * by an allocation failure might invalidate the original 'ptr', thus * causing a realloc retry to use an invalid pointer. Example: we're * reallocating the value stack and a finalizer resizes the same value * stack during mark-and-sweep. The indirect variant requests for the * current location of the pointer being reallocated using a callback * right before every realloc attempt; this circuitous approach is used * to avoid strict aliasing issues in a more straightforward indirect * pointer (void **) approach. Note: the pointer in the storage * location is read but is NOT updated; the caller must do that.
label: code-design

47498. Here 'x' is a Unicode codepoint

47499. [ToUint32(len) array ToUint32(len)] -> [ToUint32(len) array]

47500. Force a resize so that DELETED entries are eliminated. * Another option would be duk__recheck_strtab_size_probe(); * but since that happens on every intern anyway, this whole * check can now be disabled.

47501. * comp_ctx->curr_func is now ready to be converted into an actual * function template.

47502. Potentially truncated, NUL not guaranteed in any case. * The (int_rc < 0) case should not occur in practice.

47503. If not found, resume existence check from Function.prototype. * We can just substitute the value in this case; nothing will * need the original base value (as would be the case with e.g. * setters/getters).

47504. "-123456\0"

47505. Get the original arguments. Note that obj_index may be a relative * index so the stack must have the same top when we use it.

47506. curr value

47507. These are for rate limiting Status notifications and transport peeking.

47508. * Some E5/E5.1 algorithms require that array indices are iterated * in a strictly ascending order. This is the case for e.g. * Array.prototype.forEach() and JSON.stringify() PropertyList * handling. * * To ensure this property for arrays with an array part (and * arbitrary objects too, since e.g. forEach() can be applied * to an array), the caller can request that we sort the keys * here.

47509. DUK_USE_LEXER_SLIDING_WINDOW

47510. DUK_TOK_NULL

47511. * Must be careful to catch errors related to value stack manipulation * and property lookup, not just the call itself.

47512. The file/line arguments are NULL and 0, they're ignored by DUK_ERROR_RAW() * when non-verbose errors are used.

47513. **comment:** 8-byte format could be: * 1111 1111 10xx xxxx [41 bits] * * However, this format would not have a zero bit following the * leading one bits and would not allow 0xFF to be used as an * "invalid utf-8" marker for internal keys. Further, 8-byte * encodings (up to 41 bit code points) are not currently needed.
label: code-design

47514. -> [... obj value]

47515. [key trap_result] -> []

47516. replace in stack

47517. (maybe) truncated

47518. nothing to process

47519. Encode string in small chunks, estimating the maximum expansion so that * there's no need to ensure space while processing the chunk.

47520. * Object.getOwnPropertyDescriptor() (E5 Sections 15.2.3.3, 8.10.4) * * This is an actual function call.

47521. knock back "next temp" to this whenever possible

47522. * Detected a directive

47523. 'for'

47524. complete the millisecond field

47525. 'It is true'

47526. Pre/post inc/dec for register variables, important for loops.

47527. **comment:** Lightfunc name, includes Duktape/C native function pointer, which * can often be used to locate the function from a symbol table. * The name also includes the 16-bit duk_tval flags field because it * includes the magic value. Because a single native function often * provides different functionality depending on the magic value, it * seems reasonably to include it in the name. * * On the other hand, a complicated name increases string table * pressure in low memory environments (but only when function name * is accessed).
label: code-design

47528. **comment:** XXX: if duk_hobject_define_property_internal() was updated * to handle a pre-existing accessor property, this would be * a simple call (like for the ancestor case).
label: code-design

47529. Iteration solution

47530. **comment:** 16 bits would be enough for shared heaphdr flags and duk_hstring * flags. The initial parts of duk_heaphdr_string and duk_heaphdr * must match so changing the flags field size here would be quite * awkward. However, to minimize struct size, we can pack at least * 16 bits of duk_hstring data into the flags field.
label: code-design

47531. * Helper tables

47532. 'this'

47533. **comment:** XXX: could check for e16 == 0 because NULL is guaranteed to * encode to zero.
label: code-design

47534. approximate

47535. Avoid NaN-to-integer coercion as it is compiler specific.

47536. Reset to zero size, keep current limit.

47537. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

47538. callback for indirect reallocs, request for current pointer

47539. [... this]

47540. * Portable 12-byte representation

47541. avoid multiple computations of flags address; bypasses macros

47542. **comment:** Reg/const access macros: these are very footprint and performance sensitive * so modify with care.
label: code-design

47543. must not truncate

47544. prefix matches, lengths matter now

47545. Careful with reachability here: don't pop 'obj' before pushing * proxy target.
47546. * Replace valstack top with case converted version.
47547. Shuffle decision not changed.
47548. * Stack index validation/normalization and getting a stack duk_tval ptr. * * These are called by many API entrypoints so the implementations must be * fast and "inlined". * * There's some repetition because of this; keep the functions in sync.
47549. numeric value of token
47550. [... constructor arg1 ... argN] -> [retval]
47551. * Rebuild the hash part always from scratch (guaranteed to finish). * * Any resize of hash part requires rehashing. In addition, by rehashing * get rid of any elements marked deleted (DUK_HASH_DELETED) which is critical * to ensuring the hash part never fills up.
47552. temps
47553. 'Object'
47554. not strictly necessary
47555. IsDataDescriptor(desc) == true
47556. POSTFIX EXPRESSION
47557. If argument count is 1 and first argument is a buffer, write the buffer * as raw data into the file without a newline; this allows exact control * over stdout/stderr without an additional entrypoint (useful for now). * * Otherwise current print/alert semantics are to ToString() coerce * arguments, join them with a single space, and append a newline.
47558. * Exposed regexp compilation primitive. * * Sets up a regexp compilation context, and calls duk_parse_disjunction() to do the * actual parsing. Handles generation of the compiled regexp header and the * "boilerplate" capture of the matching substring (save 0 and 1). Also does some * global level regexp checks after recursive compilation has finished. * * An escaped version of the regexp source, suitable for use as a RegExp instance * 'source' property (see E5 Section 15.10.3), is also left on the stack. * * Input stack: [pattern flags] * Output stack: [bytecode escaped_source] (both as strings)
47559. 'Error'
47560. * Flags for __FILE__ / __LINE__ registered into tracedata
47561. Count actually used entry part entries (non-NULL keys).
47562. DUK_TOK_ARSHIFT
47563. **comment:** XXX: possibly incorrect handling of empty expression
 label: code-design
47564. used entries + approx 100% -> reset load to 50%
47565. '\xffRegbase'
47566. is_decl
47567. but don't allow leading plus
47568. **comment:** * String cache should ideally be at duk_hthread level, but that would * cause string finalization to slow down relative to the number of * threads; string finalization must check the string cache for "weak" * references to the string being finalized to avoid dead pointers. * * Thus, string caches are now at the heap level now.
 label: code-design
47569. For manual debugging: instruction count based on executor and * interrupt counter book-keeping. Inspect debug logs to see how * they match up.
47570. [...] holder name val enum obj_key val obj_key]
47571. lj.value1 is already set
47572. number of arguments allocated to regs
47573. Optimized for size.
47574. * Default fatal error handler
47575. named "arithmetic" because result is signed
47576. Default implicit return value.
47577. lookup name from current activation record's functions' registers
47578. custom; implies DUK_HOBJECT_IS_BUFFEROBJECT
47579. max # of key-value pairs initialized in one MPUTOBJ set
47580. unicode code points, window[0] is always next
47581. Sync so that augmentation sees up-to-date activations, NULL * thr->ptr_curr_pc so that it's not used if side effects occur * in augmentation or longjmp handling.
47582. thread not currently running
47583. Refzero head is still the same. This is the case even if finalizer * inserted more refzero objects; they are inserted to the tail.
47584. [...] key arg1 ... argN]
47585. * GETIDREF: a GetIdentifierReference-like helper. * * Provides a parent traversing lookup and a single level lookup * (for HasBinding). * * Instead of returning the value, returns a bunch of values allowing * the caller to read, write, or delete the binding. Value pointers * are duk_tval pointers which can be mutated directly as long as * refcounts are properly updated. Note that any operation which may * reallocate valstacks or compact objects may invalidate the returned * duk_tval (but not object) pointers, so caller must be very careful. * * If starting environment record 'env' is given, 'act' is ignored. * However, if 'env' is NULL, the caller may identify, in 'act', an * activation which hasn't had its declarative environment initialized * yet. The activation registers are then looked up, and its parent * traversed normally. * * The 'out' structure values are only valid if the function returns * success (non-zero).
47586. Ignore the normalize/validate helper outputs on the value stack, * they're popped automatically.
47587. **comment:** A union is used here as a portable struct size / alignment trick: * by adding a 32-bit or a 64-bit (unused) union member, the size of * the struct is effectively forced to be a multiple of 4 or 8 bytes * (respectively) without increasing the size of the struct unless * necessary.
 label: code-design
47588. Chop extra retrvals away / extend with undefined.
47589. MPUTARR emitted by outer loop
47590. **comment:** XXX: fastint?
 label: code-design
47591. compact the exports table
47592. Code is not accepted before the first case/default clause
47593. Process one debug message. Automatically restore value stack top to its * entry value, so that individual message handlers don't need exact value * stack handling which is convenient.
47594. -> [...] key val replacer holder key val]
47595. Because new_size != 0, if condition doesn't need to be * (new_valstack != NULL || new_size == 0).
47596. No debugger support.
47597. '<'
47598. treat like property not found
47599. Bound functions don't have all properties so we'd either need to * lookup the non-bound target function or reject bound functions. * For now, bound functions are rejected.
47600. Carry is detected based on wrapping which relies on exact 32-bit * types.
47601. eat (first) input slash
47602. value resides in a register or constant
47603. **comment:** "get" and "set" are tokens but NOT ReservedWords. They are currently * parsed and identifiers and these defines are actually now unused.
 label: code-design
47604. Overflow, relevant mainly when listlen is 16 bits.
47605. [...] val] -> [...] enum]
47606. E5.1 standard behavior when deleteCount is not given would be * to treat it just like if 'undefined' was given, which coerces * ultimately to 0. Real world behavior is to splice to the end * of array, see test-bi-array-proto-splice-no-delcount.js.
47607. case 0: nop
47608. **comment:** XXX: the duk_hobject_enum.c stack APIs should be reworked
 label: code-design

47609. Error: try coercing error to string once.
47610. for resizing of array part, use slow path
47611. $y == -\infty$
47612. Initialization and finalization (compaction), converting to other types.
47613. must match exactly the number of internal properties inserted to enumerator
47614. required
47615. + spare
47616. All element accessors are host endian now (driven by TypedArray spec).
47617. Push an arbitrary duk_tval to the stack, coerce it to string, and return * both a duk_hstring pointer and an array index (or DUK__NO_ARRAY_INDEX).
47618. must "goto restart_execution", e.g. breakpoints changed
47619. Avoid side effects that might disturb curr.e_idx until * we're done editing the slot.
47620. 23: getUTCMilliseconds
47621. no need to normalize
47622. * Windows Date providers ** Platform specific links: ** - [http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms725473(v=vs.85).aspx)
47623. **comment:** XXX: Change 'anon' handling here too, to use empty string for anonymous functions?
 label: code-design
47624. 'this binding' is just under bottom
47625. Default exports table
47626. **comment:** XXX: macro smaller than call?
 label: code-design
47627. XXX: assert? compiler is responsible for this never happening
47628. pc of label statement: * pc+1: break jump site * pc+2: continue jump site
47629. 'require'
47630. * Assertion helpers
47631. multiple codepoint conversion or non-ASCII mapped to ASCII * --> leave as is.
47632. Computation must not wrap, only srclen + 3 is at risk of * wrapping because after that the number gets smaller. * This limit works for 32-bit size_t: * 0x100000000
 - 3 - 1 = 4294967292
47633. 19: getUTCMMinutes
47634. Inspect a property using a virtual index into a conceptual property list * consisting of (1) all array part items from [0,a_size[(even when above * .length) and (2) all entry part items from [0,e_next[. Unused slots are * indicated using dvalue 'unused'.
47635. duk_get_min_grow_e() is always >= 1
47636. * Reachable object, keep
47637. 'eval'
47638. Array part may be larger than 'length'; if so, iterate * only up to array 'length'.
47639. regCatch+0 and regCatch+1 are reserved for TRYCATCH
47640. resolved id: require(id) must return this same module
47641. We rely on a few object flag / class number relationships here, * assert for them.
47642. default case control flow patchup; note that if pc_prevcase < 0 * (i.e. no case clauses), control enters default case automatically.
47643. DUK_HEAP_H_INCLUDED
47644. note: original, uncoerced base
47645. If len <= 1, middle will be 0 and for-loop bails out * immediately (0 < 0 -> false).
47646. DUK_OP_NONE marks a 'plain' assignment
47647. resume execution from pc_base or pc_base+1 (points to 'func' bytecode, stable pointer)
47648. Stack size decreases.
47649. heap->strs: not worth dumping
47650. -> [... closure template env funcname closure]
47651. step 12
47652. Section 7.8.3 (note): NumericLiteral must be followed by something other than * IdentifierStart or DecimalDigit.
47653. **comment:** XXX: duk_ssize_t
 label: code-design
47654. DUK_ERR_UNCAUGHT_ERROR: no macros needed
47655. -> [ToObject(this) item1 ... itemN arr]
47656. C call site gets blamed next, unless flagged not to do so. * XXX: file/line is disabled in minimal builds, so disable this * too when appropriate.
47657. * Debug dump type sizes
47658. needed for line number tracking (becomes prev_token.start_line)
47659. don't coerce yet to a plain value (variant 3 needs special handling)
47660. **comment:** XXX: opcode specific assertions on when consts are allowed
 label: code-design
47661. Read value pushed on stack.
47662. * When resizing the valstack, a mark-and-sweep may be triggered for * the allocation of the new valstack. If the mark-and-sweep needs * to use our thread for something, it may cause *the same valstack* to be resized recursively. This happens e.g. when mark-and-sweep * finalizers are called. This is taken into account carefully in * duk_resize_valstack(). ** 'new_size' is known to be <= valstack_max, which ensures that * size_t and pointer arithmetic won't wrap in duk_resize_valstack().
47663. 'null' enumerator case -> behave as with an empty enumerator
47664. Object literal set/get functions have a name (property * name) but must not have a lexical name binding, see * test-bug-getset-func-name.js.
47665. Current compiler state (if any), used for augmenting SyntaxErrors.
47666. push to stack
47667. Note: if DecimalLiteral starts with a '0', it can only be * followed by a period or an exponent indicator which starts * with 'e' or 'E'. Hence the if-check above ensures that * OctalIntegerLiteral is the only valid NumericLiteral * alternative at this point (even if y is, say, '9').
47668. 'catch'
47669. * Special cases like NaN and +/- Infinity are handled explicitly * because a plain C coercion from double to int handles these cases * in undesirable ways. For instance, NaN may coerce to INT_MIN * (not zero), and INT_MAX + 1 may coerce to INT_MIN (not INT_MAX). ** This double-to-int coercion differs from ToInteger() because it * has a finite range (ToInteger() allows e.g. +/- Infinity). It * also differs fromToInt32() because the INT_MIN/INT_MAX clamping * depends on the size of the int type on the platform. In particular, * on platforms with a 64-bit int type, the full range is allowed.
47670. element size shift: * 0 = u8/i8 * 1 = u16/i16 * 2 = u32/i32/float * 3 = double
47671. curr_pc synced back above
47672. hash size approx. 1.25 times entries size
47673. Write in little endian
47674. source starts after dest ends
47675. **comment:** * Debugging macro calls.
 label: code-design
47676. DUK_USE_VERBOSE_ERRORS
47677. 0xxx xxxx -> fast path
47678. * Ecmascript modulus (%) does not match IEEE 754 "remainder" * operation (implemented by remainder() in C99) but does seem * to match ANSI C fmod(). ** Compare E5 Section 11.5.3 and "man fmod".
47679. unsigned shift
47680. Not a very good provider: only full seconds are available.
47681. 110x xxxx 10xx xxxx [11 bits]

47682. Note: errors are augmented when they are created, not * when they are thrown. So, don't augment here, it would * break re-throwing for instance.

47683. **comment:** Not necessary to unwind catchstack: return handling will * do it. The finally flag of 'cat' is no longer set. The * catch flag may be set, but it's not checked by return handling.
label: code-design

47684. ensure value is 1 or 0 (not other non-zero)

47685. Accept plain buffer values like array initializers * (new in Duktape 1.4.0).

47686. Non-zero refcounts should not happen for unreachable strings, * because we refcount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).

47687. Indicate function type in the function body using a dummy * directive.

47688. 'ObjEnv'

47689. lj.value1 already set

47690. msec -> 100ns units since jan 1, 1970

47691. **comment:** * Note: must use indirect variant of DUK_REALLOC() because underlying * pointer may be changed by mark-and-sweep.
label: code-design

47692. DUK_TOK_SWITCH

47693. class Object, extensible

47694. idx_end

47695. _Source

47696. 1 1 0 <8 bits>

47697. **comment:** two casts to avoid gcc warning: "warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]"
label: code-design

47698. minimum new length is highest_arr_idx + 1

47699. DUK_TOK_RBRACKET

47700. property is configurable and

47701. * Matching complete, create result array or return a 'null'. Update lastIndex * if necessary. See E5 Section 15.10.6.2. ** Because lastIndex is a character (not byte) offset, we need the character * length of the match which we conveniently get as a side effect of interning * the matching substring (0th index of result array). ** saved[0] start pointer (~ byte offset) of current match * saved[1] end pointer (~ byte offset) of current match (exclusive) * char_offset start character offset of current match (-> .index of result) * char_end_offset end character offset (computed below)

47702. **comment:** XXX: Refactor key coercion so that it's only called once. It can't * be trivially lifted here because the object must be type checked * first.
label: code-design

47703. Caller must ensure 'tv' is indeed a double and not a fastint!

47704. Caller has already eaten the first character ('|') which we don't need.

47705. * String scanning helpers ** All bytes other than UTF-8 continuation bytes ([0x80,0xbff]) are * considered to contribute a character. This must match how string * character length is computed.

47706. [... func this | arg1 ... argN] ('this' must precede new bottom)

47707. structs from duk_forwdecl.h

47708. assumed to also be PC of "LABEL"

47709. Check that prev/next links are consistent: if e.g. h->prev is != NULL, * h->prev->next should point back to h.

47710. '{"_undef":true}'

47711. DUK_TOK_ADD_EQ

47712. roughly 1.0 kB -> but rounds up to DUK_VALSTACK_GROW_STEP in practice

47713. remove the original 'template' atom

47714. The ANSI C pow() semantics differ from Ecmascript. ** E.g. when x==1 and y is +/- infinite, the Ecmascript required * result is NaN, while at least Linux pow() returns 1.

47715. * Heap native function representation.

47716. Note: DUK_HEAP_HAS_REFZERO_FREE_RUNNING(heap) may be true; a refcount * finalizer may trigger a mark-and-sweep.

47717. 'Invalid Date'

47718. dynamic

47719. **comment:** * Node.js Buffer.concat()
label: code-design

47720. **comment:** XXX: substring in-place at idx_place?
label: code-design

47721. * Object's finalizer was executed on last round, and * object has been happily rescued.

47722. embed: 0 or 1 (false or true)

47723. Labels can be used for iteration statements but also for other statements, * in particular a label can be used for a block statement. All cases of a * named label accept a 'break' so that flag is set here. Iteration statements * also allow 'continue', so that flag is updated when we figure out the * statement type.

47724. string 2 of token (borrowed, stored to ctx->slot2_idx)

47725. UNARY EXPRESSIONS

47726. * Not found (even in global object). ** In non-strict mode this is a silent SUCCESS (!), see E5 Section 11.4.1, * step 3.b. In strict mode this case is a compile time SyntaxError so * we should not come here.

47727. should actually never happen, but check anyway

47728. input offset for window leading edge (not window[0])

47729. * Init object properties ** Properties should be added in decreasing order of access frequency. * (Not very critical for function templates.)

47730. [... fallback constructor fallback(this) arg1 ... argN]; * Note: idx_cons points to first 'fallback', not 'constructor'.

47731. **comment:** * Canonicalize a range, generating result ranges as necessary. * Needs to exhaustively scan the entire range (at most 65536 * code points). If 'direct' is set, caller (lexer) has ensured * that the range is already canonicalization compatible (this * is used to avoid unnecessary canonicalization of built-in * ranges like \W, which are not affected by canonicalization). ** NOTE: here is one place where we don't want to support chars * outside the BMP, because the exhaustive search would be * massively larger.
label: code-design

47732. continuation byte

47733. Note: the realloc may have triggered a mark-and-sweep which may * have resized our valstack internally. However, the mark-and-sweep * MUST NOT leave the stack bottom/top in a different state. Particular * assumptions and facts: * * - The thr->valstack pointer may be different after realloc, * and the offset between thr->valstack_end <-> thr->valstack * may have changed. * - The offset between thr->valstack_bottom <-> thr->valstack * and thr->valstack_top <-> thr->valstack MUST NOT have changed, * because mark-and-sweep must adhere to a strict stack policy. * In other words, logical bottom and top MUST NOT have changed. * - All values above the top are unreachable but are initialized * to UNDEFINED, up to the post-realloc valstack_end. * - 'old_end_offset' must be computed after realloc to be correct.

47734. Note: hstring is in heap but has refcount zero and is not strongly reachable. * Caller should increase refcount and make the hstring reachable before any * operations which require allocation (and possible gc).

47735. **comment:** * "name" is a non-standard property found in at least V8, Rhino, smjs. * For Rhino and smjs it is non-writable, non-enumerable, and non-configurable; * for V8 it is writable, non-enumerable, non-configurable. It is also defined * for an anonymous function expression in which case the value is an empty string. * We could also leave name 'undefined' for anonymous functions but that would * differ from behavior of other engines, so use an empty string. ** XXX: make optional? costs something per function.
label: code-design

47736. **comment:** * duk_hbuffer operations such as resizing and inserting/appending data to * a dynamic buffer. ** Append operations append to the end of the buffer and they are relatively * efficient: the buffer is grown with a "spare" part relative to the buffer * size to minimize reallocations. Insert operations need to move existing * data forward in the buffer with memmove() and are not very efficient. * They are used e.g. by the regexp compiler to "backpatch" regexp bytecode.
label: code-design

47737. steps 11.c.ii.1 - 11.c.ii.4, but our internal book-keeping * differs from the reference model

47738. ASCII fast path
47739. Most input bytes go through here.
47740. -> [... holder name val]
47741. Assume that year, month, day are all coerced to whole numbers. * They may also be NaN or infinity, in which case this function * must return NaN or infinity to ensure time value becomes NaN. * If 'day' is NaN, the final return will end up returning a NaN, * so it doesn't need to be checked here.
47742. useful for debugging
47743. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()! * * A conservative approach would be to use duk__ivalue_totempconst() * for 'left'. However, allowing a reg-bound variable seems safe here * and is nice because "foo.bar" is a common expression. If the ivalue * is used in an expression a GETPROP will occur before any changes to * the base value can occur. If the ivalue is used as an assignment * LHS, the assignment code will ensure the base value is safe from * RHS mutation.
47744. We should never exit the loop above.
47745. may have FINALIZED
47746. LAYOUT 3
47747. **comment:** unused
 label: code-design
47748. these computations are guaranteed to be exact for the valid * E5 time value range, assuming milliseconds without fractions.
47749. 'Int32Array'
47750. an error handler (user callback to augment/replace error) is running
47751. without refcounts
47752. arbitrary marker, outside valid exp range
47753. extra -1 for buffer
47754. For invalid characters the value -1 gets extended to * at least 16 bits. If either nybble is invalid, the * resulting 't' will be < 0.
47755. required to guarantee success of rehashing, * intentionally use unadjusted new_e_size
47756. XXX: A fast path for usual integers would be useful when * fastint support is not enabled.
47757. -> [... ToObject(this) ToUint32(length) final_len final_len]
47758. [...val] -> [...]
47759. allow empty expression
47760. [... closure template name]
47761. # total registers target function wants on entry (< 0 => "as is")
47762. **comment:** unnecessary shift for last byte
 label: code-design
47763. varenv remains reachable through 'obj'
47764. Special coercion for Uint8ClampedArray.
47765. valstack slot for temp buffer
47766. * Node.js Buffer: constructor
47767. this optimization is important to handle negative literals * (which are not directly provided by the lexical grammar)
47768. Nothing worked -> not equal.
47769. E5 Section 11.13.1 (and others for other assignments), step 4.
47770. Note: unshift() may operate on indices above unsigned 32-bit range * and the final length may be >= 2**32. However, we restrict the * final result to 32-bit range for practicality.
47771. String and buffer enumeration behavior is identical now, * so use shared handler.
47772. 37x32 = 1184 bits
47773. heap->dbg_detached_cb: keep
47774. This check used to be for (t < 0) but on some platforms * time_t is unsigned and apparently the proper way to detect * an mktime() error return is the cast above. See e.g.: * <http://pubs.opengroup.org/onlinepubs/009695299/functions/mktime.html>
47775. **comment:** Fast variants, inline refcount operations except for refzero handling. * Can be used explicitly when speed is always more important than size. * For a good compiler and a single file build, these are basically the * same as a forced inline.
 label: code-design
47776. [... key] -> [...]
47777. * Use current token to decide whether a RegExp can follow. * * We can use either 't' or 't_nores'; the latter would not * recognize keywords. Some keywords can be followed by a * RegExp (e.g. "return"), so using 't' is better. This is * not trivial, see doc/compiler.rst.
47778. Khronos/ES6 requires zeroing even when DUK_USE_ZERO_BUFFER_DATA * is not set.
47779. * Refcount memory freeing loop. * * Frees objects in the refzero_pending list until the list becomes * empty. When an object is freed, its references get decref'd and * may cause further objects to be queued for freeing. * * This could be expanded to allow incremental freeing: just bail out * early and resume at a future alloc/decref/refzero.
47780. Note: 'fast' breaks will jump to pc_label_site + 1, which will * then jump here. The double jump will be eliminated by a * peephole pass, resulting in an optimal jump here. The label * site jumps will remain in bytecode and will waste code size.
47781. **comment:** * Prototype walking starting from 'env'. * * ('act' is not needed anywhere here.)
 label: code-design
47782. mark range 'direct', bypass canonicalization (see Wiki)
47783. Do debugger forwarding before raw() because the raw() function * doesn't get the log level right now.
47784. estimate year upwards (towards positive infinity), then back down; * two iterations should be enough
47785. return module.exports
47786. * Parsing an expression starting with 'new' is tricky because * there are multiple possible productions deriving from * LeftHandSideExpression which begin with 'new'. * * We currently resort to one-token lookahead to distinguish the * cases. Hopefully this is correct. The binding power must be * such that parsing ends at an LPAREN (CallExpression) but not at * a PERIOD or LBRACKET (MemberExpression). * * See doc/compiler.rst for discussion on the parsing approach, * and testcases/test-dev-new.js for a bunch of documented tests.
47787. exptrop is the top level variant which resets nud/led counts
47788. control props
47789. push value to stack
47790. * shift()
47791. k+argCount-1; note that may be above 32-bit range
47792. * Assert macro: failure causes panic.
47793. **comment:** Similar workaround for INFINITY.
 label: code-design
47794. * Fallback marking handler if recursion limit is reached. * * Iterates 'temproots' until recursion limit is no longer hit. Note * that temproots may reside either in heap allocated list or the * refzero work list. This is a slow scan, but guarantees that we * finish with a bounded C stack. * * Note that nodes may have been marked as temproots before this * scan begun, OR they may have been marked during the scan (as * we process nodes recursively also during the scan). This is * intended behavior.
47795. **comment:** If number would need zero padding (for whole number part), use * exponential format instead. E.g. if input number is 12300, 3 * digits are generated ("123"), output "1.23e+4" instead of "12300". * Used for toPrecision().
 label: code-design
47796. number
47797. Must be element size multiple from * start offset to end of buffer.
47798. empty ("") is allowed in some formats (e.g. Number()), as zero
47799. slow path is shared
47800. * If matching with a regexp: * - non-global RegExp: lastIndex not touched on a match, zeroed * on a non-match * - global RegExp: on match, lastIndex will be updated by regexp * executor to point to next char after the matching part (so that * characters in the matching part are not matched again) * * If matching with a

string: * - always non-global match, find first occurrence ** We need: * - The character offset of start-of-match for the replacer function * - The byte offsets for start-of-match and end-of-match to implement * the replacement values \$\$, \$` and \$', and to copy non-matching * input string portions (including header and trailer) verbatim. ** NOTE: the E5.1 specification is a bit vague how the RegExp should * behave in the replacement process; e.g. is matching done first for * all matches (in the global RegExp case) before any replacer calls * are made? See: test-bi-string-proto-replace.js for discussion.

47801. -> [... func]

47802. * Various defines and file specific helper macros

47803. no net exponent

47804. **comment:** integer, but may be +/- Infinite, +/- zero (not NaN, though)

label: code-design

47805. 11 bits

47806. **comment:** Lookup active label information. Break/continue distinction is necessary to handle switch * statement related labels correctly: a switch will only catch a 'break', not a 'continue'. ** An explicit label cannot appear multiple times in the active set, but empty labels (unlabelled * iteration and switch statements) can. A break will match the closest unlabelled or labelled * statement. A continue will match the closest unlabelled or labelled iteration statement. It is * a syntax error if a continue matches a labelled switch statement; because an explicit label cannot * be duplicated, the continue cannot match any valid label outside the switch. ** A side effect of these rules is that a LABEL statement related to a switch should never actually * catch a continue abrupt completion at run-time. Hence an INVALID opcode can be placed in the * continue slot of the switch's LABEL statement.

label: code-design

47807. get before side effects

47808. DUK_HSTRING_HINCLUDED

47809. avoid empty label at the end of a compound statement

47810. no specific action, resume normal execution

47811. for manual torture testing: tight allocation, useful with valgrind

47812. XXX: skip count_free w/o debug?

47813. everything except object stay as is

47814. identifiers and environment handling

47815. back to fast loop

47816. DUK_USE_JX || DUK_USE_JC

47817. Run fake finalizer. Avoid creating new refzero queue entries * so that we are not forced into a forever loop.

47818. In some cases it may be that lo > hi, or hi < 0; these * degenerate cases happen e.g. for empty arrays, and in * recursion leaves.

47819. is resume, not a tail call

47820. * Basic stack manipulation: swap, dup, insert, replace, etc

47821. **comment:** Note: nargs (and nregs) may be negative for a native, * function, which indicates that the function wants the * input stack "as is" (i.e. handles "vararg" arguments).

label: code-design

47822. * Indirect magic value lookup for Date methods. ** Date methods don't put their control flags into the function magic value * because they wouldn't fit into a LIGHTFUNC's magic field. Instead, the * magic value is set to an index pointing to the array of control flags * below. ** This must be kept in strict sync with genbuiltins.py!

47823. strings may have inner refs (extdata) in some cases

47824. class masks

47825. keep label catcher

47826. state updated, restart bytecode execution

47827. Flags for duk_js_equals_helper().

47828. [... func this <some bound args> arg1 ... argN _Args]

47829. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined fresh_require exports module result(ignored)]

47830. DUK_USE_TAILCALL

47831. **comment:** should never be called

label: code-design

47832. denormal

47833. -> loop body

47834. note: this updates refcounts

47835. avoid calling at end of input, will DUK_ERROR (above check suffices to avoid this)

47836. t1 <- (+ r m+)

47837. denormal or zero

47838. mark-and-sweep control

47839. A 'return' statement is only allowed inside an actual function body, * not as part of eval or global code.

47840. tailcall cannot be flagged to resume calls, and a * previous frame must exist

47841. DUK_TOK_LET

47842. Remove a slice from inside buffer.

47843. valstack slot for 2nd token value

47844. one-based -> zero-based

47845. Set .buffer

47846. terminate

47847. max size: radix=2 + sign

47848. remaining actual steps are carried out if standard DefineOwnProperty succeeds

47849. **comment:** The prev/next pointers of the removed duk_heapdr are left as garbage. * It's up to the caller to ensure they're written before inserting the * object back.
label: code-design

47850. {'_func':true'}

47851. **comment:** XXX: support unary arithmetic ivalue (useful?)

label: code-design

47852. E5 Section 15.5.5.2

47853. True if slice is full, i.e. offset is zero and length covers the entire * buffer. This status may change independently of the duk_hbufferobject if * the underlying buffer is dynamic and changes without the hbufferobject * being changed.

47854. avoid calling write callback in detach1()

47855. http://en.wikipedia.org/wiki/Replacement_character#Replacement_character

47856. -> [Object res]

47857. * Property attribute defaults are defined in E5 Section 15 (first * few pages); there is a default for all properties and a special * default for 'length' properties.
Variation from the defaults is * signaled using a single flag bit in the bitstream.

47858. IdentifierStart production with Letter and ASCII excluded

47859. **comment:** * DecimalLiteral, HexIntegerLiteral, OctallIntegerLiteral * "pre-parsing", followed by an actual, accurate parser step. ** Note: the leading sign character ('+' or '-') is -not- part of * the production in E5 grammar, and that a DecimalLiteral * starting with a '0' must be followed by a non-digit. Leading * zeroes are syntax errors and must be checked for. ** XXX: the two step parsing process is quite awkward, it would * be more straightforward to allow numconv to parse the longest * valid prefix (it already does that, it only needs to indicate * where the input ended). However, the lexer decodes characters * using a lookup window, so this is not a trivial change.

label: code-design

47860. * The escapes are same as outside a character class, except that \b has a * different meaning, and \B and backreferences are prohibited (see E5 * Section 15.10.2.19). However, it's difficult to share code because we * handle e.g. "\n" very differently: here we generate a single character * range for it.

47861. **comment:** Ensure there are no stale active breakpoint pointers. * Breakpoint list is currently kept - we could empty it * here but we'd need to handle refcounts correctly, and * we'd need a 'thr' reference for that. ** XXX: clear breakpoint on either attach or detach?

label: code-design

47862. end switch

47863. DUK_TOK_LE

47864. XXX: Currently no inspection of threads, e.g. value stack, call * stack, catch stack, etc.

47865. [... v1(func) v2(pc+flags)]

47866. Matches both +0 and -0 on purpose.

47867. Note: ToPrimitive(object,hint) == [[DefaultValue]](object,hint), * so use [[DefaultValue]] directly.

47868. The code for writing reg_temps + 0 to the left hand side has already * been emitted.

47869. start line of token (first char)

47870. 10: getMonth

47871. If caller is global/eval code, 'caller' should be set to * 'null'. * * XXX: there is no exotic flag to infer this correctly now. * The NEWENV flag is used now which works as intended for * everything (global code, non-strict eval code, and functions) * except strict eval code. Bound functions are never an issue * because 'func' has been resolved to a non-bound function.

47872. x <- y

47873. eat 'var'

47874. cannot have >4G captures

47875. **comment:** not caught by current thread, thread terminates (yield error to resumer); * note that this may cause a cascade if the resumer terminates with an uncaught * exception etc (this is OK, but needs careful testing)

label: code-design

47876. normalize NaN which may not match our canonical internal NaN

47877. Recompute argument count: bound function handling may have shifted.

47878. [... key val] -> [...]

47879. minval

47880. if len_x == len_y == 0, buf_x and/or buf_y may * be NULL, but that's OK.

47881. DUK_TOK_ARSHIFT_EQ

47882. Slightly smaller code without explicit flag, but explicit flag * is very useful when 'cлен' is dropped.

47883. Here again we parse bytes, and non-ASCII UTF-8 will cause end of * parsing (which is correct except if there are non-shortest encodings). * There is also no need to check explicitly for end of input buffer as * the input is NUL padded and NUL will exit the parsing loop. * * Because no unescaping takes place, we can just scan to the end of the * plain string and intern from the input buffer.

47884. **comment:** XXX: merge this with duk_js_call.c, as this function implements * core semantics (or perhaps merge the two files altogether).

label: code-design

47885. * New length lower than old length => delete elements, then * update length. * * Note: even though a bunch of elements have been deleted, the 'desc' is * still valid as properties haven't been resized (and entries compacted).

47886. * Program code (global and eval code) has an implicit return value * from the last statement value (e.g. eval("1; 2+3;") returns 3). * This is not the case with functions. If implicit statement return * value is requested, all statements are coerced to a register * allocated here, and used in the implicit return statement below.

47887. * Pass 1

47888. Use unsigned arithmetic to optimize comparison.

47889. type and control flags, label number

47890. DUK_TOK_RSHIFT_EQ

47891. probe steps (see genhashsizes.py), currently assumed to be 32 entries long * (DUK_UTIL_GET_HASH_STEP macro).

47892. [... enum_target res trap_result]

47893. 1111 1110 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [36 bits]

47894. * Conversions and coercions * * The conversion/coercions are in-place operations on the value stack. * Some operations are implemented here directly, while others call a * helper in duk_js_ops.c after validating arguments.

47895. **comment:** Select a thread for mark-and-sweep use. * * XXX: This needs to change later.

label: code-design

47896. Automatic filename: 'eval' or 'input'.

47897. cumulative opcode execution count (overflows are OK)

47898. Fast path for fastints

47899. **comment:** XXX: any chance of merging these three similar but still slightly * different algorithms so that footprint would be reduced?

label: code-design

47900. comp_ctx->lex has been pre-initialized by caller: it has been * zeroed and input/input_length has been set.

47901. 'input'

47902. countdown state

47903. switch

47904. * Return to bytecode executor, which will resume execution from * the topmost activation.

47905. Shared exit handling for object/array serialization.

47906. input string scan

47907. as elements

47908. * Rewind lexer. * * duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp * literal" mode with current strictness. * * curr_token line number info should be initialized for pass 2 before * generating prologue, to ensure prologue bytecode gets nice line numbers.

47909. Shared helper for Object.getOwnPropertyNames() and Object.keys(). * Magic: 0=GetPropertyNames, 1=Object.keys.

47910. no need to unwind

47911. When looking for .fileName/.lineNumber, blame first * function which has a .fileName.

47912. **comment:** XXX: if attempt to push beyond allocated valstack, this double fault * handling fails miserably. We should really write the double error * directly to thr->heap->lj.value1 and avoid valstack use entirely.

label: code-design

47913. Strict functions don't get their 'caller' updated.

47914. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module mod_func]

47915. currently nop

47916. [... closure template]

47917. * A tiny random number generator. * * Currently used for Math.random(). * * <http://www.woodmann.com/forum/archive/index.php/t-3100.html>

47918. 'func' on stack (borrowed reference)

47919. **comment:** * "prototype" is, by default, a fresh object with the "constructor" * property. * * Note that this creates a circular reference for every function * instance (closure) which prevents refcount-based collection of * function instances. * * XXX: Try to avoid creating the default prototype object, because * many functions are not used as constructors and the default * prototype is unnecessary. Perhaps it could be created on-demand * when it is first accessed?

label: code-design

47920. -> [... val this]

47921. **comment:** code emission temporary

label: code-design

47922. * TypedArray.prototype.set() * * TypedArray set() is pretty interesting to implement because: * * - The source argument may be a plain array or a typedarray. If the * source is a TypedArray, values are decoded and re-encoded into the * target (not as a plain byte copy). This may happen even when the * element byte size is the same, e.g. integer values may be re-encoded * into floats. * * - Source and target may refer to the same underlying buffer, so that * the set() operation may overlap. The specification requires that this * must work as if a copy was made before the operation. Note that this * is NOT a simple memmove() situation because the source and target * byte sizes may be different -- e.g. a 4-byte source (Int8Array) may * expand to a 16-byte target (Uint32Array) so that the target overlaps * the source both from beginning and the end (unlike in typical memmove). * * - Even if 'buf' pointers of the source and target differ, there's no * guarantee that their memory areas don't overlap. This may be the * case with external buffers. * * Even so, it is nice to optimize for the common case: * * - Source and target separate buffers or non-overlapping. * * - Source and target have a compatible type so that a plain byte copy * is possible. Note that while e.g. uint8 and int8 are compatible

* (coercion one way or another doesn't change the byte representation), * e.g. int8 and uint8clamped are NOT compatible when writing int8 * values into uint8clamped typedarray (-1 would clamp to 0 for instance). ** See test-bi-typedarray-proto-set.js.

47923. (quotient-remainder (* r B) s) using a dummy subtraction loop

47924. Note: either pointer may be NULL (at entry), so don't assert

47925. String-number-buffer/object -> coerce object to primitive (apparently without hint), then try again.

47926. * Property handling ** The API exposes only the most common property handling functions. * The caller can invoke EcmaScript built-ins for full control (e.g. * defineProperty, getOwnPropertyDescriptor).

47927. Format of magic, bits: * 0...1: elem size shift (0-3) * 2...5: elem type (DUK_HBUFFEROBJECT_ELEM_xxx)

47928. **comment:** Slow gathering of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. Handling * of negative numbers is a bit non-obvious in both cases.

label: code-design

47929. Built-in 'name' is not writable by default. Function '.name' * is writable to allow user code to set a '.name' on a native * function.

47930. reuse last emitted atom for remaining 'infinite' quantifier

47931. TypeError if rval is not an object (or lightfunc which should behave * like a Function instance).

47932. * Dragon4 slow path digit generation.

47933. Rule control flags.

47934. USE_PROP_HASH_PART

47935. don't compact objects; needed during object property allocation resize

47936. r <- (remainder (* r B) s)

47937. [... error arr]

47938. String is an own (virtual) property of a lightfunc.

47939. 16: getHours

47940. 'debugger'

47941. * String manipulation

47942. fmod() return value has same sign as input (negative) so * the result here will be in the range [-2,0], 1 indicates * odd. If x is -Infinity, NaN is returned and the odd check * always concludes "not odd" which results in desired outcome.

47943. DUK_USE_DEBUGGER_INSPECT

47944. Lookup 'key' from arguments internal 'map', perform a variable write if mapped. * Used in E5 Section 10.6 algorithm for [[DefineOwnProperty]] (used by [[Put]]). * Assumes stack top contains 'put' value (which is NOT popped).

47945. * Macro for object validity check ** Assert for currently guaranteed relations between flags, for instance.

47946. **comment:** Process debug messages until we are no longer paused.

label: code-design

47947. * Entries part

47948. [in_val in_val]

47949. DUK_TOK_LBRACKET

47950. 36 bits

47951. for zero size allocations NULL is allowed

47952. -> [... obj key value]

47953. * Some change(s) need to be made. Steps 7-11.

47954. -1 = error, -2 = allowed whitespace, -3 = padding ('='), 0...63 decoded bytes

47955. duk_new() will call the constructor using duk_handle_call(). * A constructor call prevents a yield from inside the constructor, * even if the constructor is an EcmaScript function.

47956. Select a safe loop count where no output checks are * needed assuming we won't encounter escapes. Input * bound checks are not necessary as a NUL (guaranteed) * will cause a SyntaxError before we read out of bounds.

47957. end of diff run

47958. * Buffer

47959. Note: ctx.steps is intentionally not reset, it applies to the entire unanchored match

47960. **comment:** XXX: use the pointer as a seed for now: mix in time at least

label: code-design

47961. XXX: stringtable emergency compaction?

47962. thr->ptr_curr_pc is set by bytecode executor early on entry

47963. Allow leading plus sign

47964. if slice not fully valid, treat as error

47965. Now two entries in the same slot, alloc list

47966. * The loop below must avoid issues with potential callstack * reallocations. A resize (and other side effects) may happen * e.g. due to finalizer/errhandler calls caused by a refzero or * mark-and-sweep. Arbitrary finalizers may run, because when * an environment record is refzero'd, it may refer to arbitrary * values which also become refzero'd. ** So, the pointer 'p' is re-looked-up below whenever a side effect * might have changed it.

47967. thr->heap->lj.value1 is already the value to throw

47968. Trailing whitespace has been eaten by duk_dec_value(), so if * we're not at end of input here, it's a SyntaxError.

47969. If user callback did not return source code, module loading * is finished (user callback initialized exports table directly).

47970. **comment:** * Emit a final RETURN. ** It would be nice to avoid emitting an unnecessary "return" opcode * if the current PC is not reachable. However, this cannot be reliably * detected; even if the previous instruction is an unconditional jump, * there may be a previous jump which jumps to current PC (which is the * case for iteration and conditional statements, for instance).

label: code-design

47971. * Because buffer values are often looped over, a number fast path * is important.

47972. starting line number

47973. x <- x - y, use t as temp

47974. EQUALITY EXPRESSION

47975. * Unicode support tables automatically generated during build.

47976. return thread

47977. **comment:** No catch variable, e.g. a try-finally; replace LDCONST with * NOP to avoid a bogus LDCONST.

label: code-design

47978. 0x08583b00

47979. Signed shift, named "arithmetic" (asl) because the result * is signed, e.g. 4294967295 << 1 -> -2. Note that result * must be masked.

47980. fmax() with args -0 and +0 is not guaranteed to return * +0 as EcmaScript requires.

47981. not a wordchar

47982. XXX: a "first" flag would suffice

47983. reg count

47984. Buffer used for generated digits, values are in the range [0,B-1].

47985. assume memset with zero size is OK

47986. If ctx->offset >= ctx->length, we "shift zeroes in" * instead of croaking.

47987. Define own property without inheritance looks and such. This differs from * [[DefineOwnProperty]] because special behaviors (like Array 'length') are * not invoked by this method. The caller must be careful to invoke any such * behaviors if necessary.

47988. Parse a number, other than NaN or +/- Infinity

47989. Must be a "string object", i.e. class "String"

47990. no decref needed for a number

47991. [... | (crud)]

47992. x <- y - z, require x >= y => z >= 0, i.e. y >= z

47993. 26 bits
47994. Used to support single-byte stream lookahead.
47995. Filename/line from C macros (`_FILE_`, `_LINE_`) are added as an * entry with a special format: (string, number). The number contains * the line and flags.
47996. -> [ToObject(this)]
47997. Buffer sizes for some UNIX calls. Larger than strictly necessary * to avoid Valgrind errors.
47998. \Udeadbeef
47999. * every(), some(), forEach(), map(), filter()
48000. Test without volatiles
48001. Lexer codepoint with additional info like offset/line number
48002. Low memory options
48003. Constants should be signed so that signed arithmetic involving them * won't cause values to be coerced accidentally to unsigned.
48004. Note: either pointer may be NULL (at entry), so don't assert.
48005. **comment:** Portability workaround is required for platforms without * unaligned access. The replacement code emulates little * endian access even on big endian architectures, which is * OK as long as it is consistent for a build.
 label: code-design
48006. tail call -> reuse current "frame"
48007. refcount is unsigned, so always true
48008. work list for objects to be finalized (by mark-and-sweep)
48009. **comment:** XXX: remove the preventcount and make yield walk the callstack? * Or perhaps just use a single flag, not a counter, faster to just * set and restore?
 label: code-design
48010. **comment:** still in use with verbose messages
 label: code-design
48011. stack is unbalanced, but: [<something> buf]
48012. parse ranges until character class ends
48013. 'delete'
48014. **comment:** unused now, not needed until Turkish/Azeri
 label: requirement
48015. No platform-specific parsing, this is not an error.
48016. mark finalizable as reachability roots
48017. **comment:** XXX: Add a proper condition. If formals list is omitted, recheck * handling for 'length' in `duk_js_push_closure()`; it currently relies * on `_Formals` being set. Removal may need to be conditional to debugging * being enabled/disabled too.
 label: code-design
48018. **comment:** XXX: naming is inconsistent: ATOM_END_GROUP ends an ASSERT_START_LOOKAHEAD
 label: code-design
48019. `ToUint32()` coercion required
48020. 'global'
48021. not popped by side effect
48022. Add a number containing: pc, activation flags. * * PC points to next instruction, find offending PC. Note that * PC == 0 for native code.
48023. Check array density and indicate whether or not the array part should be abandoned.
48024. * Duktape.Buffer: `toString()`, `valueOf()`
48025. descriptor type specific checks
48026. `DUK_USE_ERRTHROW`
48027. Can happen if `duk_throw()` is called on an empty * callstack.
48028. * Longjmp and other control flow transfer for the bytecode executor. * * The longjmp handler can handle all longjmp types: error, yield, and * resume (pseudotypes are never actually thrown). * * Error policy for longjmp: should not ordinarily throw errors; if errors * occur (e.g. due to out-of-memory) they bubble outwards rather than being * handled recursively.
48029. unary minus
48030. ignore non-strings
48031. Clamp to target's end if too long. * * NOTE: there's no overflow possibility in the comparison; * both `target_start` and `copy_size` are ≥ 0 and based on * values in `duk_int_t` range. Adding them as `duk_uint_t` * values is then guaranteed not to overflow.
48032. move pivot to its final place
48033. Setting to NULL causes 'caller' to be set to * 'null' as desired.
48034. **comment:** * Global state for working around missing variadic macros
 label: code-design
48035. * Allocate and initialize a new entry, resizing the properties allocation * if necessary. Returns entry index (`e_idx`) or throws an error if alloc fails. * * Sets the key of the entry (increasing the key's refcount), and updates * the hash part if it exists. Caller must set value and flags, and update * the entry value refcount. A decref for the previous value is not necessary.
48036. tolerates NULL `h_buf`
48037. forced_reg
48038. `DUK_TOK_PACKAGE`
48039. B -> target register * C -> constant index of identifier name
48040. no match, next case
48041. * Interned identifier is compared against reserved words, which are * currently interned into the heap context. See `genbuiltins.py`. * * Note that an escape in the identifier disables recognition of * keywords; e.g. "\u00069f = 1;" is a valid statement (assigns to * identifier named "if"). This is not necessarily compliant, * see `test-dec-escaped-char-in-keyword.js`. * * Note: "get" and "set" are awkward. They are not officially * ReservedWords (and indeed e.g. "var set = 1;" is valid), and * must come out as `DUK_TOK_IDENTIFIER`. The compiler needs to * work around this a bit.
48042. NULL if `tv_obj` is primitive
48043. lJ value2: thread
48044. Note: use 'curr' as a temp propdesc
48045. Setter APIs detect special year numbers (0...99) and apply a +1900 * only in certain cases. The legacy `getYear()` getter applies -1900 * unconditionally.
48046. * `substring()`, `substr()`, `slice()`
48047. 30: `setHours`
48048. Expression, J' terminates
48049. * Number checkers
48050. E5 Section 11.2.3, step 6.a.i
48051. -> [hobject props]
48052. XXX: shared handling for 'duk__expr_lhs'?
48053. Fix for <https://github.com/svaarala/duktape/issues/155>: * If 'default' is first clause (detected by `pc_prevcase < 0`) * we need to ensure we stay in the matching chain.
48054. eat 'in'
48055. This may trigger mark-and-sweep with arbitrary side effects, * including an attempted resize of the object we're resizing, * executing a finalizer which may add or remove properties of * the object we're resizing etc.
48056. Call with `count == DUK_LEXER_WINDOW_SIZE` to fill buffer initially.
48057. zero-based month
48058. * Must be very careful here, every DECREF may cause reallocation * of our valstack.
48059. expect_token
48060. periods
48061. idx of first argument on stack
48062. Add trailer if: * a) non-empty input * b) empty input and no (zero size) match found (step 11)

48063. check stack first
48064. mod_id may be NULL
48065. http://en.wikipedia.org/wiki/ANSI_escape_code
48066. since closure has NEWENV, never define DUK_STRIDX_INT_VARENV, as it * will be ignored anyway
48067. DUK_TOK_SET
48068. * Lookup variable and update in-place if found.
48069. 'Int8Array'
48070. Last component name in resolved path
48071. Must match src/extract_chars.py, generate_match_table3().
48072. Resize target buffer for requested size. Called by the macro only when the * fast path test (= there is space) fails.
48073. Ready to make the copy. We must proceed element by element * and must avoid any side effects that might cause the buffer * validity check above to become invalid. * * Although we work through the value stack here, only plain * numbers are handled which should be side effect safe.
48074. **comment:** * Compared to a direct macro expansion this wrapper saves a few * instructions because no heap dereferencing is required.
label: code-design
48075. Force pause if we were doing "step into" in another activation.
48076. allow fmt==NULL
48077. **comment:** * Relocate property allocation, moving properties to the new allocation. * * Includes key compaction, rehashing, and can also optionally abandoning * the array part, 'migrating' array entries into the beginning of the * new entry part. Arguments are not validated here, so e.g. new_h_size * MUST be a valid prime. * * There is no support for in-place reallocation or just compacting keys * without resizing the property allocation. This is intentional to keep * code size minimal. * * The implementation is relatively straightforward, except for the array * abandonment process. Array abandonment requires that new string keys * are interned, which may trigger GC. All keys interned so far must be * reachable for GC at all times; valstack is used for that now. * * Also, a GC triggered during this reallocation process must not interfere * with the object being resized. This is currently controlled by using * heap->mark_and_sweep_base_flags to indicate that no finalizers will be * executed (as they can affect ANY object) and no objects are compacted * (it would suffice to protect this particular object only, though). * * Note: a non-checked variant would be nice but is a bit tricky to * implement for the array abandonment process. It's easy for * everything else. * * Note: because we need to potentially resize the valstack (as part * of abandoning the array part), any tval pointers to the valstack * will become invalid after this call.
label: code-design
48078. Note: target registers a and a+1 may overlap with DUK_REGP(b) * and DUK_REGCONSTP(c). Careful here.
48079. duk_hcompiledfunction flags; quite version specific
48080. these are not necessarily 0 or 1 (may be other non-zero), that's ok
48081. +1 = finally
48082. Clamping needed if duk_int_t is 64 bits.
48083. 'id'
48084. output bufwriter
48085. -> x in [-2**31,2**31[
48086. DUK_USE_SELF_TESTS
48087. built-in strings
48088. * Process messages and send status if necessary. * * If we're paused, we'll block for new messages. If we're not * paused, we'll process anything we can peek but won't block * for more. Detach (and re-attach) handling is all localized * to duk_debug_process_messages() too. * * Debugger writes outside the message loop may cause debugger * detach1 phase to run, after which dbg_read_cb == NULL and * dbg_detaching != 0. The message loop will finish the detach * by running detach2 phase, so enter the message loop also when * detaching.
48089. * The error object has been augmented with a traceback and other * info from its creation point -- usually the current thread. * The error handler, however, is called right before throwing * and runs in the yielder's thread.
48090. * Error throwing helpers
48091. [... new_glob new_env new_glob new_glob]
48092. For lightfuncs, simply read the virtual property.
48093. 0x70-0x7f
48094. stable
48095. XXX: Could add fast path (for each transform callback) with direct byte * lookups (no shifting) and no explicit check for x < 0x80 before table * lookup.
48096. Duktape.Thread.resume()
48097. **comment:** * Note: of native Ecmascript objects, only Function instances * have a [[HasInstance]] internal property. Custom objects might * also have it, but not in current implementation. * * XXX: add a separate flag, DUK_HOBJECT_FLAG_ALLOW_INSTANCEOF?
label: code-design
48098. nothing to NULL
48099. use SameValue instead of non-strict equality
48100. a value is left on stack regardless of rc
48101. Detach is pending; can be triggered from outside the * debugger loop (e.g. Status notify write error) or by * previous message handling. Call detached callback * here, in a controlled state, to ensure a possible * reattach inside the detached_cb is handled correctly. * * Recheck for detach in a while loop: an immediate * reattach involves a call to duk_debugger_attach() * which writes a debugger handshake line immediately * inside the API call. If the transport write fails * for that handshake, we can immediately end up in a * "transport broken, detaching" case several times here. * Loop back until we're either cleanly attached or * fully detached. * * NOTE: Reset dbg_processing = 1 forcibly, in case we * re-attached; duk_debugger_attach() sets dbg_processing * to 0 at the moment.
48102. Run the finalizer, duk_hobject_run_finalizer() sets FINALIZED. * Next mark-and-sweep will collect the object unless it has * become reachable (i.e. rescued). FINALIZED prevents the * finalizer from being executed again before that.
48103. Note: this must match ToObject() behavior
48104. 'new' MemberExpression
48105. The code below is incorrect if .toString() or .valueOf() have * have been overridden. The correct approach would be to look up * the method(s) and if they resolve to the built-in function we * can safely bypass it and look up the internal value directly. * Unimplemented for now, abort fast path for boxed values.
48106. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx [21 bits]
48107. **comment:** XXX: avoid this at least when enum_target is an Array, it has an * array part, and no ancestor properties were included? Not worth * it for JSON, but maybe worth it for forEach().
label: code-design
48108. Report it to the debug client
48109. no compact
48110. restricted to regs
48111. dummy so sp won't get updated
48112. * Replace global object.
48113. require a (mutable) temporary as a result (or a const if allowed)
48114. contains value
48115. '%s' and NULL is not portable, so special case * it for debug printing.
48116. Compiler SyntaxErrors (and other errors) come first, and are * blamed by default (not flagged "noblame").
48117. Relevant PC is just before current one because PC is * post-incremented. This should match what error augment * code does.
48118. Parse formals.
48119. [... key arg1 ... argN func]
48120. exit bytecode execution with return value
48121. * Status message and helpers
48122. Note: 'func' is popped from valstack here, but it is * already reachable from the activation.
48123. points to LABEL; pc+1 = jump site for break; pc+2 = jump site for continue
48124. DUK_SELFTEST_H_INCLUDED
48125. Duktape object

48126. 0xb0-0xbf
48127. * 'key_obj' tracks keys encountered so far by associating an * integer with flags with already encountered keys. The checks * below implement E5 Section 11.1.5, step 4 for production: ** PropertyNameAndValueList: PropertyNameAndValueList , PropertyAssignment
48128. [... res]
48129. * UTF-8 decoder which accepts longer than standard byte sequences. * This allows full 32-bit code points to be used.
48130. accessor with defined getter
48131. **comment:** XXX: optimize for packed duk_tval directly?
 label: code-design
48132. * Augment created errors upon creation (not when they are thrown or * rethrown). __FILE__ and __LINE__ are not desirable here; the call * stack reflects the caller which is correct.
48133. allow source elements
48134. one token before
48135. 'filename'
48136. Neutered. We could go into the switch-case safely with * buf == NULL because check_length == 0. To avoid scanbuild * warnings, fail directly instead.
48137. Allowed.
48138. -> x in J-2**32, 2**32[
48139. no need to incref
48140. Ecmascript function
48141. embed: void ptr
48142. Lightfuncs are always native functions and have "newenv".
48143. This upper value has been experimentally determined; debug build will check * bigint size with assertions.
48144. **comment:** XXX: cover this with the generic >1 case?
 label: code-design
48145. * Variant 1 or 3
48146. 'enumerable'
48147. **comment:** * There are two [[Construct]] operations in the specification: * * - E5 Section 13.2.2: for Function objects * - E5 Section 15.3.4.5.2: for "bound" Function objects * * The chain of bound functions is resolved in Section 15.3.4.5.2, * with arguments "piling up" until the [[Construct]] internal * method is called on the final, actual Function object. Note * that the "prototype" property is looked up *only* from the * final object, *before* calling the constructor. * * Currently we follow the bound function chain here to get the * "prototype" property value from the final, non-bound function. * However, we let duk_handle_call() handle the argument "piling" * when the constructor is called. The bound function chain is * thus now processed twice. * * When constructing new Array instances, an unnecessary object is * created and discarded now: the standard [[Construct]] creates an * object, and calls the Array constructor. The Array constructor * returns an Array instance, which is used as the result value for * the "new" operation; the object created before the Array constructor * call is discarded. * * This would be easy to fix, e.g. by knowing that the Array constructor * will always create a replacement object and skip creating the fallback * object in that case. * * Note: functions called via "new" need to know they are called as a * constructor. For instance, built-in constructors behave differently * depending on how they are called.
 label: code-design
48148. zero size not an issue: pointers are valid
48149. fromPresent = true
48150. The counter value is one less than the init value: init value should * indicate how many instructions are executed before interrupt. To * execute 1 instruction (after interrupt handler return), counter must * be 0.
48151. assumes that allocated pointers and alloc funcs are valid * if res exists
48152. DUK_TOK_EQ
48153. * >>> struct.pack('>d', 102030405060).encode('hex') * '4237c17c6dc40000'
48154. XXX: add 'language' argument when locale/language sensitive rule * support added.
48155. Currently there are no deletable virtual properties, but * with force_flag we might attempt to delete one.
48156. **comment:** XXX: This helper is a bit awkward because the handling for the different iteration * callers is quite different. This now compiles to a bit less than 500 bytes, so with * 5 callers the net result is about 100 bytes / caller.
 label: code-design
48157. Infinity
48158. **comment:** XXX: unoptimal use of temps, resetting
 label: code-design
48159. -> [... regexp string] -> [... res_obj]
48160. token type (with reserved words as DUK_TOK_IDENTIFIER)
48161. Note: assumes that these string indexes are 8-bit, genstrings.py must ensure that
48162. start variant 3/4 left-hand-side code (L1 in doc/compiler.rst example)
48163. # argument registers target function wants (< 0 => "as is")
48164. 'magic' constants for Murmurhash2
48165. If something is thrown with the debugger attached and nobody will * catch it, execution is paused before the longjmp, turning over * control to the debug client. This allows local state to be examined * before the stack is unwound. Errors are not intercepted when debug * message loop is active (e.g. for Eval).
48166. Object built-in methods
48167. Skip fully.
48168. switch (ch2)
48169. type
48170. * Allocate memory with garbage collection
48171. * Main switch for statement / source element type.
48172. Shared entry handling for object/array serialization.
48173. overwrite sep
48174. [message error message]
48175. [... parent stash stash] -> [... parent stash]
48176. Non-verbose errors for low memory targets: no file, line, or message.
48177. DUK_BUFOBJ_ARRAYBUFFER
48178. **comment:** * Recursion limit check. * * Note: there is no need for an "ignore recursion limit" flag * for duk_handle_safe_call now.
 label: code-design
48179. Exponent handling: if exponent format is used, record exponent value and * fake k such that one leading digit is generated (e.g. digits=123 -> "1.23"). * * toFixed() prevents exponent use; otherwise apply a set of criteria to * match the other API calls (toString(), toPrecision, etc).
48180. **comment:** We want the argument expression value to go to "next temp" * without additional moves. That should almost always be the * case, but we double check after expression parsing. * * This is not the cleanest possible approach.
 label: code-design
48181. seconds, doesn't fit into 16 bits
48182. unreferenced w/o tracebacks
48183. We want to avoid making a copy to process set() but that's * not always possible: the source and the target may overlap * and because element sizes are different, the overlap cannot * always be handled with a memmove() or choosing the copy * direction in a certain way. For example, if source type is * uint8 and target type is uint32, the target area may exceed * the source area from both ends! * * Note that because external buffers may point to the same * memory areas, we must ultimately make this check using * pointers. * * NOTE: careful with side effects: any side effect may cause * a buffer resize (or external buffer pointer/length update)!
48184. **comment:** NEXTENUM needs a jump slot right after the main instruction. * When the JUMP is taken, output spilling is not needed so this * workaround is possible. The jump slot PC is exceptionally * plumbed through comp_ctx to minimize call sites.
 label: code-design

48185. must be updated to work properly (e.g. creation of 'arguments')
48186. Unescaped '/' ANYWHERE in the regexp (in disjunction, '*' inside a character class, ...) => same escape works.
48187. 1:1 or special conversions, but not locale/context specific: script generated rules
48188. roughly 0.5 kB
48189. * Value stack resize and stack top adjustment helper. ** XXX: This should all be merged to duk_valstack_resize_raw().
48190. **comment:** * Macro support functions for reading/writing raw data. ** These are done using mempcy to ensure they're valid even for unaligned * reads/writes on platforms where alignment counts. On x86 at least gcc * is able to compile these into a bswap+mov. "Always inline" is used to * ensure these macros compile to minimal code. ** Not really bufwriter related, but currently used together.
label: code-design
48191. If tracebacks are enabled, the '_Tracedata' property is the only * thing we need: 'fileName' and 'lineNumber' are virtual properties * which use '_Tracedata'.
48192. **comment:** XXX: could duk_is_undefined() provide defaulting undefined to 'len' * (the upper limit)?
label: code-design
48193. DUK_TOK_WITH
48194. 0xa0-0xaf
48195. * Table for base-64 decoding
48196. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor).
label: code-design
48197. [... buf]
48198. * Regexp compilation. ** See doc/regexp.rst for a discussion of the compilation approach and * current limitations. ** Regexp bytecode assumes jumps can be expressed with signed 32-bit * integers. Consequently the bytecode size must not exceed 0x7fffffffL. * The implementation casts duk_size_t (buffer size) to duk_(u)int32_t * in many places. Although this could be changed, the bytecode format * limit would still prevent regexps exceeding the signed 32-bit limit * from working. ** XXX: The implementation does not prevent bytecode from exceeding the * maximum supported size. This could be done by limiting the maximum * input string size (assuming an upper bound can be computed for number * of bytecode bytes emitted per input byte) or checking buffer maximum * size when emitting bytecode (slower).
48199. Production allows 'DecimalDigits', including leading zeroes
48200. Note: must coerce to a (writable) temp register, so that e.g. "!x" where x * is a reg-mapped variable works correctly (does not mutate the variable register).
48201. [] -> []
48202. key
48203. * Exception for Duktape internal throws when C++ exceptions are used * for long control transfers. ** Doesn't inherit from any exception base class to minimize the chance * that user code would accidentally catch this exception.
48204. -> [array totalLength buf]
48205. Maximum prototype traversal depth. Sanity limit which handles e.g. * prototype loops (even complex ones like 1->2->3->4->2->3->4->2->3->4).
48206. is valid size
48207. [O Properties obj]
48208. extra coercion of strings is OK
48209. **comment:** XXX: increase ctx->expr_tokens here for every consumed token * (this would be a nice statistic)?
label: code-design
48210. FAIL
48211. Two digit octal escape, digits validated. ** The if-condition is a bit tricky. We could catch e.g. *\039 in the three-digit escape and fail it there (by * validating the digits), but we want to avoid extra * additional validation code.
48212. For n == 0, Node.js ignores totalLength argument and * returns a zero length buffer.
48213. * Push result object and init its flags
48214. number of continuation (non-initial) bytes in [0x80,0xbff]
48215. External buffer with 'curr_alloc' managed by user code and pointing to an * arbitrary address. When heap pointer compression is not used, this struct * has the same layout as duk_hbuffer_dynamic.
48216. **comment:** Note: we can reuse 'desc' here
label: code-design
48217. empty statement with an explicit semicolon
48218. * Constructor
48219. * Array exotic behaviors can be implemented at this point. The local variables * are essentially a 'value copy' of the input descriptor (Desc), which is modified * by the Array [[DefineOwnProperty]] (E5 Section 15.4.5.1).
48220. * Expression parsing. ** Upon entry to 'expr' and its variants, 'curr_tok' is assumed to be the * first token of the expression. Upon exit, 'curr_tok' will be the first * token not part of the expression (e.g. semicolon terminating an expression * statement).
48221. The index range space is conceptually the array part followed by the * entry part. Unlike normal enumeration all slots are exposed here as * is and return 'unused' if the slots are not in active use. In particular * the array part is included for the full a_size regardless of what the * array .length is.
48222. * toString()
48223. shuffle registers if large number of regs/consts
48224. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8
48225. **comment:** XXX: add support for variables to not be register bound always, to * handle cases with a very large number of variables?
label: code-design
48226. entry_top + 2
48227. [result]
48228. * Determine call type, then finalize activation, shift to * new value stack bottom, and call the target.
48229. [source]
48230. * Arguments object creation. ** Creating arguments objects involves many small details, see E5 Section * 10.6 for the specific requirements. Much of the arguments object exotic * behavior is implemented in duk_hobject_props.c, and is enabled by the * object flag DUK_HOBJECT_FLAG_EXOTIC_ARGUMENTS.
48231. **comment:** XXX: space in valstack? see discussion in duk_handle_call_xxx().
label: code-design
48232. combines steps 2 and 5; -len ensures max() not needed for step 5
48233. for register-bound and declarative env identifiers
48234. bound functions are never in act 'func'
48235. Lightfuncs cannot be bound.
48236. **comment:** The resulting buffer object gets the same class and prototype as * the buffer in 'this', e.g. if the input is a Node.js Buffer the * result is a Node.js Buffer; if the input is a Float32Array, the * result is a Float32Array. ** For the class number this seems correct. The internal prototype * is not so clear: if 'this' is a bufferobject with a non-standard * prototype object, that value gets copied over into the result * (instead of using the standard prototype for that object type).
label: code-design
48237. Don't allow mixed padding and actual chars.
48238. reject RegExp literal on next advance() call; needed for handling IdentifierName productions
48239. DUK_TOK_CONTINUE
48240. unbalanced stack on purpose
48241. arguments MUST have an initialized lexical environment reference
48242. **comment:** XXX: sanitize to printable (and maybe ASCII)
label: code-design
48243. tags
48244. regexp compiler should catch these

48245. **comment:** Current 'this' binding is the value just below idx_bottom. * Previously, 'this' binding was handled with an index to the * (calling) valstack. This works for everything except tail * calls, which must not "cumulate" valstack temps.
label: code-design

48246. * ctx->prev_token token to process with duk__expr_nud() * ctx->curr_token updated by caller * * Note: the token in the switch below has already been eaten.
48247. -> [... voidp true]

48248. The E5.1 specification does not seem to allow IdentifierPart characters * to be used as identity escapes. Unfortunately this includes '\$', which * cannot be escaped as '\\$'; it needs to be escaped e.g. as '\u0024'. * Many other implementations (including V8 and Rhino, for instance) do * accept '\$' as a valid identity escape, which is quite pragmatic. * See: test-regexp-identity-escape-dollar.js.

48249. mp < b^e+1)

48250. * Debugger handling for executor restart * * Check for breakpoints, stepping, etc, and figure out if we should execute * in checked or normal mode. Note that we can't do this when an activation * is created, because breakpoint status (and stepping status) may change * later, so we must recheck every time we're executing an activation. * This primitive should be side effect free to avoid changes during check.

48251. at most sizeof(buf) - 1

48252. val

48253. Old solution: don't iterate, incorrect

48254. * API calls related to general value stack manipulation: resizing the value * stack, pushing and popping values, type checking and reading values, * coercing values, etc. * * Also contains internal functions (such as duk_get_tval()), defined * in duk_api_internal.h, with semantics similar to the public API.

48255. No BC shuffling now.

48256. syntactically left-associative but parsed as right-associative

48257. * Local result type for duk__get_identifier_reference() lookup.

48258. **comment:** XXX: would be nice to enumerate an object at specified index
label: code-design

48259. ''

48260. * Heap stringtable handling, string interning.

48261. By default the global object is read-only which is often much * more of an issue than having read-only built-in objects (like * RegExp, Date, etc). Use a RAM-based copy of the global object * and the global environment object for convenience.

48262. Creation time error augmentation

48263. The casts through duk_intr_pt is to avoid the following GCC warning: * * warning: cast from pointer to integer of different size [-Wpointer-to-int-cast] * * This still generates a /Wp64 warning on VS2010 when compiling for x86.

48264. **comment:** * Note: currently register bindings must be fixed for the entire * function. So, even though the catch variable is in a register * we know, we must use an explicit environment record and slow path * accesses to read/write the catch binding to make closures created * within the catch clause work correctly. This restriction should * be fixable (at least in common cases) later. * * See: test-bug-catch-binding-2.js. * * XXX: improve to get fast path access to most catch clauses.
label: code-design

48265. check that byte has the form 10xx xxxx

48266. * Memory constants

48267. return 'res' (of right part) as our result

48268. -> [func funcname env]

48269. this variant is used when an assert would generate a compile warning by * being always true (e.g. >= 0 comparison for an unsigned value

48270. Stringcache is used for speeding up char-offset-to-byte-offset * translations for non-ASCII strings.

48271. **comment:** Note: strictness is *not* inherited from the current Duktape/C. * This would be confusing because the current strictness state * depends on whether we're running inside a Duktape/C activation * (= strict mode) or outside of any activation (= non-strict mode). * See tests/api/test-eval-strictness.c for more discussion.
label: code-design

48272. * Finalizer check. * * Note: running a finalizer may have arbitrary side effects, e.g. * queue more objects on refzero_list (tail), or even trigger a * mark-and-sweep. * * Note: quick reject check should match vast majority of * objects and must be safe (not throw any errors, ever).

48273. Get Proxy target object. If the argument is not a Proxy, return it as is. * If a Proxy is revoked, an error is thrown.

48274. * Assertion helpers.

48275. is_setget

48276. * Compute new alloc size and alloc new area. * * The new area is allocated as a dynamic buffer and placed into the * valstack for reachability. The actual buffer is then detached at * the end. * * Note: heap_mark_and_sweep_base_flags are altered here to ensure * no-one touches this object while we're resizing and rehashing it. * The flags must be reset on every exit path after it. Finalizers * and compaction is prevented currently for all objects while it * would be enough to restrict it only for the current object.

48277. replacer function

48278. [...]

48279. **comment:** remove the string (mark DELETED), could also call * duk_heap_string_remove() but that would be slow and * pointless because we already know the slot.
label: code-design

48280. +(function(){})-> NaN

48281. **comment:** * Date built-ins * * Unlike most built-ins, Date has some platform dependencies for getting * UTC time, converting between UTC and local time, and parsing and * formatting time values. These are all abstracted behind DUK_USE_xxx * config options. There are built-in platform specific providers for * POSIX and Windows, but external providers can also be used. * * See doc/datetime.rst.
label: code-design

48282. Reject attempt to change a read-only object.

48283. final result is already in 'res'

48284. 21: getUTCSeconds

48285. Eval is just a wrapper now.

48286. With ROM objects "needs refcount update" is true when the value is * heap allocated and is not a ROM object.

48287. 22: getMilliseconds

48288. binary arithmetic using extraops; DUK_EXTRAOP_INSTOF etc

48289. combined algorithm matching E5 Sections 9.5 and 9.6

48290. Evaluates to (duk_uint8_t *) pointing to start of area.

48291. module.id = resolved_id

48292. Property access expressions ('a[b]') are critical to correct * LHS evaluation ordering, see test-dev-assign-eval-order*.js. * We must make sure that the LHS target slot (base object and * key) don't change during RHS evaluation. The only concrete * problem is a register reference to a variable-bound register * (i.e., non-temp). Require temp regs for both key and base. * * Don't allow a constant for the object (even for a number * etc), as it goes into the 'A' field of the opcode.

48293. Maximum number of characters in formatted value.

48294. Read fully.

48295. * Prototypes

48296. For indirect allocs.

48297. always allow 'in', coerce to 'tr' just in case

48298. DUK_USE_DEBUGGER_SUPPORT && (DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT)

48299. Pointer object internal value is immutable

48300. Leave 'value' on stack

48301. prev char

48302. * [[DefaultValue]] (E5 Section 8.12.8) * * ==> implemented in the API.

48303. 'extends'

48304. result: hash_prime(floor(1.2 * e_size))

48305. bp is assumed to be even
48306. stridx_func[] must be static to allow initializer with old compilers like BCC
48307. Positive index can be higher than valstack top but must * not go above allocated stack (equality is OK).
48308. out_desc is left untouched (possibly garbage), caller must use return * value to determine whether out_desc can be looked up
48309. If buffer has been exhausted, truncate bitstream
48310. Can be called multiple times with no harm.
48311. Helper for component setter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), modify one or more components as specified, recompute * the time value, set it as the internal value. Finally, push the * new time value as a return value to the value stack and return 1 * (caller can then tail call us).
48312. error
48313. 1:1 conversion
48314. Encode into a double constant (53 bits can encode $6 \times 8 = 48$ bits + 3-bit length)
48315. not LS/PS
48316. invalid codepoint encoding or codepoint
48317. **comment:** packed advance/token number macro used by multiple functions
 label: code-design
48318. The JO(value) operation: encode object. * * Stack policy: [object] -> [object].
48319. DUK_TOK_BAND_EQ
48320. remaining
48321. **comment:** * Duktape built-ins * * Size optimization note: it might seem that vararg multipurpose functions * like fin(), enc(), and dec() are not very size optimal, but using a single * user-visible Ecmascript function saves a lot of run-time footprint; each * Function instance takes >100 bytes. Using a shared native helper and a * 'magic' value won't save much if there are multiple Function instances * anyway.
 label: code-design
48322. Value is required to be a number in the fast path so there * are no side effects in write coercion.
48323. Faster but value stack overruns are memory unsafe.
48324. register for writing value of 'non-empty' statements (global or eval code), -1 is marker
48325. * Possible pending array length update, which must only be done * if the actual entry write succeeded.
48326. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.
48327. If XUTF-8 decoding fails, treat the offending byte as a codepoint directly * and go forward one byte. This is of course very lossy, but allows some kind * of output to be produced even for internal strings which don't conform to * XUTF-8. All standard Ecmascript strings are always CESU-8, so this behavior * does not violate the Ecmascript specification. The behavior is applied to * all modes, including Ecmascript standard JSON. Because the current XUTF-8 * decoding is not very strict, this behavior only really affects initial bytes * and truncated codepoints. * * Another alternative would be to scan forwards to start of next codepoint * (or end of input) and emit just one replacement codepoint.
48328. For initial entry use default value; zero forces an * interrupt before executing the first instruction.
48329. already in correct reg
48330. the next 'if' is guaranteed to match after swap
48331. Encoding state. Heap object references are all borrowed.
48332. -> [... repl_value]
48333. no prototype
48334. Note: 'q' is top-1
48335. **comment:** * Abandon array part because all properties must become non-configurable. * Note that this is now done regardless of whether this is always the case * (skips check, but performance problem if caller would do this many times * for the same object; not likely).
 label: code-design
48336. **comment:** XXX: incref by count (2) directly
 label: code-design
48337. ')'
48338. source is a function expression (used for Function constructor)
48339. * Variant 2
48340. Stored in duk_heapdr h_extra16.
48341. Note: intentionally signed.
48342. 'Date'
48343. input linenumber at input_offset (not window[0]), init to 1
48344. **comment:** * Missing bytes snip base64 example * 0 4 XXXX * 1 3 XXX= * 2 2 XX==
 label: code-design
48345. * The remaining matches are emitted as sequence of SPLITs and atom * copies; the SPLITs skip the remaining copies and match the sequel. * This sequence needs to be emitted starting from the last copy * because the SPLITs are variable length due to the variable length * skip offset. This causes a lot of memory copying now. * * Example structure (greedy, match maximum # atoms): * * SPLIT1 LSEQ * (atom) * SPLIT1 LSEQ ; < the byte length of this instruction is needed * (atom) ; to encode the above SPLIT1 correctly * * LSEQ:
48346. idx_value
48347. [... this_new | arg1 ... argN]
48348. -----XX XXXX-----
48349. * Get old and new length
48350. **comment:** Hot variables for interpretation. Critical for performance, * but must add sparingly to minimize register shuffling.
 label: code-design
48351. 38: getYear
48352. initjs data is NUL terminated
48353. this.raw(buffer)
48354. return h_bufres
48355. Here we rely on duk_hstring instances always being zero * terminated even if the actual string is not.
48356. Must coerce key: if key is an object, it may coerce to e.g. 'length'.
48357. -> [... regexp_instance]
48358. Final result is at stack top.
48359. Strict equality is NOT enough, because we cannot use the same * constant for e.g. +0 and -0.
48360. right exists, [[Put]] regardless whether or not left exists
48361. * Return target object.
48362. reuse 'res' as 'left'
48363. st_used remains the same, DELETED is counted as used
48364. is_join
48365. binding power must be high enough to NOT allow comma expressions directly
48366. Note: these variables must reside in 'curr_func' instead of the global * context: when parsing function expressions, expression parsing is nested.
48367. Abandon array part, moving array entries into entries part. * This requires a props resize, which is a heavy operation. * We also compact the entries part while we're at it, although * this is not strictly required.
48368. _Formals
48369. base of known return values
48370. TimeClip(), which also handles Infinity -> NaN conversion
48371. [... obj]
48372. Encoding endianness must match target memory layout, * build scripts and genbuiltins.py must ensure this.

48373. flags used for property attributes in duk_propdesc and packed flags
48374. directly uses slow accesses
48375. * Finalizer torture. Do one fake finalizer call which causes side effects * similar to one or more finalizers on actual objects.
48376. add AC*2^32
48377. result: updated refcount
48378. Entry level info.
48379. catch depth from current func
48380. indexOf: clamp fromIndex to [-len, len] * (if fromIndex == len, for-loop terminates directly) * * lastIndexOf: clamp fromIndex to [-len - 1, len - 1] * (if clamped to -len-1 -> fromIndex becomes -1, terminates for-loop directly)
48381. decoding
48382. inline initializer for coercers[] is not allowed by old compilers like BCC
48383. * Create normalized 'source' property (E5 Section 15.10.3).
48384. variable access
48385. Double casting for pointer to avoid gcc warning (cast from pointer to integer of different size)
48386. **comment:** XXX: refactor out?
 label: code-design
48387. same for both error and each subclass like TypeError
48388. [...] res_obj re_obj input bc saved_buf]
48389. B -> register for writing enumerator object * C -> value to be enumerated (register)
48390. entire string is whitespace
48391. 'multiline'
48392. '+'
48393. * Parse an identifier and then check whether it is: * - reserved word (keyword or other reserved word) * - "null" (NullLiteral) * - "true" (BooleanLiteral) * - "false" (BooleanLiteral) * - anything else => identifier * * This does not follow the E5 productions cleanly, but is * useful and compact. * * Note that identifiers may contain Unicode escapes, * see E5 Sections 6 and 7.6. They must be decoded first, * and the result checked against allowed characters. * The above if-clause accepts an identifier start and an * '\ character -- no other token can begin with a '\'. * * Note that "get" and "set" are not reserved words in E5 * specification so they are recognized as plain identifiers * (the tokens DUK_TOK_GET and DUK_TOK_SET are actually not * used now). The compiler needs to work around this. * * Strictly speaking, following Ecmascript longest match * specification, an invalid escape for the first character * should cause a syntax error. However, an invalid escape * for IdentifierParts should just terminate the identifier * early (longest match), and let the next tokenization * fail. For instance Rhino croaks with 'foo\z' when * parsing the identifier. This has little practical impact.
48394. 'lineNumber'
48395. refer to callstack entries below current
48396. statement parsing
48397. **comment:** Pick a destination register. If either base value or key * happens to be a temp value, reuse it as the destination. * * XXX: The temp must be a "mutable" one, i.e. such that no * other expression is using it anymore. Here this should be * the case because the value of a property access expression * is neither the base nor the key, but the lookup result.
 label: code-design
48398. Property access expressions are critical for correct LHS ordering, * see comments in duk__expr()
48399. ignore retval -> [key]
48400. %ld
48401. * Octal escape or zero escape: * \0 (lookahead not DecimalDigit) * \1 ... \7 (lookahead not DecimalDigit) * \ZeroToThree OctalDigit (lookahead not DecimalDigit) * \FourToSeven OctalDigit (no lookahead restrictions) * \ZeroToThree OctalDigit OctalDigit (no lookahead restrictions) * * Zero escape is part of the standard syntax. Octal escapes are * defined in E5 Section B.1.2, and are only allowed in non-strict mode. * Any other productions starting with a decimal digit are invalid.
48402. **comment:** XXX: for Object.keys() the [[OwnPropertyKeys]] result (trap result) * should be filtered so that only enumerable keys remain. Enumerability * should be checked with [[GetOwnProperty]] on the original object * (i.e., the proxy in this case). If the proxy has a getOwnPropertyDescriptor * trap, it should be triggered for every property. If the proxy doesn't have * the trap, enumerability should be checked against the target object instead. * We don't do any of this now, so Object.keys() and Object.getOwnPropertyNames() * return the same result now for proxy traps. We still do clean up the trap * result, so that Object.keys() and Object.getOwnPropertyNames() will return a * clean array of strings without gaps.
 label: code-design
48403. syntax error
48404. don't push value
48405. 'buffer'
48406. inherit initial strictness from parent
48407. module
48408. Without tracebacks the concrete .fileName and .lineNumber need * to be added directly.
48409. not emergency
48410. * Raw memory calls: relative to heap, but no GC interaction
48411. Lookup 'key' from arguments internal 'map', perform a variable lookup * if mapped, and leave the result on top of stack (and return non-zero). * Used in E5 Section 10.6 algorithms [[Get]] and [[GetOwnProperty]].
48412. Evaluate RHS only when LHS is safe.
48413. debugger detach; used to avoid calling detach handler recursively
48414. no catchers
48415. * Variant 4
48416. 'package'
48417. nothing now
48418. * Reset trigger counter
48419. * Preliminaries
48420. count is already incremented, take into account
48421. **comment:** * Setters. * * Setters are a bit more complicated than getters. Component setters * break down the current time value into its (normalized) component * parts, replace one or more components with -unnormalized- new values, * and the components are then converted back into a time value. As an * example of using unnormalized values: * * var d = new Date(1234567890); * * is equivalent to: * * var d = new Date(0); * d.setUTCMilliseconds(1234567890); * * A shared native helper to provide almost all setters. Magic value * contains a set of flags and also packs the "maxnargs" argument. The * helper provides: * * setMilliseconds() * setUTCMilliseconds() * setSeconds() * setUTCSeconds() * setMinutes() * setUTCMinutes() * setHours() * setUTCHours() * setDate() * setUTCDate() * setMonth() * setUTCMonth() * setFullYear() * setUTCFullYear() * setYear() * * Notes: * * - Date.prototype.setYear() (Section B addition): special year check * is omitted. NaN / Infinity will just flow through and ultimately * result in a NaN internal time value. * * - Date.prototype.setYear() does not have optional arguments for * setting month and day-in-month (like setFullYear()), but we indicate * 'maxnargs' to be 3 to get the year written to the correct component * index in duk__set_part_helper(). The function has nargs == 1, so only * the year will be set regardless of actual argument count.
 label: code-design
48422. inline string concatenation
48423. DUK_USE_JX && DUK_USE_JC
48424. **comment:** XXX: ideally this would be just one flag (maybe a derived one) so * that a single bit test is sufficient to check the condition.
 label: test
48425. [key] (coerced)
48426. **comment:** XXX: no optimized variants yet
 label: requirement
48427. Encode a double (checked by caller) from stack top. Stack top may be * replaced by serialized string but is not popped (caller does that).
48428. This function is only called for objects with array exotic behavior. * The [[DefineOwnProperty]] algorithm for arrays requires that * 'length' can never have a value outside the unsigned 32-bit range, * attempt to write such a value is a RangeError. Here we can thus * assert for this. When Duktape internals go around the official

* property write interface (doesn't happen often) this assumption is * easy to accidentally break, so such code must be written carefully. * See test-bi-array-push-maxlen.js.

48429. **comment:** DUK__ADVANCECHARS(lex_ctx, 2) would be correct here, but it unnecessary
label: code-design

48430. 'base64'

48431. [sep ToObject(this) len sep]

48432. **comment:** * Various sanity checks for typing
label: code-design

48433. Check whether statement list ends.

48434. invalidates tv1, tv2

48435. Note: MUST NOT wipe_and_return here, as heap->lj must remain intact

48436. -> sets 'f' and 'e'

48437. Check for maximum string length

48438. * Parse an individual source element (top level statement) or a statement. * * Handles labeled statements automatically (peeling away labels before * parsing an expression that follows the label(s)). * * Upon entry, 'curr_tok' contains the first token of the statement (parsed * in "allow regexp literal" mode). Upon exit, 'curr_tok' contains the first * token following the statement (if the statement has a terminator, this is * the token after the terminator).

48439. **comment:** loop_stack_index could be perhaps be replaced by 'depth', but it's nice * to not couple these two mechanisms unnecessarily.

label: code-design

48440. * Fast path for defining array indexed values without interning the key. * This is used by e.g. code for Array prototype and traceback creation so * must avoid interning.

48441. accessor

48442. set to 1 if any match exists (needed for empty input special case)

48443. this should never be possible, because the switch-case is * comprehensive

48444. avoid double coercion

48445. 'DataView'

48446. * Sweep stringtable

48447. [A B C D E F G H] rel_index = 2, del_count 3, item count 1 * -> [A B F G H] (conceptual intermediate step) * -> [A B . F G H] (placeholder marked) * [A B C F G H] (actual result at this point, C will be replaced)

48448. DUK_TOK_IN

48449. jump to false

48450. **comment:** XXX: resolve macro definition issue or call through a helper function?

label: code-design

48451. **comment:** Sanity workaround for handling functions with a large number of * constants at least somewhat reasonably. Otherwise checking whether * we already have the constant would grow very slow (as it is O(N^2)).

label: code-design

48452. heapobj recursion depth when deep printing is selected

48453. basic types

48454. temp reset is not necessary after duk__parse_stmt(), which already does it

48455. mark-and-sweep: reachable

48456. Error object is augmented at its creation here.

48457. DUK_USE_COMMONJS_MODULES

48458. at least effective 'this'

48459. * Default allocation functions. * * Assumes behavior such as malloc allowing zero size, yielding * a NULL or a unique pointer which is a no-op for free.

48460. DUK_USE_LIGHTFUNC_BUILTINS

48461. some derived types

48462. functions always have a NEWENV flag, i.e. they get a * new variable declaration environment, so only lex_env * matters here.

48463. x == +Infinity

48464. temp_start+0 = key, temp_start+1 = closure

48465. E5.1 Section 15.1.3.2: empty

48466. DUK_TOK_FINALY

48467. **comment:** Static call style.

label: code-design

48468. 'deleteProperty'

48469. Compiling state of one function, eventually converted to duk_hcompiledfunction

48470. to body

48471. Coerce top into Object.prototype.toString() output.

48472. XXX: this is quite clunky. Add Unicode helpers to scan backwards and * forwards with a callback to process codepoints?

48473. actual update happens once write has been completed without * error below.

48474. * 'func' is now a non-bound object which supports [[HasInstance]] * (which here just means DUK_HOBJECT_FLAG_CALLABLE). Move on * to execute E5 Section 15.3.5.3.

48475. Target object must be checked for a conflicting * non-configurable property.

48476. property exists, either 'desc' is empty, or all values * match (SameValue)

48477. [func thisArg argArray]

48478. p overshoots

48479. prefer direct

48480. **comment:** XXX: what about statements which leave a half-cooked value in 'res' * but have no stmt value? Any such statements?

label: code-design

48481. Must restart in all cases because we NULLed thr->ptr_curr_pc.

48482. MAXVAL is inclusive

48483. "release" preallocated temps since we won't need them

48484. temp reg value for start of loop

48485. checked by Duktape.Thread.resume()

48486. * Union aliasing, see misc/clang_aliases.c.

48487. [offset value littleEndian], when DUK__FLD_TYPEDARRAY (regardless of ftype)

48488. * Replacements for missing platform functions. * * Unlike the originals, fpclassify() and signbit() replacements don't * work on any floating point types, only doubles. The C typing here * mimics the standard prototypes.

48489. eat closing quote

48490. setup many variables in nc_ctx

48491. **comment:** XXX: fast int-to-double

label: code-design

48492. Native function pointer may be different from a void pointer, * and we serialize it from memory directly now (no byte swapping etc).

48493. **comment:** XXX: shorter version for 12-byte representation?

label: code-design

48494. a fresh require() with require.id = resolved target module id

48495. Constructor

48496. elision - flush

48497. temp reg for key literal

48498. Unlike break/continue, throw statement does not allow an empty value.

48499. DUK_TOK_VAR
48500. Useful for internal call sites where we either expect an object (function) * or a lightfunc. Accepts an object (returned as is) or a lightfunc (coerced * to an object).
 Return value is NULL if value is neither an object nor a * lightfunc.
48501. char offset in [0, h_input->clen] (both ends inclusive), checked before entry
48502. NOTE: fmod(x) result sign is same as sign of x, which * differs from what Javascript wants (see Section 9.6).
48503. [...] bytecode escaped_source]
48504. [obj key desc value get set curr_value]
48505. >>
48506. msec
48507. unary plus of a number is identity
48508. equals and strict equals
48509. * Loose equality, strict equality, and SameValue (E5 Sections 11.9.1, 11.9.4, * 9.12). These have much in common so they can share some helpers. * * Future work notes: * * - Current implementation (and spec definition) has recursion; this should * be fixed if possible. * * - String-to-number coercion should be possible without going through the * value stack (and be more compact) if a shared helper is invoked.
48510. The debugger protocol doesn't support a plain integer encoding for * the full 32-bit unsigned range (only 32-bit signed). For now, * unsigned 32-bit values simply written as signed ones. This is not * a concrete issue except for 32-bit heapdfr fields. Proper solutions * would be to (a) write such integers as IEEE doubles or (b) add an * unsigned 32-bit dvalue.
48511. **comment:** * XXX: attempt to get the call result to "next temp" whenever * possible to avoid unnecessary register shuffles. * * XXX: CSPROP (and CSREG) can overwrite the call target register, and save one temp, * if the call target is a temporary register and at the top of the temp reg "stack".
 label: code-design
48512. tv points to element just below prev top
48513. * Duktape/C function magic
48514. * Identifier handling
48515. error gets its 'name' from the prototype
48516. Use a new environment but there's no 'arguments' object; * delayed environment initialization. This is the most * common case.
48517. E5 Sections 11.9.1, 11.9.3
48518. Capital sigma occurred at "end of word", lowercase to * U+03C2 = GREEK SMALL LETTER FINAL SIGMA. Otherwise * fall through and let the normal rules lowercase it to * U+03C3 = GREEK SMALL LETTER SIGMA.
48519. current digit
48520. * Extern
48521. * Mark objects on finalize_list. *
48522. **comment:** XXX: repetition of stack pre-checks -> helper or macro or inline
 label: code-design
48523. **comment:** * Data follows the struct header. The struct size is padded by the * compiler based on the struct members. This guarantees that the * buffer data will be aligned-by-4 but not necessarily aligned-by-8. * * On platforms where alignment does not matter, the struct padding * could be removed (if there is any). On platforms where alignment * by 8 is required, the struct size must be forced to be a multiple * of 8 by some means. Without it, some user code may break, and also * Duktape itself breaks (e.g. the compiler stores duk_tvals in a * dynamic buffer).
 label: code-design
48524. 'valueOf'
48525. XXX: potential issue with signed pointers, p_end < p.
48526. * Helpers to resize properties allocation on specific needs.
48527. **comment:** The line number tracking is a bit inconsistent right now, which * affects debugger accuracy. Mostly call sites emit opcodes when * they have parsed a token (say a terminating semicolon) and called * duk__advance(). In this case the line number of the previous * token is the most accurate one (except in prologue where * prev_token.start_line is 0). This is probably not 100% correct * right now.
 label: code-design
48528. shared object part
48529. "Have" flags must not be conflicting so that they would * apply to both a plain property and an accessor at the same * time.
48530. [...] re_obj input]
48531. Backpointers.
48532. string access cache (codepoint offset -> byte offset) for fast string * character looping; 'weak' reference which needs special handling in GC.
48533. bottom of new func
48534. re_ctx->captures at the start of the atom parsed in this loop
48535. * Scan from shortest distance: * - start of string * - end of string * - cache entry (if exists)
48536. refzero_list treated as reachability roots
48537. * Memory calls.
48538. +0.5 is handled by floor, this is on purpose
48539. Duplicate (shadowing) labels are not allowed, except for the empty * labels (which are used as default labels for switch and iteration * statements). * * We could also allow shadowing of non-empty pending labels without any * other issues than breaking the required label shadowing requirements * of the E5 specification, see Section 12.12.
48540. internal temporary value, used for char classes
48541. **comment:** format is too large, abort
 label: code-design
48542. -> [...] key val'
48543. 0x40-0x4f
48544. MPUTOBJ emitted by outer loop
48545. Errors are augmented when they are created, not when they are * thrown or re-thrown. The current error handler, however, runs * just before an error is thrown.
48546. -0 is accepted here as index 0 because ToString(-0) == "0" which is * in canonical form and thus an array index.
48547. be_ctx->offset == length of encoded bitstream
48548. 'in'
48549. Duktape.modLoaded[resolved_id] = module
48550. Caller must ensure 'tv' is indeed a fastint!
48551. stage 1 guarantees
48552. typed for duk_unicode_decode_xutf8()

48553. Macros for creating and checking bitmasks for character encoding. * Bit number is a bit counterintuitive, but minimizes code size.
48554. * Compile input string into an executable function template without * arguments. * * The string is parsed as the "Program" production of EcmaScript E5. * Compilation context can be either global code or eval code (see E5 * Sections 14 and 15.1.2.1). * * Input stack: [...] * Output stack: [...] func_template]

48555. **comment:** XXX: macro? sets both heapdfr and object flags
 label: code-design
48556. [...] varname]
48557. preinitialize lexer state partially
48558. **comment:** XXX: slightly awkward
 label: code-design
48559. Once recursion depth is increased, exit path must decrease * it (though it's OK to abort the fast path).
48560. require
48561. rnd in [lo,hi]
48562. must have catch and/or finally
48563. V

48564. Technically Array.prototype.push() can create an Array with length * longer than 2^32-1, i.e. outside the 32-bit range. The final length * is *not* wrapped to 32 bits in the specification. ** This implementation tracks length with a uint32 because it's much * more practical. ** See: test-bi-array-push-maxlen.js.

48565. DUK_TOK_BXOR

48566. [... re_obj input bc saved_buf res_obj val]

48567. No arr_idx update or limit check

48568. [... varname]

48569. fresh require (argument)

48570. * Resolution loop. At the top of the loop we're expecting a valid * term: '.', '..', or a non-empty identifier not starting with a period.

48571. String object internal value is immutable

48572. At this point 'res' holds the potential expression value. * It can be basically any ivalue here, including a reg-bound * identifier (if code above deems it safe) or a unary/binary * operation. Operations must be resolved to a side effect free * plain value, and the side effects must happen exactly once.

48573. highest capture number emitted so far (used as: ++captures)

48574. arbitrary remove only works with double linked heap, and is only required by * reference counting so far.

48575. **comment:** never reached, but avoids warnings of * potentially unused variables.

label: code-design

48576. new entry: previous value is garbage; set to undefined to share write_value

48577. [... name name]

48578. nargs

48579. each call this helper serves has nargs==2

48580. assertion disabled

48581. * Types are different; various cases for non-strict comparison ** Since comparison is symmetric, we use a "swap trick" to reduce * code size.

48582. Ensure dirty state causes a Status even if never process any * messages. This is expected by the bytecode executor when in * the running state.

48583. * A C * B A * C <- sce ==> B * D D

48584. Sign extend: 0x0000##00 -> 0x##000000 -> sign extend to 0xssssss##

48585. * Virtual properties. ** String and buffer indices are virtual and always enumerable, * 'length' is virtual and non-enumerable. Array and arguments * object props have special behavior but are concrete.

48586. * Case conversion

48587. * Helpers for dealing with the input string

48588. const limits

48589. 0xc0-0xcf

48590. The issues below can be solved with better flags

48591. curr and desc are data

48592. If forced reg, use it as destination. Otherwise try to * use either coerced ispec if it is a temporary. ** When using extraops, avoid reusing arg2 as dest because that * would lead to an LDREG shuffle below. We still can't guarantee * dest != arg2 because we may have a forced_reg.

48593. **comment:** XXX: putvar takes a duk_tval pointer, which is awkward and * should be reworked.

label: code-design

48594. bottom of current func

48595. statement is guaranteed to be terminal (control doesn't flow to next statement)

48596. Use a fast break/continue when possible. A fast break/continue is * just a jump to the LABEL break/continue jump slot, which then jumps * to an appropriate place (for break, going through ENDLABEL correctly). * The peephole optimizer will optimize the jump to a direct one.

48597. * Heap buffer representation. ** Heap allocated user data buffer which is either: * * 1. A fixed size buffer (data follows header statically) * 2. A dynamic size buffer (data pointer follows header) * * The data pointer for a variable size buffer of zero size may be NULL.

48598. Bernstein hash init value is normally 5381; XOR it in in case pointer low bits are 0

48599. **comment:** Allow headroom for calls during error handling (see GH-191). * We allow space for 10 additional recursions, with one extra * for, e.g. a print() call at the deepest level.

label: code-design

48600. http://en.wikipedia.org/wiki/Exponentiation_by_squaring

48601. chain

48602. [... closure]

48603. **comment:** Calling as a non-constructor is not meaningful.

label: code-design

48604. * Voluntary periodic GC (if enabled)

48605. Step types

48606. 'default'

48607. **comment:** XXX: awkward; we assume there is space for this, overwrite * directly instead?

label: code-design

48608. E5 Section steps 7-8

48609. result object is created and discarded; wasteful but saves code space

48610. Gather in little endian

48611. 'fmt'

48612. not caught by anything before entry level; rethrow and let the * final catcher unwind everything

48613. NUL terminate for convenient C access

48614. 'BYTES_PER_ELEMENT'

48615. We don't log or warn about freeing zero refcount objects * because they may happen with finalizer processing.

48616. Don't allow a zero divisor. Restrict both v1 and * v2 to positive values to avoid compiler specific * behavior.

48617. **comment:** XXX: logic for handling character ranges is now incorrect, it will accept * e.g. [\d-z] whereas it should croak from it? SMJS accepts this too, though. *

* Needs a read through and a lot of additional tests.

label: code-design

48618. * Node.js Buffer.prototype.fill()

48619. * Exposed debug macros: debugging disabled

48620. **comment:** No NUL term checks in this debug code.

label: code-design

48621. DUK_BUFOBJ_FLOAT64ARRAY

48622. embed: duk_hobject ptr

48623. getters

48624. DUK_TOK_DELETE

48625. DUK_TOK_BOR

48626. **comment:** pseudotypes, not used in actual longjmps

label: code-design

48627. A -> target reg * BC -> inner function index

48628. refcount code is processing refzero list

48629. [...] retval]; popped below

48630. 0x80...0x8f

48631. Get current EcmaScript time (= UNIX/Posix time, but in milliseconds).

48632. Note: all references inside 'data' need to get their refcounts * upped too. This is the case because refcounts are decreased * through every function referencing 'data' independently.

48633. **comment:** NOTE: level is used only by the debugger and should never be present * for an EcmaScript eval().

label: code-design

48634. entry_top + 1
48635. Set datetime parts from stack arguments, defaulting any missing values. * Day-of-week is not set; it is not required when setting the time value.
48636. * ToBoolean() (E5 Section 9.2)
48637. * Shift to new valstack_bottom.
48638. 0 = don't push current value
48639. "." is not accepted in any format
48640. tmp -> res
48641. [body formals source]
48642. **comment:** XXX: for simple cases like x['y'] an unnecessary LDREG is * emitted for the base value; could avoid it if we knew that * the key expression is safe (e.g. just a single literal).
label: code-design
48643. Other initial bytes.
48644. Similar to String comparison.
48645. Finally, blame the innermost callstack entry which has a *.fileName property.
48646. skip finalizers; queue finalizable objects to heap_allocated
48647. Note that duk_exptrtop() here can clobber any reg above current temp_next, * so any loop variables (e.g. enumerator) must be "preallocated".
48648. * Parse a statement list. ** Handles automatic semicolon insertion and implicit return value. ** Upon entry, 'curr_tok' should contain the first token of the first * statement (parsed in the "allow regexp literal" mode). Upon exit, * 'curr Tok' contains the token following the statement list terminator * (EOF or closing brace).
48649. fixed top level hashtable size (separate chaining)
48650. not present
48651. end marker
48652. handle negative values
48653. invalidated by pushes, so get out of the way
48654. h is NULL for lightfunc
48655. * Error throwing
48656. Non-buffer value is first ToString() coerced, then converted * to a buffer (fixed buffer is used unless a dynamic buffer is * explicitly requested).
48657. **comment:** unused: not accepted in inbound messages
label: code-design
48658. Reuse buffer as is unless buffer has grown large.
48659. Offset is coerced first to signed integer range and then to unsigned. * This ensures we can add a small byte length (1-8) to the offset in * bound checks and not wrap.
48660. internal align target for props allocation, must be 2*n for some n
48661. Set timeval to 'this' from dparts, push the new time value onto the * value stack and return 1 (caller can then tail call us). Expects * the value stack to contain 'this' on the stack top.
48662. target
48663. DUK_FLD_FLOAT
48664. XXX: shared helper fortoFixed(), toExponential(), toPrecision()
48665. **comment:** Run an duk_hobject finalizer. Used for both reference counting * and mark-and-sweep algorithms. Must never throw an error. ** There is no return value. Any return value or error thrown by * the finalizer is ignored (although errors are debug logged). ** Notes: ** - The thread used for calling the finalizer is the same as the * 'thr' argument. This may need to change later. ** - The finalizer thread 'top' assertions are there because it is * critical that strict stack policy is observed (i.e. no cruft * left on the finalizer stack).
label: code-design
48666. -> x in [0, 2**32[
48667. rethrow original error
48668. '\uffThis'
48669. Round up digits to a given position. If position is out-of-bounds, * does nothing. If carry propagates over the first digit, a '1' is * prepended to digits and 'k' will be updated. Return value indicates * whether carry propagated over the first digit. ** Note that nc_ctx->count is NOT updated based on the rounding position * (it is updated only if carry overflows over the first digit and an * extra digit is prepended).
48670. Check if any lookup above had a negative result.
48671. DUK_USE_DEBUGGER_SUPPORT
48672. * For/for-in main variants are: ** 1. for (ExpressionNoIn_opt; Expression_opt; Expression_opt) Statement * 2. for (var VariableDeclarationNoIn; Expression_opt; Expression_opt) Statement * 3. for (LeftHandSideExpression in Expression) Statement * 4. for (var VariableDeclarationNoIn in Expression) Statement ** Parsing these without arbitrary lookahead or backtracking is relatively * tricky but we manage to do so for now. ** See doc/compiler.rst for a detailed discussion of control flow * issues, evaluation order issues, etc.
48673. The specification is a bit vague what to do if the return * value is not a number. Other implementations seem to * tolerate non-numbers but e.g. V8 won't apparently do a * ToNumber().
48674. DUK_TOK_SUB_EQ
48675. * Do a longjmp call, calling the fatal error handler if no * catchpoint exists.
48676. Fixed header info.
48677. nothing to finalize
48678. throw flag irrelevant (false in std alg)
48679. **comment:** This approach is a bit shorter than a straight * if-else-ladder and also a bit faster.
label: code-design
48680. All lightfunc own properties are non-writable and the lightfunc * is considered non-extensible. However, the write may be captured * by an inherited setter which means we can't stop the lookup here.
48681. name
48682. accept string if next char is NUL (otherwise reject)
48683. * "WhiteSpace" production check.
48684. value1 -> yield value, iserror -> error / normal
48685. Number of characters that the atom matches (e.g. 3 for 'abc'), * -1 if atom is complex and number of matched characters either * varies or is not known.
48686. **comment:** 'funcs' is quite rarely used, so no local for it
label: code-design
48687. reachable so pop OK
48688. may be NULL, too
48689. No entry in the catchstack which would actually catch a * throw can refer to the callstack entry being reused. * There *can* be catchstack entries referring to the current * callstack entry as long as they don't catch (e.g. label sites).
48690. * Heap creation and destruction
48691. [... regexp_object escaped_source]
48692. now expected to fit into a 32-bit integer
48693. Duktape.modSearch(resolved_id, fresh_require, exports, module).
48694. bits 0...1: shift
48695. Architecture, OS, and compiler strings
48696. overflow
48697. must fit into duk_small_int_t
48698. 'continue'
48699. 'value'
48700. Non-critical.
48701. **comment:** XXX: tail call: return duk_push_false(ctx)

label: code-design

48702. **comment:** * Sweep garbage and remove marking flags, and move objects with * finalizers to the finalizer work list. ** Objects to be swept need to get their refcounts finalized before * they are swept. In other words, their target object refcounts * need to be decreased. This has to be done before freeing any * objects to avoid decreffing dangling pointers (which may happen * even without bugs, e.g. with reference loops) ** Because strings don't point to other heap objects, similar * finalization is not necessary for strings.

label: code-design

48703. Note: target registers a and a+1 may overlap with DUK__REGCONSTP(b) * and DUK__REGCONSTP(c). Careful here.

48704. DUK_USE_DATE_NOW_GETTIMEOFDAY

48705. **comment:** XXX: what's the safest way of creating a negative zero?

label: code-design

48706. **comment:** XXX: several nice-to-have improvements here: * - Use direct reads avoiding value stack operations * - Avoid triggering getters, indicate getter values to debug client * - If side effects are possible, add error catching

label: code-design

48707. ptr may be NULL

48708. E5 Sections 15.10.2.8, 7.3

48709. Target must be stored so that we can recheck whether or not * keys still exist when we enumerate. This is not done if the * enumeration result comes from a proxy trap as there is no * real object to check against.

48710. * Note: currently no fast path for array index writes. * They won't be possible anyway as strings are immutable.

48711. Important to do a fastint check so that constants are * properly read back as fastints.

48712. remove artificial bump

48713. important to use normalized NaN with 8-byte tagged types

48714. Only objects in heap_allocated may have finalizers. Check that * the object itself has a _Finalizer property (own or inherited) * so that we don't execute finalizers for e.g. Proxy objects.

48715. crossed offsets or zero size

48716. [... escaped_source bytecode]

48717. * E5 Section 11.4.8

48718. * Manipulate valstack so that args are on the current bottom and the * previous caller's 'this' binding (which is the value preceding the * current bottom) is replaced with the new 'this' binding: * * [... this_old | (crud) func this_new arg1 ... argN] * --> [... this_new | arg1 ... argN] * * For tail calling to work properly, the valstack bottom must not grow * here; otherwise crud would accumulate on the valstack.

48719. * ToInteger() (E5 Section 9.4)

48720. invalidates tv

48721. * Fatal error * * There are no fatal error macros at the moment. There are so few call * sites that the fatal error handler is called directly.

48722. num_stack_args

48723. property is virtual: used in duk_propdesc, never stored * (used by e.g. buffer virtual properties)

48724. '\xffVarmap'

48725. Compute (extended) utf-8 length without codepoint encoding validation, * used for string interning. ** NOTE: This algorithm is performance critical, more so than string hashing * in some cases. It is needed when interning a string and needs to scan * every byte of the string with no skipping. Having an ASCII fast path * is useful if possible in the algorithm. The current algorithms were * chosen from several variants, based on x64 gcc -O2 testing. See: * <https://github.com/svaarala/duktape/pull/422> * * NOTE: must match src/dukutil.py:duk_unicode_unvalidated_utf8_length().

48726. Only allow Duktape.Buffer when support disabled.

48727. * Digit generation loop. * * Different termination conditions: * * 1. Free format output. Terminate when shortest accurate * representation found. * * 2. Fixed format output, with specific number of digits. * Ignore termination conditions, terminate when digits * generated. Caller requests an extra digit and rounds. * * 3. Fixed format output, with a specific absolute cut-off * position (e.g. 10 digits after decimal point). Note * that we always generate at least one digit, even if * the digit is below the cut-off point already.

48728. number of digits changed

48729. **comment:** re-lookup to update curr.flags * XXX: would be faster to update directly

label: code-design

48730. * Heap Buffer object representation. Used for all Buffer variants.

48731. **comment:** Note: reuse variables

label: code-design

48732. _Varmap

48733. = 1 + arg count

48734. * Tables generated with src/gennumdigits.py. * * duk__str2num_digits_for_radix indicates, for each radix, how many input * digits should be considered significant for string-to-number conversion. * The input is also padded to this many digits to give the Dragon4 * conversion enough (apparent) precision to work with. * * duk__str2num_exp_limits indicates, for each radix, the radix-specific * minimum/maximum exponent values (for a Dragon4 integer mantissa) * below and above which the number is guaranteed to underflow to zero * or overflow to Infinity. This allows parsing to keep bigint values * bounded.

48735. 'case'

48736. num_pairs and temp_start reset at top of outer loop

48737. Because size > 0, NULL check is correct

48738. 17: getUTCHours

48739. [... enum] -> [... next_key]

48740. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2017 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

48741. Encode to CESU-8; 'out' must have space for at least * DUK_UNICODE_MAX_CESU8_LENGTH bytes; codepoints above U+10FFFF * will encode to garbage but won't overwrite the output buffer.

48742. if density < L, abandon array part, L = 3-bit fixed point, e.g. 2 -> 2/8 = 25%

48743. * Finalize stack

48744. Only a reg fits into 'A' so coerce 'res' into a register * for PUTVAR. ** XXX: here the current A/B/C split is suboptimal: we could * just use 9 bits for reg_res (and support constants) and 17 * instead of 18 bits for the varname const index.

48745. * Helper for setting up var_env and lex_env of an activation, * assuming it does NOT have the DUK_HOBJECT_FLAG_NEWENV flag.

48746. no need for 'caller' post-check, because 'key' must be an array index

48747. [arg undefined]

48748. "

48749. * String-to-number conversion

48750. We can't shortcut zero here if it goes through special formatting * (such as forced exponential notation).

48751. 0x10...0x1f

48752. Main operation

48753. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT8CLAMPED * Note: INT8 is -not- copy compatible, e.g. -1 would coerce to 0x00.

48754. invalidated

48755. DUK_USE_CPP_EXCEPTIONS

48756. max(incl)

48757. * Arguments objects have exotic [[DefineOwnProperty]] which updates * the internal 'map' of arguments for writes to currently mapped * arguments. More concretely, writes to mapped arguments generate * a write to a bound variable. * * The [[Put]] algorithm invokes [[DefineOwnProperty]] for existing * data properties and new properties, but not for existing accessors. * Hence, in E5 Section 10.6 ([[DefinedOwnProperty]] algorithm), we * have a Desc with 'Value' (and possibly other properties too), and * we end up in step 5.b.i.

48758. A top-level assignment is e.g. "x = y". For these it's safe * to use the RHS as-is as the expression value, even if the RHS * is a reg-bound identifier. The RHS ('res') is right associative * so it has consumed all other assignment level operations; the * only relevant lower binding power construct is comma operator * which will ignore the expression value provided here. Usually * the top level assignment expression value is ignored, but it * is relevant for e.g. eval code.

48759. **comment:** XXX: use helper for size optimization
label: code-design

48760. 'ignoreCase'

48761. Note: env_thr != thr is quite possible and normal, so careful * with what thread is used for valstack lookup.

48762. frozen and sealed

48763. tv -> value just before prev top value; must relookup

48764. * Found existing inherited plain property. * Do an access control check, and if OK, write * new property to 'orig'.

48765. initial estimate for ASCII only codepoints

48766. Parser separator masks.

48767. just in case callback is broken and won't write 'x'

48768. **comment:** NEXTENUM needs a jump slot right after the main opcode. * We need the code emitter to reserve the slot: if there's * target shuffling, the target shuffle opcodes must happen * after the jump slot (for NEXTENUM the shuffle opcodes are * not needed if the enum is finished).
label: code-design

48769. then to a register

48770. retval to result[i]

48771. Sanity limit on max number of properties (allocated, not necessarily used). * This is somewhat arbitrary, but if we're close to 2**32 properties some * algorithms will fail (e.g. hash size selection, next prime selection). * Also, we use negative array/entry table indices to indicate 'not found', * so anything above 0x80000000 will cause trouble now.

48772. expect EOF instead of }

48773. resizing parameters

48774. value from target

48775. this is especially critical to avoid another write call in detach1()

48776. **comment:** Stripping the filename might be a good idea * ("foo/bar/quux.js" -> logger name "quux"), * but now used verbatim.
label: code-design

48777. return the argument object

48778. new buffer of specified size

48779. * Conversion helpers

48780. %x; only 16 bits are guaranteed

48781. mix-in value for computing string hashes; should be reasonably unpredictable

48782. 'debug'

48783. keep default instance

48784. Make buffer compact, matching current written size.

48785. explicit semi follows

48786. * Packed 8-byte representation

48787. **comment:** XXX: currently no handling for non-allowed identifier characters, * e.g. a '{' in the function name.
label: code-design

48788. No support for lvalues returned from new or function call expressions. * However, these must NOT cause compile-time SyntaxErrors, but run-time * ReferenceErrors. Both left and right sides of the assignment must be * evaluated before throwing a ReferenceError. For instance: * * f() = g(); * * must result in f() being evaluated, then g() being evaluated, and * finally, a ReferenceError being thrown. See E5 Section 11.13.1.

48789. **comment:** XXX: awkward helpers
label: code-design

48790. advance, expecting current token to be a specific token; parse next token in regexp context

48791. stack prepped for func call: [... trap handler]

48792. '{"_nan":true}'

48793. tagged null pointers should never occur

48794. * Heap header definition and assorted macros, including ref counting. * Access all fields through the accessor macros.

48795. * Finally, longjmp

48796. e.g. DUK_OP_POSTINCP

48797. Allow leading minus sign

48798. copy to buffer with spare to avoid Valgrind gripes from strftime

48799. throw

48800. 4: toLocaleDateString

48801. eat 'with'

48802. [... this func]

48803. setters

48804. step 4.c

48805. Note: cannot assume that a non-zero return value of signbit() would * always be the same -- hence cannot (portably) use something like: * * signbit(x) == signbit(y)

48806. check func supports [[HasInstance]] (this is checked for every function * in the bound chain, including the final one)

48807. There is no need to NUL delimit the sscanf() call: trailing garbage is * ignored and there is always a NUL terminator which will force an error * if no error is encountered before it. It's possible that the scan * would scan further than between [js_ctx->p,p[though and we'd advance * by less than the scanned value. * * Because pointers are platform specific, a failure to scan a pointer * results in a null pointer which is a better placeholder than a missing * value or an error.

48808. **comment:** XXX: relocate
label: code-design

48809. match fail

48810. * Make a value copy of the input val. This ensures that * side effects cannot invalidate the pointer.

48811. DUK_USE_DATE_TZO_GMTIME

48812. Treat like debugger statement: nop

48813. Get valstack index for the func argument or throw if insane stack.

48814. **comment:** 'd3' is never NaN, so no need to normalize
label: code-design

48815. XXX: push_uint_string / push_u32_string

48816. return value from Duktape.Thread.resume()

48817. For the case n==1 Node.js doesn't seem to type check * the sole member but we do it before returning it. * For this case only the original buffer object is * returned (not a copy).

48818. end of loop (careful with len==0)

48819. Result is always fastint compatible.

48820. Handle a BREAK/CONTINUE opcode. Avoid using longjmp() for BREAK/CONTINUE * handling because it has a measurable performance impact in ordinary * environments and an extreme impact in Emscripten (GH-342).

48821. DUK_TOK_TRY

48822. Optional UTC conversion.

48823. 6
48824. [requested_id require require.id resolved_id last_comp Duktape Duktape.modLoaded undefined fresh_require exports module]
48825. provideThis=false
48826. * Self test main
48827. prototype is lex_env before catcher created
48828. ToInteger() coercion; NaN -> 0, infinities are clamped to 0 and 10
48829. Keep throwing an error whenever we get here. The unusual values * are set this way because no instruction is ever executed, we just * throw an error until all try/catch/finally and other catchpoints * have been exhausted. Duktape/C code gets control at each protected * call but whenever it enters back into Duktape the RangeError gets * raised. User exec timeout check must consistently indicate a timeout * until we've fully bubbled out of Duktape.
48830. tc1 = true, tc2 = false
48831. no need to handle thread state book-keeping here
48832. Not found. Empty string case is handled specially above.
48833. 4 low bits
48834. **comment:** XXX: option to pretend property doesn't exist if sanity limit is * hit might be useful.
label: code-design
48835. Use explicit steps in computation to try to ensure that * computation happens with intermediate results coerced to * double values (instead of using something more accurate). * E.g. E5.1 Section 15.9.1.11 requires use of IEEE 754 * rules (= Ecmascript 'l' and '**' operators). ** Without 'volatile' even this approach fails on some platform * and compiler combinations. For instance, gcc 4.8.1 on Ubuntu * 64-bit, with -m32 and without -std=c99, test-bi-date-canceling.js * would fail because of some optimizations when computing tmp_time * (MakeTime below). Adding 'volatile' to tmp_time solved this * particular problem (annoyingly, also adding debug prints or * running the executable under valgrind hides it).
48836. **comment:** Smaller heapdr than for other objects, because strings are held * in string intern table which requires no link pointers. Much of * the 32-bit flags field is unused by flags, so we can stuff a 16-bit * field in there.
label: code-design
48837. **comment:** Delete semantics are a bit tricky. The description in E5 specification * is kind of confusing, because it distinguishes between resolvability of * a reference (which is only known at runtime) seemingly at compile time * (= SyntaxError throwing).
label: code-design
48838. [...] -> [...]]
48839. * Augment an error being created using Duktape specific properties * like _Tracedata or .fileName/.lineNumber.
48840. misc constants and helper macros
48841. zero-based day
48842. 28: setMinutes
48843. non-object base, no offending virtual property
48844. Fixed seed value used with ROM strings.
48845. duk_parse_stmts() expects curr_tok to be set; parse in "allow regexp literal" mode with current strictness
48846. E5 Sections 11.8.1, 11.8.5; x < y
48847. may be < thr->catchstack initially
48848. complex, multicharacter conversion
48849. Not enough data to provide a full window, so "scroll" window to * start of buffer and fill up the rest.
48850. E5.1 Section 15.1.3.1: uriReserved + '#'
48851. **comment:** XXX: should there be an error or an automatic detach if * already attached?
label: code-design
48852. For user code this could just return 1 (strict) always * because all Duktape/C functions are considered strict, * and strict is also the default when nothing is running. * However, Duktape may call this function internally when * the current activation is an Ecmascript function, so * this cannot be replaced by a 'return 1' without fixing * the internal call sites.
48853. "" was eaten by caller
48854. expression is left-hand-side compatible
48855. string is a reserved word (strict)
48856. **comment:** XXX: Output type could be encoded into native function 'magic' value to * save space. For setters the stridx could be encoded into 'magic'.
label: code-design
48857. **comment:** * Manipulation of thread stacks (valstack, callstack, catchstack). ** Ideally unwinding of stacks should have no side effects, which would * then favor separate unwinding and shrink check primitives for each * stack type. A shrink check may realloc and thus have side effects. ** However, currently callstack unwinding itself has side effects, as it * needs to DECREF multiple objects, close environment records, etc. * Stacks must thus be unwound in the correct order by the caller. ** (XXX: This should be probably reworked so that there is a shared * unwind primitive which handles all stacks as requested, and knows * the proper order for unwinding.) * Valstack entries above 'top' are always kept initialized to * "undefined unused". Callstack and catchstack entries above 'top' * are not zeroed and are left as garbage. ** Value stack handling is mostly a part of the API implementation.
label: code-design
48858. Step 9: copy elements-to-be-deleted into the result array
48859. 0x30-0x3f
48860. 2 bits for heap type
48861. setjmp catchpoint setup
48862. **comment:** not sure whether or not needed; Thursday
label: requirement
48863. emergency mode: try extra hard
48864. Set object 'length'.
48865. byte index limit for element access, exclusive
48866. We could rely on max temp/const checks: if they don't exceed BC * limit, nothing here can either (just asserts would be enough). * Currently we check for the limits, which provides additional * protection against creating invalid bytecode due to compiler * bugs.
48867. require initializer for var/const
48868. DUK_TOK_FALSE
48869. fall-through control flow patchup; note that pc_prevstmt may be * < 0 (i.e. no case clauses), in which case this is a no-op.
48870. [...] eval "eval" eval_input level]
48871. * ToObject() (E5 Section 9.9) * ==> implemented in the API.
48872. DUK_USE_MARKANDSWEEP_FINALIZER_TORTURE
48873. **comment:** * Handle integers in 32-bit range (that is, [-(2**32-1),2**32-1]) * specially, as they're very likely for embedded programs. This * is now done for all radix values. We must be careful not to use * the fast path when special formatting (e.g. forced exponential) * is in force. ** XXX: could save space by supporting radix 10 only and using * sprintf "%lu" for the fast path and for exponent formatting.
label: code-design
48874. !(p < r)
48875. Maximum exponent value when parsing numbers. This is not strictly * compliant as there should be no upper limit, but as we parse the * exponent without a bigint, impose some limit.
48876. num_stack_args
48877. **comment:** Executor interrupt counter check, used to implement breakpoints, * debugging interface, execution timeouts, etc. The counter is heap * specific but is maintained in the current thread to make the check * as fast as possible. The counter is copied back to the heap struct * whenever a thread switch occurs by the DUK_HEAP_SWITCH_THREAD() macro.
label: code-design
48878. complete 'sub atom'
48879. * Intern key via the valstack to ensure reachability behaves * properly. We must avoid longjmp's here so use non-checked * primitives. ** Note: duk_check_stack() potentially reallocs the valstack, * invalidating any duk_tval pointers to valstack. Callers * must be careful.

48880. round out to 8 bytes
48881. for convenience
48882. buffer limits
48883. Handle TypedArray vs. Node.js Buffer arg differences
48884. DUK_TOK_WHILE
48885. offset/value order different from Node.js
48886. * Other cases (function declaration, anonymous function expression, * strict direct eval code). The "outer" environment will be whatever * the caller gave us.
48887. * Setup environment record properties based on the template and * its flags. * * If DUK_HOBJECT_HAS_NEWENV(fun_temp) is true, the environment * records represent identifiers "outside" the function; the * "inner" environment records are created on demand. Otherwise, * the environment records are those that will be directly used * (e.g. for declarations). * * _Lexenv is always set; _Varenv defaults to _Lexenv if missing, * so _Varenv is only set if _Lexenv != _Varenv. * * This is relatively complex, see doc/identifier-handling.rst.
48888. continue matched this label -- we can only continue if this is the empty * label, for which duplication is allowed, and thus there is hope of * finding a match deeper in the label stack.
48889. Plus sign must be accepted for positive exponents * (e.g. '1.5e+2'). This clause catches NULs.
48890. DUK_TOK_VOID
48891. DUK_USE_PC2LINE
48892. DUK_HOBJECT_FLAG_STRICT varies
48893. [... errval errhandler]
48894. **comment:** XXX: Valstack, callstack, and catchstack are currently assumed * to have non-NULL pointers. Relaxing this would not lead to big * benefits (except perhaps for terminated threads).
 label: code-design
48895. serialize the wrapper with empty string key
48896. * Free a heap object. * * Free heap object and its internal (non-heap) pointers. Assumes that * caller has removed the object from heap allocated list or the string * intern table, and any weak references (which strings may have) have * been already dealt with.
48897. BITWISE EXPRESSIONS
48898. **comment:** * Copy some internal properties directly * * The properties will be writable and configurable, but not enumerable.
 label: code-design
48899. Valstack resize flags
48900. **comment:** XXX: Hex encoded, length limited buffer summary here?
 label: code-design
48901. finish up pending jump and split for last alternative
48902. nop
48903. * Local declarations.
48904. stack[0] = object (this) * stack[1] = ToUint32(length) * stack[2] = elem at index 0 (retval)
48905. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers). A prototype loop must not cause * an error to be thrown here; duk_hobject_hasprop_raw() will ignore a * prototype loop silently and indicate that the property doesn't exist.
48906. **comment:** XXX: coerce to regs? it might be better for enumeration use, where the * same PROP ivalue is used multiple times. Or perhaps coerce PROP further * there?
 label: code-design
48907. x > y --> y < x
48908. * Found existing accessor property (own or inherited). * Call setter with 'this' set to orig, and value as the only argument. * Setter calls are OK even for ROM objects. * * Note: no exotic arguments object behavior, because [[Put]] never * calls [[DefineOwnProperty]] (E5 Section 8.12.5, step 5.b).
48909. resume: [... initial_func] (currently actually: [initial_func])
48910. Lookup 'key' from arguments internal 'map', and leave replacement value * on stack top if mapped (and return non-zero). * Used in E5 Section 10.6 algorithm for [[GetOwnProperty]] (used by [[Get]]).
48911. proto can also be NULL here (allowed explicitly)
48912. * split()
48913. Validate that the whole slice [0,length[is contained in the underlying * buffer. Caller must ensure 'buf' != NULL.
48914. could also just pop?
48915. Parser part indices.
48916. Compiler SyntaxError (or other error) gets the primary blame. * Currently no flag to prevent blaming.
48917. We don't check the zero padding bytes here right now * (that they're actually zero). This seems to be common * behavior for base-64 decoders.
48918. An object may be in heap_allocated list with a zero * refcount if it has just been finalized and is waiting * to be collected by the next cycle.
48919. * Local prototypes
48920. low-level property functions
48921. helpers for defineProperty() and defineProperties()
48922. w/o refcounts
48923. Skip dvalues to EOM.
48924. all native functions have NEWENV
48925. +1, right after inserted jump
48926. only accept lowercase 'utf8' now.
48927. **comment:** XXX: duplicates should be eliminated here
 label: code-design
48928. **comment:** XXX: block removal API primitive
 label: code-design
48929. LOGICAL EXPRESSIONS
48930. %lf, %ld etc
48931. * Switch is pretty complicated because of several conflicting concerns: * * - Want to generate code without an intermediate representation, * i.e., in one go * * - Case selectors are expressions, not values, and may thus e.g. throw * exceptions (which causes evaluation order concerns) * * - Evaluation semantics of case selectors and default clause need to be * carefully implemented to provide correct behavior even with case value * side effects * * - Fall through case and default clauses; avoiding dead JUMPs if case * ends with an unconditional jump (a break or a continue) * * - The same case value may occur multiple times, but evaluation rules * only process the first match before switching to a "propagation" mode * where case values are no longer evaluated * * See E5 Section 12.11. Also see doc/compiler.rst for compilation * discussion.
48932. 8: getFullYear
48933. * Debug connection peek and flush primitives
48934. 'do'
48935. duk_xdef_prop() will define an own property without any array * special behaviors. We'll need to set the array length explicitly * in the end. For arrays with elisions, the compiler will emit an * explicit SETALEN which will update the length.
48936. exponential notation forced
48937. NB: 'val' may be invalidated here because put_value may realloc valstack, * caller beware.
48938. Here we'd have the option to normalize -0 to +0.
48939. **comment:** XXX: share helper from lexer; duk_lexer.c / hexval().
 label: code-design
48940. **comment:** * Create an internal enumerator object E, which has its keys ordered * to match desired enumeration ordering. Also initialize internal control * properties for enumeration. * * Note: if an array was used to hold enumeration keys instead, an array * scan would be needed to eliminate duplicates found in the prototype chain.
 label: code-design

48941. **comment:** This is essentially the 'scale' algorithm, with recursion removed. * Note that 'k' is either correct immediately, or will move in one * direction in the loop. There's no need to do the low/high checks * on every round (like the Scheme algorithm does). ** The scheme algorithm finds 'k' and updates 's' simultaneously, * while the logical algorithm finds 'k' with 's' having its initial * value, after which 's' is updated separately (see the Burger-Dybvig * paper, Section 3.1, steps 2 and 3). ** The case where $m+ == m-$ (almost always) is optimized for, because * it reduces the bigint operations considerably and almost always * applies. The scale loop only needs to work with $m+$, so this works.
- label:** code-design
48942. A -> flags * BC -> regCatch; base register for two registers used both during * trycatch setup and when catch is triggered ** If DUK_BC_TRYCATCH_FLAG_CATCH_BINDING set: * regCatch + 0: catch binding variable name (string). * Automatic declarative environment is established for * the duration of the 'catch' clause. ** If DUK_BC_TRYCATCH_FLAG_WITH_BINDING set: * regCatch + 0: with 'target value', which is coerced to * an object and then used as a bindind object for an * environment record. The binding is initialized here, for * the 'try' clause. ** Note that a TRYCATCH generated for a 'with' statement has no * catch or finally parts.
48943. default: false
48944. **comment:** fill new_h with u32 0xff = UNUSED
- label:** code-design
48945. A non-extensible object cannot gain any more properties, * so this is a good time to compact.
48946. if direct eval, calling activation must exist
48947. **comment:** XXX: because of the final check below (that the literal is not * followed by a digit), this could maybe be simplified, if we bail * out early from a leading zero (and if there are no periods etc). * Maybe too complex.
- label:** code-design
48948. * Create escaped RegExp source (E5 Section 15.10.3). ** The current approach is to special case the empty RegExp * (" -> '(?:')") and otherwise replace unescaped '/' characters * with '\' regardless of where they occur in the regexp. ** Note that normalization does not seem to be necessary for * RegExp literals (e.g. '/foo/') because to be acceptable as * a RegExp literal, the text between forward slashes must * already match the escaping requirements (e.g. must not contain * unescaped forward slashes or be empty). Escaping IS needed * for expressions like 'new RegExp("...","")' however. * Currently, we re-escape in either case. ** Also note that we process the source here in UTF-8 encoded * form. This is correct, because any non-ASCII characters are * passed through without change.
48949. Value stack is used to ensure reachability of constants and * inner functions being loaded. Require enough space to handle * large functions correctly.
48950. **comment:** Automatic defaulting of logger name from caller. This would * work poorly with tail calls, but constructor calls are currently * never tail calls, so tail calls are not an issue now.
- label:** code-design
48951. Fast check for extending array: check whether or not a slow density check is required.
48952. **comment:** XXX: using duk_put_prop_index() would cause obscure error cases when Array.prototype * has write-protected array index named properties. This was seen as DoubleErrors * in e.g. some test262 test cases. Using duk_xdef_prop_index() is better but heavier. * The best fix is to fill in the tracedata directly into the array part. There are * no side effect concerns if the array part is allocated directly and only INCREFS * happen after that.
- label:** code-design
48953. filename may be NULL in which case file/line is not recorded
48954. unsigned
48955. **comment:** XXX: function flag to make this automatic?
- label:** requirement
48956. Must have array part and no conflicting exotic behaviors. * Doesn't need to have array special behavior, e.g. Arguments * object has array part.
48957. Breakpoint entries above the used area are left as garbage.
48958. allow_source_elem
48959. JSON parsing code is allowed to read [p_start,p_end]: p_end is * valid and points to the string NUL terminator (which is always * guaranteed for duk_hstrings).
48960. fall through if overflow etc
48961. inner function numbering
48962. **comment:** XXX: could write output in chunks with fewer ensure calls, * but relative benefit would be small here.
- label:** code-design
48963. Assumes that saved[0] and saved[1] are always * set by regexp bytecode (if not, char_end_offset * will be zero). Also assumes clen reflects the * correct char length.
48964. **comment:** XXX: V8 throws a TypeError for negative values. Would it * be more useful to interpret negative offsets here from the * end of the buffer too?
- label:** code-design
48965. function needs shuffle registers
48966. The compiler should never emit DUK_OP_REGEXP if there is no * regexp support.
48967. invalid padding
48968. Note: here we must be wary of the fact that a pointer may be * valid and be a NULL.
48969. **comment:** Here we can choose either to end parsing and ignore * whatever follows, or to continue parsing in case * multiple (possibly padded) base64 strings have been * concatenated. Currently, keep on parsing.
- label:** code-design
48970. [requested_id require require.id resolved_id last_comp]
48971. advance to get one step of lookup
48972. Write signed 32-bit integer.
48973. * Canonicalize() abstract operation needed for canonicalization of individual * codepoints during regexp compilation and execution, see E5 Section 15.10.2.8. * Note that codepoints are canonicalized one character at a time, so no context * specific rules can apply. Locale specific rules can apply, though.
48974. This should not be necessary because no-one should tamper with the * regexp bytecode, but is prudent to avoid potential segfaults if that * were to happen for some reason.
48975. Needs lookahead
48976. * Churn refzero_list until empty
48977. Note: 'q_instr' is still used below
48978. duk_unicode_caseconv_uc[]
48979. **comment:** * Make a copy of tv_obj, tv_key, and tv_val to avoid any issues of * them being invalidated by a valstack resize. ** XXX: this is now an overkill for many fast paths. Rework this * to be faster (although switching to a valstack discipline might * be a better solution overall).
- label:** code-design
48980. failed, resize and try again
48981. extend with undefined
48982. **comment:** * Traceback handling ** The unified helper decodes the traceback and produces various requested * outputs. It should be optimized for size, and may leave garbage on stack, * only the topmost return value matters. For instance, traceback separator * and decoded strings are pushed even when looking for filename only. * * NOTE: although _Tracedata is an internal property, user code can currently * write to the array (or replace it with something other than an array). * The code below must tolerate arbitrary _Tracedata. It can throw errors * etc, but cannot cause a segfault or memory unsafe behavior.
- label:** code-design
48983. Error; error value is in heap->lj.value1.
48984. -> [... key val toJSON val key]
48985. callstack
48986. read 3 bytes into 't', padded by zero
48987. XXX: remove this feature entirely? it would only matter for * emergency GC. Disable for lowest memory builds.
48988. * Computed values (e.g. INFINITY)
48989. DUK_TOK_DIV_EQ
48990. already set
48991. input 'd' in Windows UTC, 100ns units
48992. Keep current size
48993. 50x heap size

48994. Dynamic buffer with 'curr_alloc' pointing to a dynamic area allocated using * heap allocation primitives. Also used for external buffers when low memory * options are not used.

48995. [name this]

48996. [... holder name val enum obj_key new_elem]

48997. patched later

48998. [... regexp_object]

48999. **comment:** * Note: the description in E5 Section 15.10.2.6 has a typo, it * contains 'A' twice and lacks 'a'; the intent is [0-9a-zA-Z_].
label: documentation

49000. This was the last term, and q_last was * updated to match this term at loop top.

49001. x <- b^y; use t1 and t2 as temps

49002. prev_token.start_offset points to the closing brace here; when skipping * we're going to reparse the closing brace to ensure semicolon insertion * etc work as expected.

49003. more initializers

49004. ref.holder is global object, holder is the object with the * conflicting property.

49005. essentially tracks digit position of lowest 'f' digit

49006. **comment:** Prevent any side effects on the string table and the caller provided * str/blen arguments while interning is in progress. For example, if * the caller provided str/blen from a dynamic buffer, a finalizer might * resize that dynamic buffer, invalidating the call arguments.
label: code-design

49007. binding power "levels" (see doc/compiler.rst)

49008. Enter message processing loop for sending Status notifys and * to finish a pending detach.

49009. result reg

49010. strings are only tracked by stringtable

49011. to avoid relookups

49012. abandoning requires a props allocation resize and * 'rechecks' the valstack, invalidating any existing * valstack value pointers!

49013. If object has a . toJSON() property, we can't be certain * that it wouldn't mutate any value arbitrarily, so bail * out of the fast path. * * If an object is a Proxy we also can't avoid side effects * so abandon.

49014. Reviving an object using a heap pointer is a dangerous API * operation: if the application doesn't guarantee that the * pointer target is always reachable, difficult-to-diagnose * problems may ensue. Try to validate the 'ptr' argument to * the extent possible.

49015. '.'

49016. **comment:** Unlike in duk_hex_encode() 'dst' is not necessarily aligned by 2. * For platforms where unaligned accesses are not allowed, shift 'dst' * ahead by 1 byte to get alignment and then DUK_MEMMOVE() the result * in place. The faster encoding loop makes up the difference. * There's always space for one extra byte because a terminator always * follows the hex data and that's been accounted for by the caller.
label: code-design

49017. E5 Section 7.3, treat the following as newlines: * LF * CR [not followed by LF] * LS * PS * * For CR LF, CR is ignored if it is followed by LF, and the LF will bump * the line number.

49018. **comment:** * 'nregs' registers are allocated on function entry, at most 'nargs' * are initialized to arguments, and the rest to undefined. Arguments * above 'nregs' are not mapped to registers. All registers in the * active stack range must be initialized because they are GC reachable. * 'nargs' is needed so that if the function is given more than 'nargs' * arguments, the additional arguments do not 'clobber' registers * beyond 'nregs' which must be consistently initialized to undefined. * * Usually there is no need to know which registers are mapped to * local variables. Registers may be allocated to variable in any * way (even including gaps). However, a register-variable mapping * must be the same for the duration of the function execution and * the register cannot be used for anything else. * * When looking up variables by name, the '_Varmap' map is used. * When an activation closes, registers mapped to arguments are * copied into the environment record based on the same map. The * reverse map (from register to variable) is not currently needed * at run time, except for debugging, so it is not maintained.
label: code-design

49019. Slice starting point is beyond current length.

49020. ... target is extensible

49021. duk_unicode_support.c

49022. **comment:** XXX: easier check with less code?
label: code-design

49023. Used when precision is undefined; also used for NaN (-> "NaN"), * and +/- infinity (-> "Infinity", "-Infinity").

49024. x <- x * y, use t as temp

49025. [... v1 v2 str] -> [... str v2]

49026. Zero 'count' is also allowed to make call sites easier.

49027. **comment:** XXX: size optimize
label: code-design

49028. != 0 => post-update for array 'length' (used when key is an array index)

49029. **comment:** has inner functions which may slow access (XXX: this can be optimized by looking at the inner functions)
label: code-design

49030. **comment:** XXX: need to determine LHS type, and check that it is LHS compatible
label: code-design

49031. false

49032. * Built-in strings

49033. * Clear (reachable) flags of finalize_list * * We could mostly do in the sweep phase when we move objects from the * heap into the finalize_list. However, if a finalizer run is skipped * during a mark-and-sweep, the objects on the finalize_list will be marked * reachable during the next mark-and-sweep. Since they're already on the * finalize_list, no-one will be clearing their REACHABLE flag so we do it * here. (This now overlaps with the sweep handling in a harmless way.)

49034. * Variant 3

49035. * On second pass, skip the function.

49036. OK for NULL.

49037. Note: key issue here is to re-lookup the base pointer on every attempt. * The pointer being reallocated may change after every mark-and-sweep.

49038. -> [... key val toJSON val]

49039. **comment:** XXX: It would be nice to build the string directly but ToUint16() * coercion is needed so a generic helper would not be very * helpful (perhaps coerce the value stack first here and then * build a string from a duk_tval number sequence in one go?).
label: code-design

49040. Note: getprop may invoke any getter and invalidate any * duk_tval pointers, so this must be done first.

49041. [obj trap_result res_arr propname]

49042. split1: prefer direct execution (no jump)

49043. in resumer's context

49044. Conceptually we'd extract the plain underlying buffer * or its slice and then do a type mask check below to * see if we should reject it. Do the mask check here * instead to avoid making a copy of the buffer slice.

49045. 'try'

49046. -> [in_val]

49047. **comment:** Currently all built-in native functions are strict. * This doesn't matter for many functions, but e.g. * String.prototype.charAt (and other string functions) * rely on being strict so that their 'this' binding is * not automatically coerced.
label: code-design

49048. **comment:** * Helper for handling an Ecmascript-to-Ecmascript call or an Ecmascript * function (initial) Duktape.Thread.resume(). * * Compared to normal calls handled by duk_handle_call(), there are a * bunch of differences: * * - the call is never protected * - there is no C recursion depth increase (hence an "ignore recursion * limit" flag is not applicable) * - instead of making the call, this helper just performs the thread * setup and returns; the bytecode executor then restarts execution * internally * - ecmascript functions are never 'vararg' functions (they access * varargs through the 'arguments' object) * * The callstack of the target contains an earlier Ecmascript call in case * of an Ecmascript-to-Ecmascript call (whose idx_retval is updated), or * is empty in case of an initial

Duktape.Thread.resume(). * * The first thing to do here is to figure out whether an ecma-to-ecma * call is actually possible. It's not always the case if the target is * a bound function; the final function may be native. In that case, * return an error so caller can fall back to a normal call path.

label: code-design

49049. Note: 0xff != DUK_BC_B_MAX

49050. * Unions and structs for self tests

49051. Note: masking is done for 'i' to deal with negative numbers correctly

49052. * For property layout 1, tweak e_size to ensure that the whole entry * part (key + val + flags) is a suitable multiple for alignment * (platform specific). * * Property layout 2 does not require this tweaking and is preferred * on low RAM platforms requiring alignment.

49053. don't run finalizers; queue finalizable objects back to heap_allocated

49054. val1 = count

49055. * Function declaration, function expression, or (non-standard) * function statement. * * The E5 specification only allows function declarations at * the top level (in "source elements"). An ExpressionStatement * is explicitly not allowed to begin with a "function" keyword * (E5 Section 12.4). Hence any non-error semantics for such * non-top-level statements are non-standard. Duktape semantics * for function statements are modelled after V8, see * test-dev-func-decl-outside-top.js.

49056. Get minimum array part growth for a certain size.

49057. **comment:** XXX: very slow - better to bulk allocate a gap, and copy * from args_array directly (we know it has a compact array * part, etc).

label: code-design

49058. 'enumerate'

49059. DUK_TOK_RSHIFT

49060. roughly 256 bytes

49061. **comment:** For now, just use duk_safe_call() to wrap duk_new(). We can't * simply use a protected duk_handle_call() because there's post * processing which might throw. It should be possible to ensure * the post processing never throws (except in internal errors and * out of memory etc which are always allowed) and then remove this * wrapper.

label: code-design

49062. object is now reachable

49063. flags: "im"

49064. Copy trap result keys into the enumerator object.

49065. * Mark-and-sweep flags * * These are separate from heap level flags now but could be merged. * The heap structure only contains a 'base mark-and-sweep flags' * field and the GC caller can impose further flags.

49066. Success path.

49067. * If entering a 'catch' block which requires an automatic * catch variable binding, create the lexical environment. * * The binding is mutable (= writable) but not deletable. * Step 4 for the catch production in E5 Section 12.14; * no value is given for CreateMutableBinding 'D' argument, * which implies the binding is not deletable.

49068. t1 <- (* r B)

49069. DUK_BUFOBJ_UINT8ARRAY

49070. Accept any duk_hbufferobject, though we're only normally * called for Duktape.Buffer values.

49071. Lenient: allow function declarations outside top level in * non-strict mode but reject them in strict mode.

49072. assertions: env must be closed in the same thread as where it runs

49073. EOF (-1) will be cast to an unsigned value first * and then re-cast for the switch. In any case, it * will match the default case (syntax error).

49074. refzero not running -> must be empty

49075. **comment:** XXX: differentiate null pointer from empty string?

label: code-design

49076. Normal object which doesn't get automatically coerced to a * primitive value. Functions are checked for specially. The * primitive value coercions for Number, String, Pointer, and * Boolean can't result in functions so suffices to check here.

49077. DUK_USE_MARK_AND_SWEEP

49078. * Coercion and fast path processing

49079. y == +Infinity

49080. caller required to know

49081. * Function name (if any) * * We don't check for prohibited names here, because we don't * yet know whether the function will be strict. Function body * parsing handles this retroactively. * * For function expressions and declarations function name must * be an Identifier (excludes reserved words). For setter/getter * it is a PropertyName which allows reserved words and also * strings and numbers (e.g. "{ get 1) { ... } }").

49082. **comment:** XXX: There are no format strings in duk_config.h yet, could add * one for formatting duk_int64_t. For now, assumes "%lld" and that * "long long" type exists. Could also rely on C99 directly but that * won't work for older MSVC.

label: code-design

49083. toLocaleString'

49084. * Manipulate value stack so that exactly 'num_stack_rets' return * values are at 'idx_rebase' in every case, assuming there are * 'rc' return values on top of stack. * * This is a bit tricky, because the called C function operates in * the same activation record and may have e.g. popped the stack * empty (below idx_rebase).

49085. Exotic behaviors are only enabled for arguments objects * which have a parameter map (see E5 Section 10.6 main * algorithm, step 12). * * In particular, a non-strict arguments object with no * mapped formals does *NOT* get exotic behavior, even * for e.g. "caller" property. This seems counterintuitive * but seems to be the case.

49086. buffer is automatically zeroed

49087. num_args

49088. DUK_OP_EXTRA, sub-operation in A

49089. insert the required matches (qmin) by copying the atom

49090. DUK_USE_INTEGER_BE

49091. out_clamped

49092. adv = 2 default OK

49093. There is no need to decref anything else than 'env': if 'env' * becomes unreachable, refzero will handle decref'ing its prototype.

49094. '-0'

49095. module table remains registered to modLoaded, minimize its size

49096. * Detach handling

49097. write back

49098. [... filename (temps) func]

49099. **comment:** It's not strictly necessary to track the current size, but * it is useful for writing robust native code.

label: code-design

49100. * Second pass parsing.

49101. **comment:** Note: cannot portably debug print a function pointer, hence 'func' not printed!

label: code-design

49102. * Arbitrary byteswap for potentially unaligned values * * Used to byteswap pointers e.g. in debugger code.

49103. **comment:** XXX: There's some overlap with duk_js_closure() here, but * seems difficult to share code. Ensure that the final function * looks the same as created by duk_js_closure().

label: code-design

49104. **comment:** It is important not to call this if the last byte read was an EOM. * Reading ahead in this scenario would cause unnecessary blocking if * another message is not available.

label: code-design

49105. DUK_USE_EXEC_TIMEOUT_CHECK

49106. * Misc defines

49107. E5.1 Section 15.1.3.4: uriUnescaped

49108. Combined size of separators already overflows

49109. -> [... res]
49110. [... this name message]
49111. "/" and not in regexp mode
49112. DUK_USE_BYTECODE_DUMP_SUPPORT
49113. **comment:** may indirectly slow access through a direct eval
 label: code-design
49114. no incref needed
49115. variable binding register if register-bound (otherwise < 0)
49116. How much to advance before next loop; note that next loop * will advance by 1 anyway, so -1 from the total escape * length (e.g. len("\uXXXXX") - 1 = 6 - 1). As a default, * 1 is good.
49117. **comment:** 'sce' points to the wrong entry here, but is no longer used
 label: code-design
49118. Note: this check relies on the fact that even a zero-size string * has a non-NUL pointer.
49119. [... varname val this] (because throw_flag == 1, always resolved)
49120. No need to reinit setjmp() catchpoint, as call handling * will store and restore our state.
49121. embed: duk_hbuffer ptr
49122. t >= 0 always true, unsigned
49123. Maximum expansion per input byte is 6: * - invalid UTF-8 byte causes "\uXXXX" to be emitted (6/1 = 6). * - 2-byte UTF-8 encodes as "\uXXXX" (6/2 = 3). * - 4-byte UTF-8 encodes as "\Uxxxxxxxx" (10/4 = 2.5).
49124. Shared helper to implement Object.getPrototypeOf and the ES6 * Object.prototype.__proto__ getter. ** http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__
49125. xxx -> DUK_HBUFFEROBJECT_ELEM_INT16
49126. vararg function, careful arg handling (e.g. thisArg may not be present)
49127. implicit this value always undefined for * declarative environment records.
49128. **comment:** * IsCallable() (E5 Section 9.11) ** XXX: API equivalent is a separate implementation now, and this has * currently no callers.
 label: code-design
49129. Extract 'top' bits of currvil; note that the extracted bits do not need * to be cleared, we just ignore them on next round.
49130. duk_unicode_ids_noabmp[]
49131. Return value of 'sz' or more indicates output was (potentially) * truncated.
49132. A finalizer is looked up from the object and up its prototype chain * (which allows inherited finalizers).
49133. Encoding/decoding flags
49134. Shared helper to implement ES6 Object.setPrototypeOf and * Object.prototype.__proto__ setter. ** [http://www.ecma-international.org/ecma-262/6.0/index.html#sec-object.setPrototypeOf](http://www.ecma-international.org/ecma-262/6.0/index.html#sec-get-object.prototype.__proto__)
49135. no need to write 'out_bufdata'
49136. new_used / size <= 1 / DIV <=> new_used <= size / DIV
49137. Register the module table early to modLoaded[] so that we can * support circular references even in modSearch(). If an error * is thrown, we'll delete the reference.
49138. label handling
49139. Expression
49140. **comment:** This is performance critical because it appears in every DECREF.
 label: code-design
49141. code_idx = entry_top + 0
49142. -> [... lval rval]
49143. * Standard algorithm succeeded without errors, check for exotic post-behaviors. ** Arguments exotic behavior in E5 Section 10.6 occurs after the standard * [[DefineOwnProperty]] has completed successfully. ** Array exotic behavior in E5 Section 15.4.5.1 is implemented partly * prior to the default [[DefineOwnProperty]], but: * - for an array index key (e.g. "10") the final 'length' update occurs here * - for 'length' key the element deletion and 'length' update occurs here
49144. * All defined array-indexed properties are in the array part * (we assume the array part is comprehensive), and all array * entries are writable, configurable, and enumerable. Thus, * nothing can prevent array entries from being deleted.
49145. **comment:** * Helpers for creating and querying pc2line debug data, which * converts a bytecode program counter to a source line number. ** The run-time pc2line data is bit-packed, and documented in: ** doc/function-objects.rst
 label: documentation
49146. Represent a null pointer as 'null' to be consistent with * the JX format variant. Native '%op' format for a NULL * pointer may be e.g. '(nil)'.
49147. eat 'throw'
49148. always grow the array, no sparse / abandon support here
49149. leave formals on stack for later use
49150. since index non-negative
49151. tzoffset
49152. duk_has_prop() popped the second 'name'
49153. varname
49154. * E5 Section 15.10.2.6 "IsWordChar" abstract operation. Assume * x < 0 for characters read outside the string.
49155. -----XXXX
49156. stack top: new time value, return 1 to allow tail calls
49157. basereg
49158. E5 Section 15.4.5.1, step 3, steps a - i are implemented here, j - n at the end
49159. jump for 'catch' case
49160. Module object containing module.exports, etc
49161. 'prototype' property for all built-in objects (which have it) has attributes: * [[Writable]] = false, * [[Enumerable]] = false, * [[Configurable]] = false
49162. stack[0] = start * stack[1] = deleteCount * stack[2...nargs-1] = items * stack[nargs] = ToObject(this) -3 * stack[nargs+1] = ToUint32(length) -2 * stack[nargs+2] = result array -1
49163. -> [... escaped_source bytecode]
49164. * Write to array part? ** Note: array abandoning requires a property resize which uses * 'rechecks' valstack for temporaries and may cause any existing * valstack pointers to be invalidated. To protect against this, * tv_obj, tv_key, and tv_val are copies of the original inputs.
49165. copy existing entries as is
49166. 5: toLocaleTimeString
49167. mark-and-sweep: children not processed
49168. Catches EOF of JSON input.
49169. radix
49170. **comment:** XXX: when optimizing for guaranteed property slots, use a guaranteed * slot for internal value; this call can then access it directly.
 label: code-design
49171. no wrapping
49172. **comment:** * Create and throw an error (originating from Duktape internally) ** Push an error object on top of the stack, possibly throw augmenting * the error, and finally longjmp. ** If an error occurs while we're dealing with the current error, we might * enter an infinite recursion loop. This is prevented by detecting a * "double fault" through the heap->handling_error flag; the recursion * then stops at the second level.
 label: code-design
49173. if property key begins with underscore, encode it with * forced quotes (e.g. "_Foo") to distinguish it from encoded * internal properties (e.g. \xffBar -> _Bar).
49174. object or array
49175. no need to check now: both success and error are OK
49176. Note: allow backtracking from p == ptr_end

49177. Relookup if relocated
49178. module.filename for .fileName, default to resolved ID if * not present.
49179. -> [... toLocaleString ToObject(val)]
49180. DUK_TOK_LBRACKET already eaten, current token is right after that
49181. **comment:** * Array needs to grow, but we don't want it becoming too sparse. * If it were to become sparse, abandon array part, moving all * array entries into the entries part (for good). ** Since we don't keep track of actual density (used vs. size) of * the array part, we need to estimate somehow. The check is made * in two parts: ** - Check whether the resize need is small compared to the * current size (relatively); if so, resize without further * checking (essentially we assume that the original part is * "dense" so that the result would be dense enough). ** - Otherwise, compute the resize using an actual density * measurement based on counting the used array entries.
label: code-design
49182. Restore stack top after unbalanced code paths.
49183. position of highest digit changed
49184. * duk_heap allocation and freeing.
49185. Careful with borrow condition: * - If borrow not subtracted: $0x12345678 - 0 - 0xffffffff = 0x12345679$ ($> 0x12345678$) * - If borrow subtracted: $0x12345678 - 1 - 0xffffffff = 0x12345678$ ($= 0x12345678$)
49186. no statement value (unlike function expression)
49187. Three digit octal escape, digits validated.
49188. **comment:** * Addition operator is different from other arithmetic * operations in that it also provides string concatenation. * Hence it is implemented separately. ** There is a fast path for number addition. Other cases go * through potentially multiple coercions as described in the * E5 specification. It may be possible to reduce the number * of coercions, but this must be done carefully to preserve * the exact semantics. ** E5 Section 11.6.1. ** Custom types also have special behavior implemented here.
label: code-design
49189. Quite approximate but should be useful for little and big endian.
49190. For X <op>= Y we need to evaluate the pre-op * value of X before evaluating the RHS: the RHS * can change X, but when we do <op> we must use * the pre-op value.
49191. Count is incremented before DUK__DRAGON4_OUTPUT_PREINC() call * on purpose, which is taken into account by the macro.
49192. NUL also comes here. Comparison order matters, 0x20 * is most common whitespace.
49193. retval for success path
49194. * Wrapping duk_safe_call() will mangle the stack, just return stack top
49195. * Node.js Buffer.prototype: toJSON()
49196. We'll need to interrupt early so recompute the init * counter to reflect the number of bytecode instructions * executed so that step counts for e.g. debugger rate * limiting are accurate.
49197. 'fileName'
49198. * Convert duk_compiler_func to a function template and add it * to the parent function table.
49199. parse args starting from "next temp"
49200. This seems like a good overall approach. Fast path for ASCII in 4 byte * blocks.
49201. * Mark-and-sweep garbage collection.
49202. **comment:** * Object compaction (emergency only). ** Object compaction is a separate step after sweeping, as there is * more free memory for it to work with. Also, currently compaction * may insert new objects into the heap allocated list and the string * table which we don't want to do during a sweep (the reachability * flags of such objects would be incorrect). The objects inserted * are currently: ** - a temporary duk_hbuffer for a new properties allocation * - if array part is abandoned, string keys are interned * * The object insertions go to the front of the list, so they do not * cause an infinite loop (they are not compacted).
label: code-design
49203. **comment:** unnecessary div for last bye
label: code-design
49204. buffer values are coerced first to string here
49205. sanity limit for function pointer size
49206. [sep ToObject(this) len sep str0 ... str(count-1)]
49207. DUK_TOK_YIELD
49208. **comment:** XXX: is valstack top best place for argument?
label: code-design
49209. reset bytecode buffer but keep current size; pass 2 will * require same amount or more.
49210. duk_hobject_run_finalizer() sets
49211. [source template closure]
49212. Successful return: restore jmpbuf and return to caller.
49213. negative -> no atom matched on previous round
49214. [... name]
49215. Note: may be NULL if first call
49216. unresolvable, no stack changes
49217. **comment:** The DataView .buffer property is ordinarily set to the argument * which is an ArrayBuffer. We accept any duk_hbufferobject as * an argument and .buffer will be set to the argument regardless * of what it is. This may be a bit confusing if the argument * is e.g. a DataView or another TypedArray view. ** XXX: Copy .buffer property from a DataView/TypedArray argument? * Create a fresh ArrayBuffer for Duktape.Buffer and Node.js Buffer * arguments? See: test-bug-dataview-buffer-prop.js.
label: code-design
49218. DUK_USE_DATE_NOW_TIME
49219. XXX: valstack handling is awkward. Add a valstack helper which * avoids dup():ing; valstack_copy(src, dst)?
49220. On entry, item at idx_func is a bound, non-lightweight function, * but we don't rely on that below.
49221. * Reference counting implementation.
49222. use strict equality instead of non-strict equality
49223. This assert depends on the input parts representing time inside * the Ecmascript range.
49224. controls for minimum array part growth
49225. -> [... key]
49226. * Returns non-zero if a key and/or value was enumerated, and: ** [enum] -> [key] (get_value == 0) * [enum] -> [key value] (get_value == 1) ** Returns zero without pushing anything on the stack otherwise.
49227. allow basic ASCII whitespace
49228. Heap allocated: return heap pointer which is NOT useful * for the caller, except for debugging.
49229. activation is a direct eval call
49230. current lex_env of the activation (created for catcher)
49231. **comment:** Figure out all active breakpoints. A breakpoint is * considered active if the current function's fileName * matches the breakpoint's fileName, AND there is no * inner function that has matching line numbers * (otherwise a breakpoint would be triggered both * inside and outside of the inner function which would * be confusing). Example: ** function foo() { * print('foo'); * function bar() { <-.. breakpoints in these * print('bar'); | lines should not affect * } <-' foo() execution * bar(); * } ** We need a few things that are only available when * debugger support is enabled: (1) a line range for * each function, and (2) access to the function * template to access the inner functions (and their * line ranges). ** It's important to have a narrow match for active * breakpoints so that we don't enter checked execution * when that's not necessary. For instance, if we're * running inside a certain function and there's * breakpoint outside in (after the call site), we * don't want to slow down execution of the function.
label: code-design
49232. start byte offset of token in lexer input
49233. * Note: lj.value1 is 'value', lj.value2 is 'resume'. * This differs from YIELD.
49234. curr_token = get/set name

49235. Initial '[' has been checked and eaten by caller.
49236. Thread may have changed, e.g. YIELD converted to THROW.
49237. Lookup to encode one byte directly into 2 characters: * * def genhextab(bswap): * for i in xrange(256): * t = chr(i).encode('hex') * if bswap: * t = t[1] + t[0] * print('0x' + t.encode('hex') + 'U') * print('big endian'); genhextab(False) * print('little endian'); genhextab(True)
49238. * 64-bit arithmetic * * There are some platforms/compilers where 64-bit types are available * but don't work correctly. Test for known cases.
49239. be paranoid for 32-bit time values (even avoiding negative ones)
49240. * Node.js Buffer.byteLength()
49241. emit instruction; could return PC but that's not needed in the majority * of cases.
49242. Fast write calls which assume you control the spare beforehand. * Multibyte write variants exist and use a temporary write pointer * because byte writes alias with anything: with a stored pointer * explicit pointer load/stores get generated (e.g. gcc -Os).
49243. All the flags fit in 16 bits, so will fit into duk_bool_t.
49244. flags: "g"
49245. For cross-checking during development: ensure dispatch count * matches cumulative interrupt counter init value sums.
49246. valstack should not need changes
49247. end of _Varmap
49248. **comment:** Certain boxed types are required to go through * automatic unboxing. Rely on internal value being * sane (to avoid infinite recursion).
 label: code-design
49249. [-0x80000000,0x7fffffff]
49250. DUK_HOBJECT_FLAG_EXOTIC_STRINGOBJ varies
49251. 'Arguments'
49252. [key getter this] -> [key retval]
49253. Retry allocation after mark-and-sweep for this * many times. A single mark-and-sweep round is * not guaranteed to free all unreferenced memory * because of finalization (in fact, ANY number of * rounds is strictly not enough).
49254. only functions can have arguments
49255. E5 Section 9.3.1
49256. * GETVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is GetValue'd] * 8.7.1 GetValue (V) * 8.12.1 [[GetOwnProperty]] (P) * 8.12.2 [[GetProperty]] (P) * 8.12.3 [[Get]] (P) * * If 'throw' is true, always leaves two values on top of stack: [val this]. * * If 'throw' is false, returns 0 if identifier cannot be resolved, and the * stack will be unaffected in this case. If identifier is resolved, returns * 1 and leaves [val this] on top of stack. * * Note: the 'strict' flag of a reference returned by GetIdentifierReference * is ignored by GetValue. Hence we don't take a 'strict' parameter. * * The 'throw' flag is needed for implementing 'typeof' for an unreferenced * identifier. An unreferenced identifier in other contexts generates a * ReferenceError.
49257. **comment:** XXX: top of stack must contain value, which helper doesn't touch, * rework to use tv_val directly?
 label: code-design
49258. * Executor interrupt handling * * The handler is called whenever the interrupt countdown reaches zero * (or below). The handler must perform whatever checks are activated, * e.g. check for cumulative step count to impose an execution step * limit or check for breakpoints or other debugger interaction. * * When the actions are done, the handler must reinit the interrupt * init and counter values. The 'init' value must indicate how many * bytecode instructions are executed before the next interrupt. The * counter must interface with the bytecode executor loop. Concretely, * the new init value is normally one higher than the new counter value. * For instance, to execute exactly one bytecode instruction the init * value is set to 1 and the counter to 0. If an error is thrown by the * interrupt handler, the counters are set to the same value (e.g. both * to 0 to cause an interrupt when the next bytecode instruction is about * to be executed after error handling). * * Maintaining the init/counter value properly is important for accurate * behavior. For instance, executor step limit needs a cumulative step * count which is simply computed as a sum of 'init' values. This must * work accurately even when single stepping.
49259. 0xd0...0xdf
49260. tv1 is -below- valstack_bottom
49261. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT32
49262. [obj Properties]
49263. * First attempt
49264. not found
49265. Loop check.
49266. XXX: just assert non-NULL values here and make caller arguments * do the defaulting to the default implementations (smaller code)?
49267. reserved words: additional future reserved words in strict mode
49268. * String table resize check. * * Note: this may silently (and safely) fail if GC is caused by an * allocation call in stringtable resize_hash(). Resize_hash() * will prevent a recursive call to itself by setting the * DUK_MS_FLAG_NO_STRINGTABLE_RESIZE in heap->mark_and_sweep_base_flags.
49269. Sync and NULL early.
49270. 12 to 21
49271. **comment:** XXX: we should never shrink here; when we error out later, we'd * need to potentially grow the value stack in error unwind which could * cause another error.
 label: code-design
49272. -> pushes fixed buffer
49273. **comment:** XXX: improve check; check against nregs, not against top
 label: code-design
49274. byte sizes will match
49275. max # of values initialized in one MPUTARR set
49276. not reached
49277. -> [...] this]
49278. * Local defines and forward declarations.
49279. **comment:** prev value must be unused, no decref
 label: code-design
49280. Calling duk_debugger_cooperate() while Duktape is being * called into is not supported. This is not a 100% check * but prevents any damage in most cases.
49281. Note: careful with indices like '-x'; if 'x' is zero, it refers to bottom
49282. x >= y --> not (x < y)
49283. yes, must set array length explicitly
49284. * Post resize assertions.
49285. Note: for dstlen=0, dst may be NULL
49286. !DUK_USE_TRACEBACKS
49287. -> [func funcname env funcname]
49288. Constants for duk_lexer_ctx.buf.
49289. Difference is in 100ns units, convert to milliseconds w/o fractions
49290. * "IdentifierPart" production check.
49291. prev exists and is not a letter
49292. Note: no recursion issue, we can trust 'map' to behave
49293. if fails, e_size will be zero = not an issue, except performance-wise
49294. **comment:** XXX: use a NULL error handler for the finalizer call?
 label: code-design
49295. **comment:** workaround: max nbis = 24 now
 label: code-design
49296. DUK_USE_HEX_FASTPATH
49297. DUK_UTIL_H_INCLUDED
49298. 'Infinity'

49299. We know _Varmap only has own properties so walk property * table directly. We also know _Varmap is dense and all * values are numbers; assert for these. GC and finalizers * shouldn't affect _Varmap so side effects should be fine.

49300. debugger state, only relevant when attached

49301. highest value of temp_reg (temp_max - 1 is highest used reg)

49302. * Init compiler and lexer contexts

49303. Input values are signed 48-bit so we can detect overflow * reliably from high bits or just a comparison.

49304. pre/post opcode values have constraints,

49305. allocated heap objects

49306. re-read to avoid spill / fetch

49307. LRU: move our entry to first

49308. argument/function declaration shadows 'arguments'

49309. Proxy target

49310. unconditional block

49311. **comment:** XXX: Using a string return value forces a string intern which is * not always necessary. As a rough performance measure, hex encode * time for tests/perf/test-hex-encode.js dropped from ~35s to ~15s * without string coercion. Change to returning a buffer and let the * caller coerce to string if necessary? **label:** code-design

49312. * Node.js Buffer.prototype.copy()

49313. DUK_VERBOSE_ERRORS

49314. skip ''

49315. XXX: additional conditions when to close variables? we don't want to do it * unless the environment may have "escaped" (referenced in a function closure). * With delayed environments, the existence is probably good enough of a check.

49316. When debugger is enabled, we need to recheck the activation * status after returning. This is now handled by call handling * and heap->dbg_force_restart.

49317. 'byteLength'

49318. * Parsing implementation. * * JSON lexer is now separate from duk_lexer.c because there are numerous * small differences making it difficult to share the lexer. * * The parser here works with raw bytes directly; this works because all * JSON delimiters are ASCII characters. Invalid xUTF-8 encoded values * inside strings will be passed on without normalization; this is not a * compliance concern because compliant inputs will always be valid * CESU-8 encodings.

49319. * Detect iteration statements; if encountered, establish an * empty label.

49320. skip ')

49321. * Poppers

49322. DUK_HOBJECT_H_INCLUDED

49323. always at least the header

49324. A -> target register * B -> bytecode (also contains flags) * C -> escaped source

49325. Fast path, decode as is.

49326. * "/" followed by something in regexp mode. See E5 Section 7.8.5. * * RegExp parsing is a bit complex. First, the regexp body is delimited * by forward slashes, but the body may also contain forward slashes as * part of an escape sequence or inside a character class (delimited by * square brackets). A mini state machine is used to implement these. * * Further, an early (parse time) error must be thrown if the regexp * would cause a run-time error when used in the expression new RegExp(...). * Parsing here simply extracts the (candidate) regexp, and also accepts * invalid regular expressions (which are delimited properly). The caller * (compiler) must perform final validation and regexp compilation. * * RegExp first char may not be '/' (single line comment) or '*' (multi- * line comment). These have already been checked above, so there is no * need below for special handling of the first regexp character as in * the E5 productions. * * About unicode escapes within regexp literals: * * E5 Section 7.8.5 grammar does NOT accept \uHHHH escapes. * However, Section 6 states that regexps accept the escapes, * see paragraph starting with "In string literals...". * The regexp grammar, which sees the decoded regexp literal * (after lexical parsing) DOES have a \uHHHH unicode escape. * So, for instance: * * \u1234/ * * should first be parsed by the lexical grammar as: * * \'u' RegularExpressionBackslashSequence * '1' RegularExpressionNonTerminator * '2' RegularExpressionNonTerminator * '3' RegularExpressionNonTerminator * '4' RegularExpressionNonTerminator * * and the escape itself is then parsed by the regexp engine. * This is the current implementation. * * Minor spec inconsistency: * * E5 Section 7.8.5 RegularExpressionBackslashSequence is: * * \ RegularExpressionNonTerminator * * while Section A.1 RegularExpressionBackslashSequence is: * * \ NonTerminator * * The latter is not normative and a typo. *

49327. 'break'

49328. in yielder's context

49329. * E5 Section 15.4.5.1, steps 3.k - 3.n. The order at the end combines * the error case 3.l.iii and the success case 3.m-3.n. * * Note: 'length' is always in entries part, so no array abandon issues for * 'writable' update.

49330. * Avoid a GC if GC is already running. This can happen at a late * stage in a GC when we try to e.g. resize the stringtable * or compact objects.

49331. Reject attempt to change virtual properties: not part of the * standard algorithm, applies currently to e.g. virtual index * properties of buffer objects (which are virtual but writable). * (Cannot "force" modification of a virtual property.)

49332. MakeDay

49333. Push 'this' binding, check that it is a Date object; then push the * internal time value. At the end, stack is: [... this timeval]. * Returns the time value. Local time adjustment is done if requested.

49334. Cannot use duk_to_string() on the buffer because it is usually * larger than 'len'. Also, 'buf' is usually a stack buffer.

49335. non-leap year: sunday, monday, ...

49336. activation has tail called one or more times

49337. extended types: custom encoding

49338. reg

49339. -> [... enum key val]

49340. EOF

49341. XXX: optimize value stack operation

49342. * Proxy object handling

49343. * Then decode the builtins init data (see genbuiltins.py) to * init objects

49344. mark as complex (capture handling)

49345. string is a reserved word (non-strict)

49346. s <- b^(1-e) * 2

49347. * Object inspection commands: GetHeapObjInfo, GetObjPropDesc, * GetObjPropDescRange

49348. * Unreachable object about to be swept. Finalize target refcounts * (objects which the unreachable object points to) without doing * refzero processing. Recursive decrefs are also prevented when * refzero processing is disabled. * * Value cannot be a finalizable object, as they have been made * temporarily reachable for this round.

49349. Map DUK_FLX_xxx to byte size.

49350. Note that we currently parse -bytes-, not codepoints. * All non-ASCII extended UTF-8 will encode to bytes >= 0x80, * so they'll simply pass through (valid UTF-8 or not).

49351. * Get/require

49352. key, getter and setter, now reachable through object

49353. **comment:** * Finalize objects in the finalization work list. Finalized * objects are queued back to heap_allocated with FINALIZED set. * * Since finalizers may cause arbitrary side effects, they are * prevented during string table and object property allocation * resizing using the DUK_MS_FLAG_NO_FINALIZERS flag in * heap->mark_and_sweep_base_flags. In this case the objects * remain in the finalization work list after mark-and-sweep * exits and they may be finalized on the next pass. * * Finalization currently happens inside "MARKANDSWEET_RUNNING" * protection (no mark-and-sweep may be triggered by the * finalizers). As a side effect: * * 1) an out-of-memory error inside a finalizer will not * cause a mark-and-sweep and may cause the finalizer * to fail unnecessarily * * 2) any temporary objects whose refcount decreases to zero * during finalization will not be put into refzero_list; * they can only be collected by another mark-and-sweep * * This is not optimal, but since the sweep for this phase has * already happened, this is probably good enough for now. **label:** code-design

49354. * A Dragon4 number-to-string variant, based on: * * Guy L. Steele Jr., Jon L. White: "How to Print Floating-Point Numbers * Accurately" * * Robert G. Burger, R. Kent Dybvig: "Printing Floating-Point Numbers * Quickly and Accurately" * * The current algorithm is based on Figure 1 of the Burger-Dybvig paper, * i.e.

the base implementation without logarithm estimation speedups * (these would increase code footprint considerably). Fixed-format output * does not follow the suggestions in the paper; instead, we generate an * extra digit and round-with-carry. ** The same algorithm is used for number parsing (with b=10 and B=2) * by generating one extra digit and doing rounding manually. ** See doc/number-conversion.rst for limitations.

49355. holder will be set to the target object, not the actual object * where the property was found (see duk_get_identifier_reference()).

49356. 'Array'

49357. * INITSET/INITGET are only used to initialize object literal keys. * The compiler ensures that there cannot be a previous data property * of the same name. It also ensures that setter and getter can only * be initialized once (or not at all).

49358. extended utf-8 not allowed for URIs

49359. * E5 Section 15.7.2.1 requires that the constructed object * must have the original Number.prototype as its internal * prototype. However, since Number.prototype is non-writable * and non-configurable, this doesn't have to be enforced here: * The default object (bound to 'this') is OK, though we have * to change its class. * * Internal value set to ToNumber(arg) or +0; if no arg given, * ToNumber(undefined) = NaN, so special treatment is needed * (above). String internal value is immutable.

49360. bytecode limits

49361. guaranteed to find an empty slot

49362. Because 'day since epoch' can be negative and is used to compute weekday * using a modulo operation, add this multiple of 7 to avoid negative values * when year is below 1970 epoch. Ecmascript time values are restricted to * +/- 100 million days from epoch, so this adder fits nicely into 32 bits. * Round to a multiple of 7 (= floor(100000000 / 7) * 7) and add margin.

49363. Allow trailing garbage (e.g. treat "123foo" as "123")

49364. **comment:** XXX: Are steps 6 and 7 in E5 Section 15.11.4.4 duplicated by * accident or are they actually needed? The first ToString() * could conceivably return 'undefined'.

label: code-design

49365. **comment:** Size sanity check. Should not be necessary because caller is * required to check this, but we don't want to cause a segfault * if the size wraps either in duk_size_t computation or when * storing the size in a 16-bit field.

label: code-design

49366. steps 3.h and 3.i

49367. Note: toJSON() is a generic function which works even if 'this' * is not a Date. The sole argument is ignored.

49368. module (argument)

49369. valstack space that suffices for all local calls, including recursion * of other than Duktape calls (getters etc)

49370. current paren level allows 'in' token

49371. **comment:** XXX: fast-int-to-double

label: code-design

49372. * ArrayBuffer.isView()

49373. Magically bound variable cannot be an accessor. However, * there may be an accessor property (or a plain property) in * place with magic behavior removed. This happens e.g. when * a magic property is redefined with defineProperty(). * Cannot assert for "not accessor" here.

49374. * We could clear the book-keeping variables for the topmost activation, * but don't do so now.

49375. * Debugging enabled

49376. * E5 Section 7.4, allow SourceCharacter (which is any 16-bit * code point).

49377. **comment:** XXX: duk_set_length

label: code-design

49378. **comment:** XXX: which lists should participate? to be finalized?

label: code-design

49379. XXXXXX-- -----

49380. * Bitstream decoder.

49381. * The handling here is a bit tricky. If a previous ']' has been processed, * we have a pending split1 and a pending jump (for a previous match). These * need to be back-patched carefully. See docs for a detailed example.

49382. Step 16: update length; note that the final length may be above 32 bit range * (but we checked above that this isn't the case here)

49383. **comment:** XXX: 'internal error' is a bit of a misnomer

label: code-design

49384. **comment:** In compatible mode and standard JSON mode, output * something useful for non-BMP characters. This won't * roundtrip but will still be more or less readable and * more useful than an error.

label: code-design

49385. A debugger forced interrupt check is not needed here, as * problematic safe calls are not caused by side effects.

49386. Real world behavior for map(): trailing non-existent * elements don't invoke the user callback, but are still * counted towards result 'length'.

49387. Ensure argument name is not a reserved word in current * (final) strictness. Formal argument parsing may not * catch reserved names if strictness changes during * parsing. ** We only need to do this in strict mode because non-strict * keyword are always detected in formal argument parsing.

49388. retval (sub-atom char length) unused, tainted as complex above

49389. no pre-checks now, assume a previous yield() has left things in * tip-top shape (longjmp handler will assert for these).

49390. * Peephole optimizer for finished bytecode. ** Does not remove opcodes; currently only straightens out unconditional * jump chains which are generated by several control structures.

49391. should not happen

49392. Avoid doing an actual write callback with length == 0, * because that's reserved for a write flush.

49393. DUK_TOK_REGEXP

49394. * Peephole optimize JUMP chains.

49395. * Execution timeout check

49396. Recursive value reviver, implements the Walk() algorithm. No C recursion * check is done here because the initial parsing step will already ensure * there is a reasonable limit on C recursion depth and hence object depth.

49397. type to represent a reg/const reference during compilation

49398. TypedArray views need an automatic ArrayBuffer which must be * provided as .buffer property of the view. Just create a new * ArrayBuffer sharing the same underlying buffer. ** The ArrayBuffer offset is always set to zero, so that if one * accesses the ArrayBuffer at the view's .byteOffset, the value * matches the view at index 0.

49399. **comment:** Indent helper. Calling code relies on js_ctx->recursion_depth also being * directly related to indent depth.

label: code-design

49400. DUK_TOK_PUBLIC

49401. E5 Sections 11.8.4, 11.8.5; x >= y --> not (x < y)

49402. 'get'

49403. **comment:** function name (borrowed reference), ends up in _name

label: code-design

49404. repl_value

49405. quotes

49406. 'undefined' already on stack top

49407. 'res' is used for "left", and 'tmp' for "right"

49408. def_value

49409. Delete elements required by a smaller length, taking into account * potentially non-configurable elements. Returns non-zero if all * elements could be deleted, and zero if all or some elements could * not be deleted. Also writes final "target length" to 'out_result_len'. * This is the length value that should go into the 'length' property * (must be set by the caller). Never throws an error.

49410. * duk_handle_call_protected() and duk_handle_call_unprotected(): * call into a Duktape/C or an Ecmascript function from any state. ** Input stack (thr): * * [func this arg1 ... argN] * * Output stack (thr): * * [retval] (DUK_EXEC_SUCCESS) * [errobj] (DUK_EXEC_ERROR (normal error), protected call) * * Even when executing a protected call an error may be thrown in rare cases * such as an insane num_stack_args argument. If there is no catchpoint for * such errors, the

fatal error handler is called. ** The error handling path should be error free, even for out-of-memory * errors, to ensure safe sandboxing. (As of Duktape 1.4.0 this is not * yet the case, see XXX notes below.)

49411. **comment:** XXX: skip null filename?

label: code-design

49412. avoid multiply

49413. New require() function for module, updated resolution base

49414. **comment:** * Function pointers ** Printing function pointers is non-portable, so we do that by hex printing * bytes from memory.

label: code-design

49415. "-Infinity", '-' has been eaten

49416. make absolute

49417. 'act' already set above

49418. DUK_BUFOBJ_UINT8CLAMPEDARRAY

49419. **comment:** XXX: turkish / azeri, lowercase rules

label: code-design

49420. **comment:** Coerce like a string. This makes sense because addition also treats * buffers like strings.

label: code-design

49421. Non-zero refcounts should not happen for unreachable strings, * because we recount finalize all unreachable objects which * should have decreased unreachable string refcounts to zero * (even for cycles).

49422. no comma

49423. strict outer context

49424. jump for 'finally' case or end (if no finally)

49425. * Structs

49426. DUK__L0 -> '\ char * DUK__L1 ... DUK__L5 -> more lookup

49427. from next pc

49428. If the value happens to be 0xFFFFFFFF, it's not a valid array index * but will then match DUK__NO_ARRAY_INDEX.

49429. use 'left' as a temp

49430. JSON

49431. binary arithmetic; DUK_OP_ADD, DUK_OP_EQ, other binary ops

49432. Innermost fast path processes 4 valid base-64 characters at a time * but bails out on whitespace, padding chars ('=') and invalid chars. * Once the slow path segment has been processed, we return to the * inner fast path again. This handles e.g. base64 with newlines * reasonably well because the majority of a line is in the fast path.

49433. Key and value indices are either (-2, -1) or (-1, -2). Given idx_key, * idx_val is always (idx_key ^ 0x01).

49434. * Pointer built-ins

49435. Zero length string is not accepted without quotes

49436. [key] -> []

49437. No activation, no variable access. Could also pretend * we're in the global program context and read stuff off * the global object.

49438. current function being compiled (embedded instead of pointer for more compact access)

49439. Note: src_data may be NULL if input is a zero-size * dynamic buffer.

49440. Note: 'this' may be bound to any value, not just an object

49441. **comment:** XXX: Not needed for now, so not implemented. Note that * function pointers may have different size/layout than * a void pointer.

label: requirement

49442. out of spec, don't care

49443. **comment:** XXX: idx_val would fit into 16 bits, but using duk_small_uint_t * might not generate better code due to casting.

label: code-design

49444. used by e.g. duk_regexp_executor.c, string built-ins

49445. **comment:** XXX: could also copy from template, but there's no way to have any * other value here now (used code has no access to the template).

label: code-design

49446. Share yield longjmp handler.

49447. curr_token slot2 (matches 'lex' slot2_idx)

49448. Error message doesn't matter: the error is ignored anyway.

49449. Have a calling activation, check for direct eval (otherwise * assume indirect eval.

49450. **comment:** Compact but lots of churn.

label: code-design

49451. Output shuffling: only one output register is realistically possible. ** (Zero would normally be an OK marker value: if the target register * was zero, it would never be shuffled. But with DUK_USE_SHUFFLE_TORTURE * this is no longer true, so use -1 as a marker instead.)

49452. **comment:** actually used

label: code-design

49453. rule match

49454. * Helper structs

49455. terminating conditions

49456. see below

49457. allow automatic semicolon even without lineterm (compatibility)

49458. Outside any activation -> put to global.

49459. +0 = break, +1 = continue

49460. try locale specific formatter; if it refuses to format the * string, fall back to an ISO 8601 formatted value in local * time.

49461. -> [val arr]

49462. * Byte matching for back-references would be OK in case- * sensitive matching. In case-insensitive matching we need * to canonicalize characters, so back-reference matching needs * to be done with codepoints instead. So, we just decode * everything normally here, too. ** Note: back-reference index which is 0 or higher than * NCapturingParens (= number of capturing parens in the * -entire- regexp) is a compile time error. However, a * backreference referring to a valid capture which has * not matched anything always succeeds! See E5 Section * 15.10.2.9, step 5, sub-step 3.

49463. this is needed for regexp mode

49464. jump to next case clause

49465. pop plain buffer, now reachable through h_bufres

49466. * PUTPROP: Ecmascript property write. ** Unlike Ecmascript primitive which returns nothing, returns 1 to indicate * success and 0 to indicate failure (assuming throw is not set). ** This is an extremely tricky function. Some examples: * * * Currently a decref may trigger a GC, which may compact an object's * property allocation. Consequently, any entry indices (e_idx) will * be potentially invalidated by a decref. * * * Exotic behaviors (strings, arrays, arguments object) require, * among other things: * * - Preprocessing before and postprocessing after an actual property * write. For example, array index write requires pre-checking the * array 'length' property for access control, and may require an * array 'length' update after the actual write has succeeded (but * not if it fails). * * - Deletion of multiple entries, as a result of array 'length' write. * * * Input values are taken as pointers which may point to the valstack. * If valstack is resized because of the put (this may happen at least * when the array part is abandoned), the pointers can be invalidated. * (We currently make a copy of all of the input values to avoid issues.)

49467. add AD*2^16

49468. Proxy handler

49469. **comment:** * Tailcall handling ** Although the callstack entry is reused, we need to explicitly unwind * the current activation (or simulate an unwind). In particular, the * current activation must be closed, otherwise something like * test-bug-reduce-judofy.js results. Also catchstack needs be unwound * because there may be non-error-catching label entries in valid tail calls.

label: code-design

49470. **comment:** Should not be required because the code below always sets both high * and low parts, but at least gcc-4.4.5 fails to deduce this correctly * (perhaps because the low part is set (seemingly) conditionally in a * loop), so this is here to avoid the bogus warning.
label: code-design

49471. Impose a maximum string length for now. Restricted artificially to * ensure adding a heap header length won't overflow size_t. The limit * should be synchronized with DUK_HBUFFER_MAX_BYTELEN. ** E5.1 makes provisions to support strings longer than 4G characters. * This limit should be eliminated on 64-bit platforms (and increased * closer to maximum support on 32-bit platforms).

49472. relookup exports from module.exports in case it was changed by modSearch

49473. * Indexed read/write helpers (also used from outside this file)

49474. adv = 2 - 1 default OK

49475. **comment:** XXX: the key in 'key in obj' is string coerced before we're called * (which is the required behavior in E5/E5.1/E6) so the key is a string * here already.
label: code-design

49476. **comment:** XXX: side effect handling is quite awkward here
label: code-design

49477. act->func

49478. **comment:** Set stack top within currently allocated range, but don't reallocate. * This is performance critical especially for call handling, so whenever * changing, profile and look at generated code.
label: code-design

49479. **comment:** Neutered checks not necessary here: neutered buffers have * zero 'length' so we'll effectively skip them.
label: code-design

49480. Call setup checks callability.

49481. **comment:** Flags for intermediate value coercions. A flag for using a forced reg * is not needed, the forced_reg argument suffices and generates better * code (it is checked as it is used).
label: code-design

49482. **comment:** Non-ASCII slow path (range-by-range linear comparison), very slow
label: code-design

49483. Contract, either: * - Push value on stack and return 1 * - Don't push anything on stack and return 0

49484. 0: not IdentifierStart or IdentifierPart * 1: IdentifierStart and IdentifierPart * -1: IdentifierPart only

49485. DUK_BUFOBJ_UINT32ARRAY

49486. * Restore 'caller' property for non-strict callee functions.

49487. $(\geq e 0) \text{ AND } (\text{not } (= f (\text{expt } b (- p 1))))$ * * be < (expt b e) == b^e * r <- (* f be 2) == 2 * f * b^e [if b==2 -> f * b^(e+1)] * s <- 2 * m+ <- be == b^e * m- <- be == b^e * k <- 0 * B <- B * low_ok <- round * high_ok <- round

49488. A -> unused (reserved for flags, for consistency with DUK_OP_CALL) * B -> target register and start reg: constructor, arg1, ..., argN * (for DUK_OP_NEWI, 'b' is indirect) * C -> num args (N)

49489. 0x60...0x6f

49490. ≥ 0

49491. mimic semantics for strings

49492. Parse a single statement. ** Creates a label site (with an empty label) automatically for iteration * statements. Also "peels off" any label statements for explicit labels.

49493. DUK_FLD_8BIT

49494. [key]

49495. normal key/value

49496. mm <- b^e

49497. steps 4.a and 4.b are tricky

49498. The external string struct is defined even when the feature is inactive.

49499. fixed arg count

49500. A -> flags * B -> return value reg/const * C -> currently unused

49501. **comment:** Because there are quite many call sites, pack error code (require at most * 8-bit) into a single argument.
label: code-design

49502. Raw helper to extract internal information / statistics about a value. * The return values are version specific and must not expose anything * that would lead to security issues (e.g. exposing compiled function * 'data' buffer might be an issue). Currently only counts and sizes and * such are given so there should not be a security impact.

49503. -> [... resume template newobj]

49504. XXX: This causes recursion up to inner function depth * which is normally not an issue, e.g. mark-and-sweep uses * a recursion limiter to avoid C stack issues. Avoiding * this would mean some sort of a work list or just refusing * to serialize deep functions.

49505. * Size-optimized pc->line mapping.

49506. return as is

49507. value1 -> resume value, value2 -> resume thread, iserror -> error/normal

49508. function: create binding for func name (function templates only, used for named function expressions)

49509. The low 8 bits map directly to duk_hobject.h DUK_PROPDESC_FLAG_xxx. * The remaining flags are specific to the debugger.

49510. **comment:** XXX: copy from caller?
label: code-design

49511. always for register bindings

49512. **comment:** XXX: optimize: allocate an array part to the necessary size (upwards * estimate) and fill in the values directly into the array part; finally * update 'length'.
label: code-design

49513. heap->dbg_udata: keep

49514. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

49515. IdentifierStart production with Letter, ASCII, and non-BMP excluded

49516. **comment:** * Custom formatter for debug printing, allowing Duktape specific data * structures (such as tagged values and heap objects) to be printed with * a nice format string. Because debug printing should not affect execution * state, formatting here must be independent of execution (see implications * below) and must not allocate memory. ** Custom format tags begin with a '%!' to safely distinguish them from * standard format tags. The following conversions are supported: * * %!T tagged value (duk_tval *) * %!O heap object (duk_heapobj *) * %!I decoded bytecode instruction * %!C bytecode instruction opcode name (arg is long) * * Everything is serialized in a JSON-like manner. The default depth is one * level, internal prototype is not followed, and internal properties are not * serialized. The following modifiers change this behavior: * * @ print pointers * # print binary representations (where applicable) * d deep traversal of own properties (not prototype) * p follow prototype chain (useless without 'd') * i include internal properties (other than prototype) * x hexdump buffers * h heavy formatting * * For instance, the following serializes objects recursively, but does not * follow the prototype chain nor print internal properties: "%!do". ** Notes: * * Standard snprintf return value semantics seem to vary. This * implementation returns the number of bytes it actually wrote * (excluding the null terminator). If retval == buffer size, * output was truncated (except for corner cases). ** Output format is intentionally different from Ecmascript * formatting requirements, as formatting here serves debugging * of internals. ** Depth checking (and updating) is done in each type printer * separately, to allow them to call each other freely. ** Some pathological structures might take ages to print (e.g. * self recursion with 100 properties pointing to the object * itself). To guard against these, each printer also checks * whether the output buffer is full; if so, early exit. ** Reference loops are detected using a loop stack.
label: code-design

49517. dummy value used as marker

49518. never here, but fall through

49519. **comment:** * duk_handle_safe_call(): make a "C protected call" within the * current activation. ** The allowed thread states for making a call are the same as for * duk_handle_call_xxx(). * * Error handling is similar to duk_handle_call_xxx(); errors may be thrown * (and result in a fatal error) for insane arguments.
label: code-design

49520. must never longjmp

49521. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside object 'a_size'.

49522. Must set 'length' explicitly when using duk_xdef_prop_xxx() to * set the values.

49523. heapdr size and additional allocation size, followed by * type specific stuff (with varying value count)

49524. stack top contains 'true'

49525. callbacks and udata; dbg_read_cb != NULL is used to indicate attached state

49526. '\xffHandler'

49527. Note: string is a terminal heap object, so no depth check here

49528. 0x80-0x8f

49529. * Internal helper for defining an accessor property, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes. This is called * very rarely, so the implementation first sets a value to undefined * and then changes the entry to an accessor (this is to save code space).

49530. Debug print which visualizes the qsort partitioning process.

49531. XXX: this placeholder is not always correct, but use for now. * It will fail in corner cases; see test-dev-func-cons-args.js.

49532. r <- (2 * f) * b^e

49533. unreferenced with some options

49534. always terminates led()

49535. (reference is valid as long activation exists)

49536. don't report __FILE__ / __LINE__ as fileName/lineNumber

49537. **comment:** Note: don't bail out early, we must read all the ranges from * bytecode. Another option is to skip them efficiently after * breaking out of here. Prefer smallest code.
label: code-design

49538. Pause for all step types: step into, step over, step out. * This is the only place explicitly handling a step out.

49539. awkward; we assume there is space for this

49540. Peek ahead in the stream one byte.

49541. * Breakpoint management

49542. * Compact an object

49543. slot B is both a target and a source (used by extraops like DUK_EXTRAOP_INSTOF)

49544. alias for above

49545. handle comma and closing brace

49546. shrink case; leave some spare

49547. strictness is not inherited, intentional

49548. only Array.prototype matches

49549. Parse a function-like expression, assuming that 'comp_ctx->curr_func' is * correctly set up. Assumes that curr_token is just after 'function' (or * 'set'/'get' etc).

49550. function: always register bound

49551. **comment:** XXX: fast primitive to set a bunch of values to UNDEFINED
label: code-design

49552. Buffer size needed for duk_bi_date_format_timeval(). * Accurate value is 32 + 1 for NUL termination: * >>> len('+123456-01-23T12:34:56.123+12:34') * 32 *
Include additional space to be safe.

49553. We want to compare the slice/view areas of the arguments. * If either slice/view is invalid (underlying buffer is shorter) * ensure equals() is false, but otherwise the only thing that * matters is to be memory safe.

49554. This seems reasonable overall.

49555. Loop prevention

49556. 'throw'

49557. we're running inside the caller's activation, so no change in call/catch stack or valstack bottom

49558. Assume that the native representation never contains a closing * parenthesis.

49559. varargs marker

49560. entry level reached

49561. Current strictness flag: affects API calls.

49562. Because value stack init policy is 'undefined above top', * we don't need to write, just assert.

49563. '*'

49564. call is from executor, so we know we have a jmpbuf

49565. Yield/resume book-keeping.

49566. finally part will catch

49567. Quite lenient, e.g. allow empty as zero, but don't allow trailing * garbage.

49568. IEEE requires that NaNs compare false

49569. Load inner functions to value stack, but don't yet copy to buffer.

49570. [start end]

49571. **comment:** Try to make do with a stack buffer to avoid allocating a temporary buffer. * This works 99% of the time which is quite nice.
label: code-design

49572. This testcase fails when Emscripten-generated code runs on Firefox. * It's not an issue because the failure should only affect packed * duk_tval representation, which is not used with Emscripten.

49573. **comment:** XXX: keys is an internal object with all keys to be processed * in its (gapless) array part. Because nobody can touch the keys * object, we could iterate its array part directly (keeping in mind * that it can be reallocated).
label: code-design

49574. Return value is like write(), number of bytes written. * The return value matters because of code like: * "off += buf.copy(...)".

49575. Exponent without digits (e.g. "1e" or "1e+"). If trailing garbage is * allowed, ignore exponent part as garbage (= parse as "1", i.e. exp 0).

49576. DUK_TOK_THROW

49577. **comment:** Note: not a typo, "object" is returned for a null value
label: documentation

49578. atom_char_length, atom_start_offset, atom_start_offset reflect the * atom matched on the previous loop. If a quantifier is encountered * on this loop, these are needed to handle the quantifier correctly. * new_atom_char_length etc are for the atom parsed on this round; * they're written to atom_char_length etc at the end of the round.

49579. Original idiom used, minimal code size.

49580. **comment:** 'fun' is quite rarely used, so no local for it
label: code-design

49581. **comment:** XXX: fastint
label: code-design

49582. Note: in integer arithmetic, (x / 4) is same as floor(x / 4) for non-negative * values, but is incorrect for negative ones.

49583. normal

49584. Careful with wrapping (left shifting idx would be unsafe).

49585. **comment:** XXX: for negative input offsets, 'offset' will be a large * positive value so the result here is confusing.
label: code-design

49586. no overflow

49587. "+11:22:0"

49588. borrowed: function being executed; for bound function calls, this is the final, real function, NULL for lightfuncs

49589. **comment:** * Augment error (throw time), unless alloc/double error
label: code-design

49590. -> [... ToObject(this) ToUInt32(length) arg[i]]
49591. The specification has quite awkward order of coercion and * checks for toPrecision(). The operations below are a bit * reordered, within constraints of observable side effects.
49592. caller
49593. match followed by capture(s)
49594. return input buffer, converted to a Duktape.Buffer object * if called as a constructor (no change if called as a * function).
49595. We could use a switch-case for the class number but it turns out * a small if-else ladder on class masks is better. The if-ladder * should be in order of relevancy.
49596. * New length is smaller than old length, need to delete properties above * the new length. * * If array part exists, this is straightforward: array entries cannot * be non-configurable so this is guaranteed to work. * * If array part does not exist, array-indexed values are scattered * in the entry part, and some may not be configurable (preventing length * from becoming lower than their index + 1). To handle the algorithm * in E5 Section 15.4.5.1, step 1 correctly, we scan the entire property * set twice.
49597. 'ptr' is evaluated both as LHS and RHS.
49598. * Ecmascript [[Class]]
49599. **comment:** Quick reject of too large or too small exponents. This check * would be incorrect for zero (e.g. "0e1000" is zero, not Infinity) * so zero check must be above.
label: code-design
49600. finalize_list will always be processed completely
49601. **comment:** XXX: exposed duk_debug_read_pointer
label: code-design
49602. -> [... key val replacer holder]
49603. If index is not valid, idx will be DUK__NO_ARRAY_INDEX which * is 0xffffffffUL. We don't need to check for that explicitly * because 0xffffffffUL will never be inside bufferobject length.
49604. open scope information, for compiled functions only
49605. Handle empty separator case: it will always match, and always * triggers the check in step 13.c.iii initially. Note that we * must skip to either end of string or start of first codepoint, * skipping over any continuation bytes! * * Don't allow an empty string to match at the end of the input.
49606. temp reg handling
49607. * Parse function body
49608. x <- y - z
49609. 'Pointer'
49610. relookup if changed
49611. **comment:** * Augment an error at creation time with _Tracedata/fileName/lineNumber * and allow a user error handler (if defined) to process/replace the error. * The error to be augmented is at the stack top. * * thr: thread containing the error value * thr_callstack: thread which should be used for generating callstack etc. * c_filename: C __FILE__ related to the error * c_line: C __LINE__ related to the error * noblame_fileline: if true, don't fileName/line as error source, otherwise use traceback * (needed because user code filename/line are reported but internal ones * are not) * * XXX: rename noblame_fileline to flags field; combine it to some existing * field (there are only a few call sites so this may not be worth it).
label: code-design
49612. throw_flag
49613. * Else, must be one of: * - ExpressionStatement, possibly a directive (String) * - LabelledStatement (Identifier followed by ':') * * Expressions beginning with 'function' keyword are covered by a case * above (such expressions are not allowed in standard E5 anyway). * Also expressions starting with '{' are interpreted as block * statements. See E5 Section 12.4. * * Directive detection is tricky; see E5 Section 14.1 on directive * prologue. A directive is an expression statement with a single * string literal and an explicit or automatic semicolon. Escape * characters are significant and no parens etc are allowed: * * 'use strict'; // valid 'use strict' directive * 'use\u0020strict'; // valid directive, not a 'use strict' directive * ('use strict'); // not a valid directive * * The expression is determined to consist of a single string literal * based on duk__expr_nud() and duk__expr_led() call counts. The string literal * of a 'use strict' directive is determined to lack any escapes based * num_escapes count from the lexer. Note that other directives may be * allowed to contain escapes, so a directive with escapes does not * terminate a directive prologue. * * We rely on the fact that the expression parser will not emit any * code for a single token expression. However, it will generate an * intermediate value which we will then successfully ignore. * * A similar approach is used for labels.
49614. The act->prev_caller should only be set if the entry for 'caller' * exists (as it is only set in that case, and the property is not * configurable), but handle all the cases anyway.
49615. break/continue without label
49616. **comment:** XXX: the best typing needs to be validated by perf measurement: * e.g. using a small type which is the cast to a larger duk_idx_t * may be slower than declaring the variable as a duk_idx_t in the * first place.
label: code-design
49617. **comment:** XXX: conversion errors should not propagate outwards. * Perhaps values need to be coerced individually?
label: code-design
49618. XXX: here we need to know if 'left' is left-hand-side compatible. * That information is no longer available from current expr parsing * state; it would need to be carried into the 'left' ivalue or by * some other means.
49619. non-global regexp: lastIndex never updated on match
49620. Note: undefined from Section 11.8.5 always results in false * return (see e.g. Section 11.8.3) - hence special treatment here.
49621. If there's no catch block, rc_varname will be 0 and duk__patch_trycatch() * will replace the LDCONST with a NOP. For any actual constant (including * constant 0) the DUK__CONST_MARKER flag will be set in rc_varname.
49622. exposed because lexer needs these too
49623. empty match (may happen with empty separator) -> bump and continue
49624. As with all inspection code, we rely on the debug client providing * a valid, non-stale pointer: there's no portable way to safely * validate the pointer here.
49625. Set up curr_pc for opcode dispatch.
49626. [... pattern flags escaped_source]
49627. **comment:** The 'magic' field allows an opaque 16-bit field to be accessed by the * Duktape/C function. This allows, for instance, the same native function * to be used for a set of very similar functions, with the 'magic' field * providing the necessary non-argument flags / values to guide the behavior * of the native function. The value is signed on purpose: it is easier to * convert a signed value to unsigned (simply AND with 0xffff) than vice * versa. * * Note: cannot place nargs/magic into the heapdr flags, because * duk_hobject takes almost all flags already (and needs the spare).
label: code-design
49628. Get minimum entry part growth for a certain size.
49629. valstack index of start of args (arg1) (relative to entry valstack_bottom)
49630. [ToObject(this) item1 ... itemN arr item(i) item(i)[j]]
49631. thr->heap->lj.value2 is 'thread', will be wiped out at the end
49632. break/continue with label (label cannot be a reserved word, production is 'Identifier'
49633. property access
49634. Coerce all finite parts with ToInteger(). ToInteger() must not * be called for NaN/Infinity because it will convert e.g. NaN to * zero. If ToInteger() has already been called, this has no side * effects and is idempotent. * * Don't read dparts[DUK_DATE_IDX_WEEKDAY]; it will cause Valgrind * issues if the value is uninitialized.
49635. internal property functions
49636. Report thrown value to client coerced to string
49637. **comment:** ArrayBuffer argument is handled specially above; the rest of the * argument variants are handled by shared code below.
label: code-design
49638. DUK_USE_USER_INITJS
49639. Range limited to [0, 0x7fffffff] range, i.e. range that can be * represented with duk_int32_t. Use this when the method doesn't * handle the full 32-bit unsigned range correctly.
49640. 110x xxxx 10xx xxxx
49641. Sanity limits for stack sizes.

49642. **comment:** Linear scan: more likely because most objects are small. * This is an important fast path. ** XXX: this might be worth inlining for property lookups.
label: code-design

49643. XXX: will need a force flag if garbage collection is triggered * explicitly during paused state.

49644. Array length is larger than 'asize'. This shouldn't * happen in practice. Bail out just in case.

49645. No net refcount change.

49646. * Assert context is valid: non-NULL pointer, fields look sane. ** This is used by public API call entrypoints to catch invalid 'ctx' pointers * as early as possible; invalid 'ctx' pointers cause very odd and difficult to * diagnose behavior so it's worth checking even when the check is not 100%.

49647. * Setup value stack: clamp to 'nargs', fill up to 'nregs'

49648. This should not happen because DUK_TAG_OBJECT case checks * for this already, but check just in case.

49649. result array is already at the top of stack

49650. args go here as a comma expression in parens

49651. [...] source? filename? &comp_args] (depends on flags)

49652. no extra padding

49653. Start position (inclusive) and end position (exclusive)

49654. tentative, checked later

49655. slot A is a source (default: target)

49656. indexed by recursion_depth

49657. * Fixed buffer helper useful for debugging, requires no allocation * which is critical for debugging.

49658. Order of unwinding is important

49659. * Final sigma context specific rule. This is a rather tricky * rule and this handling is probably not 100% correct now. * The rule is not locale/language specific so it is supported.

49660. tracks maximum initialized index + 1

49661. **comment:** first register that is a temporary (below: variables)
label: code-design

49662. A -> register of target object * B -> first register of value data (start_index, value1, value2, ..., valueN) * C -> number of key/value pairs (N)

49663. Just skip, leaving zeroes in the result.

49664. [...] arr]

49665. Note: successive characters could be joined into string matches * but this is not trivial (consider e.g. '/xyz+'); see docs for * more discussion.

49666. Coerce an duk_alue to a register or constant; result register may * be a temp or a bound register. ** The duk_alue argument ('x') is converted into a regconst as a * side effect.

49667. bytes in source

49668. no voluntary gc

49669. A -> result reg * B -> object reg * C -> key reg/const

49670. An object may have FINALIZED here if it was finalized by mark-and-sweep * on a previous run and refcount then decreased to zero. We won't run the * finalizer again here.

49671. **comment:** XXX: allow object to be a const, e.g. in 'foo'.toString()? * On the other hand, DUK_REGCONSTP() is slower and generates * more code.
label: code-design

49672. refcount

49673. regexp res_obj is at index 4

49674. stack[0] = callback fn * stack[1] = initialValue * stack[2] = object (coerced this) * stack[3] = length (not needed, but not popped above) * stack[4] = accumulator

49675. **comment:** Node.js Buffer variable width integer field. We don't really * care about speed here, so aim for shortest algorithm.
label: code-design

49676. The spec algorithm first does "R = ToString(separator)" before checking * whether separator is undefined. Since this is side effect free, we can * skip the ToString() here.

49677.]'

49678. NULL with zero length represents an empty string; NULL with higher * length is also now treated like an empty string although it is * a bit dubious. This is unlike duk_push_string() which pushes a * 'null' if the input string is a NULL.

49679. -> [func funcname env funcname func]

49680. true, because v[1] has at least one bit set

49681. bp to use when parsing a top level Expression

49682. Caller will finish the marking process if we hit a recursion limit.

49683. y

49684. **comment:** * E5 Section 7.6: ** IdentifierStart: * UnicodeLetter * \$ * _ * \ UnicodeEscapeSequence ** IdentifierStart production has one multi-character production: ** \ UnicodeEscapeSequence ** The '\ character is -not- matched by this function. Rather, the caller * should decode the escape and then call this function to check whether the * decoded character is acceptable (see discussion in E5 Section 7.6). ** The "UnicodeLetter" alternative of the production allows letters * from various Unicode categories. These can be extracted with the * 'src/extract_chars.py' script. ** Because the result has hundreds of Unicode codepoint ranges, matching * for any values >= 0x80 are done using a very slow range-by-range scan * and a packed range format. ** The ASCII portion (codepoints 0x00 ... 0x7f) is fast-patched below because * it matters the most. The ASCII related ranges of IdentifierStart are: ** 0x0041 ... 0x005a ['A' ... 'Z'] * 0x0061 ... 0x007a ['a' ... 'z'] * 0x0024 ['\$'] * 0x005f['_']
label: code-design

49685. Must be an object, otherwise TypeError (E5.1 Section 8.10.5, step 1).

49686. TypeError if fails

49687. * Entries part

49688. * Delayed env creation check

49689. insert 'undefined' values at idx_rcbase to get the * return values to idx_rebase

49690. xxx -> DUK_HBUFFEROBJECT_ELEM_INT8

49691. guarantees entry_callstack_top - 1 >= 0

49692. no code needs to be emitted, the regs already have values

49693. magic: 0=getter call, 1=Object.getPrototypeOf()

49694. PC is unsigned. If caller does PC arithmetic and gets a negative result, * it will map to a large PC which is out of bounds and causes a zero to be * returned.

49695. * Debug connection write primitives

49696. **comment:** XXX: join these ops (multiply-accumulate), but only if * code footprint decreases.
label: code-design

49697. module.name for .name, default to last component if * not present.

49698. * Exposed debug macros: debugging enabled

49699. Object.defineProperty() calls [[DefineOwnProperty]] with Throw=true

49700. 0xe0-0xef

49701. compact the prototype

49702. XXX: default priority for infix operators is duk_expr_lbp(tok) -> get it here?

49703. read header

49704. prop index in 'hash part', < 0 if not there

49705. This is based on Rhino EquivalentYear() algorithm: *
<https://github.com/mozilla/rhino/blob/f99cc11d616f0cdda2c42bde72b3484df6182947/src/org.mozilla/javascript/NativeDate.java>

49706. **comment:** Clamping to zero makes the API more robust to calling code * calculation errors.
label: code-design

49707. Surround with parentheses like in JX, ensures NULL pointer * is distinguishable from null value ("(null)" vs "null").

49708. Attempt to write 'stack', 'fileName', 'lineNumber' works as if * user code called Object.defineProperty() to create an overriding * own property. This allows user code to overwrite .fileName etc * intuitively as e.g. "err.fileName = 'dummy'" as one might expect. * See <https://github.com/svaarala/duktape/issues/387>.

49709. object is a native function (duk_hnativefunction)
49710. [... error func fileName lineNumber]
49711. 21 bits
49712. **comment:** XXX: thread selection for mark-and-sweep is currently a hack. * If we don't have a thread, the entire mark-and-sweep is now * skipped (although we could just skip finalizations).
label: code-design
49713. embed: duk_hstring ptr
49714. enumeration
49715. * Free memory
49716. Maximum value check ensures 'nbytes' won't wrap below.
49717. Use existing env (e.g. for non-strict eval); cannot have * an own 'arguments' object (but can refer to an existing one).
49718. currently, even for Array.prototype
49719. unsigned intentionally
49720. DUK_TOK_SNEQ
49721. only outer_lex_env matters, as functions always get a new * variable declaration environment.
49722. No need to assert, buffer size maximum is 0xffff.
49723. default is explicit index read/write copy
49724. not enumerable
49725. * Exposed data
49726. avoid problems if p == h->prototype
49727. DUK_USE_PROVIDE_DEFAULT_ALLOC_FUNCTIONS
49728. Output shuffle needed after main operation
49729. For native calls must be NULL so we don't sync back
49730. **comment:** XXX: very basic optimization -> duk_get_prop_stridx_top
label: code-design
49731. omitted
49732. toGMTString'
49733. * Object related * * Note: seal() and freeze() are accessible through Ecmascript bindings, * and are not exposed through the API.
49734. * ISO 8601 subset parser.
49735. Internal keys are prefixed with 0xFF in the stringtable * (which makes them invalid UTF-8 on purpose).
49736. without explicit non-BMP support, assume non-BMP characters * are always accepted as identifier characters.
49737. [res]
49738. dash is already 0
49739. Object flag. There are currently 26 flag bits available. Make sure * this stays in sync with debugger object inspection code.
49740. This may only happen if built-ins are being "torn down". * This behavior is out of specification scope.
49741. Not possible because array object 'length' is present * from its creation and cannot be deleted, and is thus * caught as an existing property above.
49742. Note: strictness is not inherited from the current Duktape/C * context. Otherwise it would not be possible to compile * non-strict code inside a Duktape/C activation (which is * always strict now). See tests/api/test-eval-strictness.c * for discussion.
49743. A -> flags * B -> base register for call (base -> func, base+1 -> this, base+2 -> arg1 ... base+2+N-1 -> argN) * (for DUK_OP_CALLI, 'b' is indirect) * C -> nargs
49744. * Wrapper for jmp_buf. * * This is used because jmp_buf is an array type for backward compatibility. * Wrapping jmp_buf in a struct makes pointer references, sizeof, etc, * behave more intuitively. * * http://en.wikipedia.org/wiki/Setjmp.h#Member_types
49745. * This is a bit tricky to implement portably. The result depends * on the timestamp (specifically, DST depends on the timestamp). * If e.g. UNIX APIs are used, they'll have portability issues with * very small and very large years. * * Current approach: * * - Stay within portable UNIX limits by using equivalent year mapping. * Avoid year 1970 and 2038 as some conversions start to fail, at * least on some platforms. Avoiding 1970 means that there are * currently DST discrepancies for 1970. * * - Create a UTC and local time breakdowns from 't'. Then create * a time_t using gmtime() and localtime() and compute the time * difference between the two. * * Equivalent year mapping (E5 Section 15.9.1.8): * * If the host environment provides functionality for determining * daylight saving time, the implementation of ECMAScript is free * to map the year in question to an equivalent year (same * leap-year-ness and same starting week day for the year) for which * the host environment provides daylight saving time information. * The only restriction is that all equivalent years should produce * the same result. * * This approach is quite reasonable but not entirely correct, e.g. * the specification also states (E5 Section 15.9.1.8): * * The implementation of ECMAScript should not try to determine * whether the exact time was subject to daylight saving time, but * just whether daylight saving time would have been in effect if * the _current daylight saving time algorithm_ had been used at the * time. This avoids complications such as taking into account the * years that the locale observed daylight saving time year round. * * Since we rely on the platform APIs for conversions between local * time and UTC, we can't guarantee the above. Rather, if the platform * has historical DST rules they will be applied. This seems to be the * general preferred direction in Ecmascript standardization (or at least * implementations) anyway, and even the equivalent year mapping should * be disabled if the platform is known to handle DST properly for the * full Ecmascript range. * * The following has useful discussion and links: * * https://bugzilla.mozilla.org/show_bug.cgi?id=351066
49746. keep val_lowest
49747. **comment:** XXX: limit to quoted strings only, to save keys from being cluttered?
label: code-design
49748. DUK_USE_DEBUGGER_THROW_NOTIFY || DUK_USE_DEBUGGER_PAUSE_UNCAUGHT
49749. can't get value, may be accessor
49750. Relookup and initialize dispatch loop variables. Debugger check.
49751. * Longjmp state, contains the information needed to perform a longjmp. * Longjmp related values are written to value1, value2, and iserror.
49752. leave result on stack top
49753. Resolved, normalized absolute module ID
49754. marker; not actual tagged value
49755. re-lookup curr char on first round
49756. [... formals arguments map mappedNames]
49757. borrowed: full duk_rval for function being executed; for lightfuncs
49758. Shared handler to minimize parser size. Cause will be * hidden, unfortunately, but we'll have an offset which * is often quite enough.
49759. Skip dvalue.
49760. **comment:** XXX: inefficient; block remove primitive
label: code-design
49761. If duk_ivalue_toplain_raw() allocates a temp, forget it and * restore next temp state.
49762. DUK_HOBJECT_FLAG_CONSTRUCTABLE varies
49763. [... pattern flags]
49764. custom; implies DUK_HOBJECT_IS_THREAD
49765. [thisArg arg1 ... argN]
49766. for array and object literals
49767. validation of the regexp is caller's responsibility
49768. regnum is sane
49769. * To avoid creating a heavy intermediate value for the list of ranges, * only the start token ('[or '[^') is parsed here. The regexp * compiler parses the ranges itself.
49770. **comment:** XXX: Currently function source code is not stored, as it is not * required by the standard. Source code should not be stored by * default (user should enable it explicitly), and the source should * probably be compressed with a trivial text compressor; average * compression of 20-30% is quite easy to achieve even with a trivial * compressor (RLE + backwards lookup). * * Debugging needs source code to be useful: sometimes input code is * not found in files as it may be generated and then eval()'d, given * by dynamic C code, etc. * * Other issues: * * - Need tokenizer indices for start and end to substring * - Always normalize function declaration part? * - If we keep _Formals, only need to store body
label: code-design
49771. Preinc/predec for var-by-name, slow path.

49772. tm_isdst is both an input and an output to mktime(), use 0 to * avoid DST handling in mktime(): * - https://github.com/svaarala/duktape/issues/406 * - http://stackoverflow.com/questions/8558919/mktime-and-tm-isdst

49773. Small variant; roughly 150 bytes smaller than the fast variant.

49774. -> [... closure template env funcname]

49775. * Free the heap. * * Frees heap-related non-heap-tracked allocations such as the * string intern table; then frees the heap allocated objects; * and finally frees the heap structure itself. Reference counts * and GC markers are ignored (and not updated) in this process, * and finalizers won't be called. * * The heap pointer and heap object pointers must not be used * after this call.

49776. rescue no longer supported

49777. res->mark_and_sweep_trigger_counter == 0 -> now causes immediate GC; which is OK

49778. _Pc2line

49779. Sanity checks for string and token defines

49780. number of escapes and line continuations (for directive prologue)

49781. MakeTime

49782. **comment:** XXX: this is possible without resorting to the value stack
label: code-design

49783. ignored

49784. NULL for lightfunc

49785. lookup shorthands (note: assume context variable is named 'lex_ctxt')

49786. Other callbacks are optional.

49787. truncated in case pass 3 needed

49788. With constants and inner functions on value stack, we can now * atomically finish the function 'data' buffer, bump refcounts, * etc. * * Here we take advantage of the value stack being just a duk_tval * array: we can just memcpy() the constants as long as we incref * them afterwards.

49789. DUK_USE_NONSTD_FUNC_SOURCE_PROPERTY

49790. Leave 'getter' on stack

49791. Called as a function, pattern has [[Class]] "RegExp" and * flags is undefined -> return object as is.

49792. Q...f

49793. x < y

49794. set exotic behavior only after we're done

49795. **comment:** Default function to write a formatted log line. Writes to stderr, * appending a newline to the log line. * * The argument is a buffer whose visible size contains the log message. * This function should avoid coercing the buffer to a string to avoid * string table traffic.
label: code-design

49796. Current size is about 152 bytes.

49797. **comment:** These base values are never used, but if the compiler doesn't know * that DUK_ERROR() won't return, these are needed to silence warnings. * On the other hand, scan-build will warn about the values not being * used, so add a DUK_UNREF.
label: code-design

49798. hash size relative to entries size: for value X, approx. hash_prime(e_size + e_size / X)

49799. while repl

49800. Note: we don't parse back exponent notation for anything else * than radix 10, so this is not an ambiguous check (e.g. hex * exponent values may have 'e' either as a significand digit * or as an exponent separator). * * If the exponent separator occurs twice, 'e' will be interpreted * as a digit (= 14) and will be rejected as an invalid decimal * digit.

49801. DUK_HEAPHDR_HAS_FINALIZED may be set if we're doing a * refzero finalization and mark-and-sweep gets triggered * during the finalizer.

49802. Setup state. Initial ivalue is 'undefined'.

49803. 'object'

49804. If we processed any debug messages breakpoints may have * changed; restart execution to re-check active breakpoints.

49805. 1 1 1 <32 bits>

49806. **comment:** Leading / trailing whitespace is sometimes accepted and * sometimes not. After white space trimming, all valid input * characters are pure ASCII.
label: code-design

49807. patch in range count later

49808. * If tracebacks are disabled, 'fileName' and 'lineNumber' are added * as plain own properties. Since Error.prototype has accessors of * the same name, we need to define own properties directly (cannot * just use e.g. duk_put_prop_stridx). Existing properties are not * overwritten in case they already exist.

49809. 0: toString

49810. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small.
label: code-design

49811. UTF-8 encoded bytes escaped as %xx%xx%xx... -> 3 * nbytes. * Codepoint range is restricted so this is a slightly too large * but doesn't matter.

49812. lowest mantissa for this exponent

49813. * Traceback handling when tracebacks disabled. * * The fileName / lineNumber stubs are now necessary because built-in * data will include the accessor properties in Error.prototype. If those * are removed for builds without tracebacks, these can also be removed. * 'stack' should still be present and produce a ToString() equivalent: * this is useful for user code which prints a stacktrace and expects to * see something useful. A normal stacktrace also begins with a ToString() * of the error so this makes sense.

49814. all we need to know

49815. ASSIGNMENT EXPRESSION

49816. IdentityEscape

49817. Must decrease recursion depth before returning.

49818. note: mixing len into seed improves hashing when skipping

49819. For join(), nargs is 1. For toLocaleString(), nargs is 0 and * setting the top essentially pushes an undefined to the stack, * thus defaulting to a comma separator.

49820. Return with final function pushed on stack top.

49821. Note: 'e' and 'E' are also accepted here.

49822. %c', passed concretely as int

49823. thread resumed another thread (active but not running)

49824. or a non-catching entry

49825. next char

49826. -> [... key val]

49827. add F

49828. **comment:** return a specific NaN (although not strictly necessary)
label: code-design

49829. * Variant 1

49830. global regexp: lastIndex updated on match

49831. **comment:** pre-check how many atom copies we're willing to make (atom_copies not needed below)
label: requirement

49832. Current size (not counting a dynamic buffer's "spare").

49833. +2 = catcher value, catcher lj_type

49834. parsing in "directive prologue", recognize directives

49835. non-strict: non-deletable, writable

49836. DUK_USE_64BIT_OPS

49837. simulate alloc failure on every realloc (except when mark-and-sweep is running)

49838. 7

49839. **comment:** alloc temp just in case, to update max temp
label: code-design

49840. * An array must have a 'length' property (E5 Section 15.4.5.2). * The special array behavior flag must only be enabled once the * length property has been added. *
* The internal property must be a number (and preferably a * fastint if fastint support is enabled).

49841. * Create a hstring and insert into the heap. The created object * is directly garbage collectable with reference count zero. * * The caller must place the interned string into the stringtable * immediately (without chance of a longjmp); otherwise the string * is lost.

49842. Note: this is correct even for default clause statements: * they participate in 'fall-through' behavior even if the * default clause is in the middle.

49843. exposed, used by e.g. duk.bi_date.c

49844. * Parsing of ints and floats

49845. flags: "gi"

49846. For TypedArrays 'undefined' return value is specified * by ES6 (matches V8).

49847. * Return target object

49848. side effects

49849. [... result]

49850. only objects have finalizers

49851. interpret e.g. '0x' and '0xg' as a NaN (= parse error)

49852. **comment:** * Get and call the finalizer. All of this must be wrapped * in a protected call, because even getting the finalizer * may trigger an error (getter may throw one, for instance).
label: code-design

49853. **comment:** * String/JSON conversions * * Human readable conversions are now basically ISO 8601 with a space * (instead of 'T') as the date/time separator. This is a good baseline * and is platform independent. * * A shared native helper to provide many conversions. Magic value contains * a set of flags. The helper provides: * * toString() * toDateString() * toTimeString() * toLocaleString() * toLocaleDateString() * toLocaleTimeString() * toUTCString() * toISOString() * * Notes: * * - Date.prototype.toGMTString() and Date.prototype.toUTCString() are * required to be the same Ecmascript function object (!), so it is * omitted from here. * * - Date.prototype.toUTCString(): E5.1 specification does not require a * specific format, but result should be human readable. The * specification suggests using ISO 8601 format with a space (instead of 'T') separator if a more human readable format is not available. * * - Date.prototype.toISOString(): unlike other conversion functions, * toISOString() requires a RangeError for invalid date values.
label: code-design

49854. * Thread support.

49855. bytecode start offset of the atom parsed in this loop * (allows quantifiers to copy the atom bytecode)

49856. temp copy, write back for next loop

49857. char loop

49858. basic platform types

49859. we currently assume virtual properties are not configurable (as none of them are)

49860. Stage 1: find highest preventing non-configurable entry (if any). * When forcing, ignore non-configurability.

49861. **comment:** XXX: keep property attributes or tweak them here? * Properties will now be non-configurable even when they're * normally configurable for the global object.
label: code-design

49862. address

49863. * Property already exists. Steps 5-6 detect whether any changes need * to be made.

49864. may be NULL

49865. * Parse Ecmascript source InputElementDiv or InputElementRegExp * (E5 Section 7), skipping whitespace, comments, and line terminators. * * Possible results are: * (1) a token * (2) a line terminator (skipped) * (3) a comment (skipped) * (4) EOF * * White space is automatically skipped from the current position (but * not after the input element). If input has already ended, returns * DUK_TOK_EOF indefinitely. If a parse error occurs, uses an DUK_ERROR() * macro call (and hence a longjmp through current heap longjmp context). * Comments and line terminator tokens are automatically skipped. * * The input element being matched is determined by regexp_mode; if set, * parses a InputElementRegExp, otherwise a InputElementDiv. The * difference between these are handling of productions starting with a * forward slash. * * If strict_mode is set, recognizes additional future reserved words * specific to strict mode, and refuses to parse octal literals. * * The matching strategy below is to (currently) use a six character * lookup window to quickly determine which production is the -longest- * matching one, and then parse that. The top-level if-else clauses * match the first character, and the code blocks for each clause * handle -all- alternatives for that first character. Ecmascript * specification uses the "longest match wins" semantics, so the order * of the if-clauses matters. * * Misc notes: * * * Ecmascript numeric literals do not accept a sign character. * Consequently e.g. "-1.0" is parsed as two tokens: a negative * sign and a positive numeric literal. The compiler performs * the negation during compilation, so this has no adverse impact. * * * There is no token for "undefined": it is just a value available * from the global object (or simply established by doing a reference * to an undefined value). * * * Some contexts want Identifier tokens, which are IdentifierNames * excluding reserved words, while some contexts want IdentifierNames * directly. In the latter case e.g. "while" is interpreted as an * identifier name, not a DUK_TOK WHILE token. The solution here is * to provide both token types: DUK_TOK WHILE goes to 't' while * DUK_TOK_IDENTIFIER goes to 't_nores', and 'slot1' always contains * the identifier / keyword name. * * * Directive prologue needs to identify string literals such as * "use strict" and 'use strict', which are sensitive to line * continuations and escape sequences. For instance, "use\0020strict" * is a valid directive but is distinct from "use strict". The solution * here is to decode escapes while tokenizing, but to keep track of the * number of escapes. Directive detection can then check that the * number of escapes is zero. * * * Multi-line comments with one or more internal LineTerminator are * treated like a line terminator to comply with automatic semicolon * insertion.

49866. IEEE double is approximately 16 decimal digits; print a couple extra

49867. * Resizing and hash behavior

49868. String sanitizer which escapes ASCII control characters and a few other * ASCII characters, passes Unicode as is, and replaces invalid UTF-8 with * question marks. No errors are thrown for any input string, except in out * of memory situations.

49869. **comment:** Try to optimize X <op>= Y for reg-bound * variables. Detect side-effect free RHS * narrowly by seeing whether it emits code. * If not, rewind the code emitter and overwrite * the unnecessary temp reg load.
label: code-design

49870. [obj key undefined]

49871. **comment:** XXX: can be optimized for smaller footprint esp. on 32-bit environments
label: code-design

49872. * isArray()

49873. DUK_USE_HOBJECT_HASH_PART

49874. trailing elisions?

49875. * Simple commands

49876. XXX: The TypeError is currently not applied to bound * functions because the 'strict' flag is not copied by * bind(). This may or may not be correct, the specification * only refers to the value being a "strict mode Function * object" which is ambiguous.

49877. [... global val] -> [... global]

49878. length in codepoints (must be E5 compatible)

49879. The smallest fastint is no longer 48-bit when * negated. Positive zero becomes negative zero * (cannot be represented) when negated.

49880. Decode helper. Return zero on error.

49881. XXX: specify array size, as we know it

49882. **comment:** Numbers are normalized to big (network) endian. We can * (but are not required) to use integer dvalues when there's * no loss of precision. * * XXX: share check with other code; this check is slow but * reliable and doesn't require careful exponent/mantissa * mask tricks as in the fastint downgrade code.
label: code-design

49883. * Use static helpers which can work with math.h functions matching * the following signatures. This is not portable if any of these math * functions is actually a macro. * * Typing here is intentionally 'double' wherever values interact with * the standard library APIs.

49884. 1111 0xxx; 4 bytes

49885. start array index of current MPUTARR set

49886. cp == -1 (EOF) never matches and causes return value 0

49887. **comment:** XXX: the handling of character range detection is a bit convoluted. * Try to simplify and make smaller.
label: code-design

49888. DUK_OP_CALL flags in A

49889. * Shared helper for non-bound func lookup. ** Returns duk_hobject * to the final non-bound function (NULL for lightfunc).
49890. Must be ≥ 0 and multiple of element size.
49891. A stubbed built-in is useful for e.g. compilation torture testing with BCC.
49892. Decode a plain string consisting entirely of identifier characters. * Used to parse plain keys (e.g. "foo: 123").
49893. note that we can't reliably pop anything here
49894. [... put_value]
49895. 5
49896. ref.value and ref.this_binding invalidated here
49897. nret
49898. object is a compiled function (duk_hcompiledfunction)
49899. Push a new closure on the stack. ** Note: if fun_temp has NEWENV, i.e. a new lexical and variable declaration * is created when the function is called, only outer_lex_env matters * (outer_var_env is ignored and may or may not be same as outer_lex_env).
49900. Need a short reg/const, does not have to be a mutable temp.
49901. delete Duktape.modLoaded[resolved_id]
49902. keep current valstack_top
49903. Disabled until fixed, see above.
49904. 'message'
49905. Just transfer the refcount from act->prev_caller to tv_caller, * so no need for a refcount update. This is the expected case.
49906. Suggested step-by-step method from documentation of RtTimeToSecondsSince1970: * [http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724928(v=vs.85).aspx)
49907. * Migrate array to start of entries if requested. ** Note: from an enumeration perspective the order of entry keys matters. * Array keys should appear wherever they appeared before the array abandon * operation.
49908. 0x70...0x7f
49909. Maximum iteration count for computing UTC-to-local time offset when * creating an Ecmascript time value from local parts.
49910. **comment:** XXX: depend on available temps?
 label: code-design
49911. default case
49912. %u; only 16 bits are guaranteed
49913. MultiplicativeExpression
49914. Trim white space (= allow leading and trailing whitespace)
49915. * Call user provided module search function and build the wrapped * module source code (if necessary). The module search function * can be used to implement pure Ecmascript, pure C, and mixed * Ecmascript/C modules. ** The module search function can operate on the exports table directly * (e.g. DLL code can register values to it). It can also return a * string which is interpreted as module source code (if a non-string * is returned the module is assumed to be a pure C one). If a module * cannot be found, an error must be thrown by the user callback. ** Because Duktape.modLoaded[] already contains the module being * loaded, circular references for C modules should also work * (although expected to be quite rare).
49916. One particular problem case is where an object has been * queued for finalization but the finalizer hasn't yet been * executed. ** Corner case: we're running in a finalizer for object X, and * user code calls duk_push_heapptr() for X itself. In this * case X will be in finalize_list, and we can detect the case * by seeing that X's FINALIZED flag is set (which is done before * the finalizer starts executing).
49917. does not modify tv_x
49918. 26: setSeconds
49919. patch pending jump and split
49920. traceback depth doesn't take into account the filename/line * special handling above (intentional)
49921. DUK_USE_ERRTHROW || DUK_USE_ERRCREATE
49922. only needed by debugger for now
49923. Compute day number of the first day of a given year.
49924. ignore_loop
49925. * Begin
49926. Output #1: resolved absolute name
49927. and even number
49928. * DELPROP: Ecmascript property deletion.
49929. normal: implicit leading 1-bit
49930. may throw an error
49931. * Encoding constants, must match genbuiltins.py
49932. 'RegExp'
49933. **comment:** XXX: do something to force a callstack grow/shrink, perhaps * just a manual forced resize?
 label: code-design
49934. <<
49935. **comment:** XXX: double evaluation of DUK_HCOMPILEDFUNCTION_GET_DATA()
 label: code-design
49936. ensure full memcmp() fits in while
49937. unreachable
49938. push operation normalizes NaNs
49939. getter/setter
49940. x == 0x00 (EOF) causes syntax_error
49941. XXX: range assert
49942. **comment:** XXX: move masks to js_ctx? they don't change during one * fast path invocation.
 label: code-design
49943. flags: "i"
49944. * Init stringtable: probe variant
49945. Catches both doubles and cases where only one argument is a fastint
49946. key coercion (unless already coerced above)
49947. **comment:** XXX: There's quite a bit of overlap with buffer creation handling in * duk_bi_buffer.c. Look for overlap and refactor.
 label: code-design
49948. Limit checks for bytecode byte size and line number.
49949. The necessary #includes are in place in duk_config.h.
49950. **comment:** XXX: this could be optimized
 label: code-design
49951. duk_handle_return() is guaranteed never to throw, except * for potential out-of-memory situations which will then * propagate out of the executor longjmp handler.
49952. **comment:** * XXX: if duk_handle_call() took values through indices, this could be * made much more sensible. However, duk_handle_call() needs to fudge * the 'this' and 'func' values to handle bound function chains, which * is now done "in-place", so this is not a trivial change.
 label: code-design
49953. Number serialization has a significant impact relative to * other fast path code, so careful fast path for fastints.
49954. * Compiler intermediate values ** Intermediate values describe either plain values (e.g. strings or * numbers) or binary operations which have not yet been coerced into * either a left-hand-side or right-hand-side role (e.g. object property).
49955. **comment:** Silence a few global unused warnings here.
 label: code-design
49956. -> [...] arr num]

49957. tzoffset seconds are dropped; 16 bits suffice for * time offset in minutes
49958. thread has yielded
49959. Lightfuncs are always strict.
49960. '\xffArgs'
49961. **comment:** XXX: for array instances we could take a shortcut here and assume * Array.prototype doesn't contain an array index property.
 label: code-design
49962. PropertyName -> IdentifierName | StringLiteral | NumericLiteral
49963. -1 = top callstack entry, callstack[callstack_top - 1] * -callstack_top = bottom callstack entry, callstack[0]
49964. Further state-dependent pre-checks
49965. E5 Section 8.12.8
49966. is_decl
49967. at least one activation, ours
49968. s < b^(~e) * 2
49969. use p_src_base from now on
49970. 'taint' result as complex
49971. no issues with memcmp() zero size, even if broken
49972. * Disjunction struct: result of parsing a disjunction
49973. * Emit initializers in sets of maximum max_init_pairs keys. * Setter/getter is handled separately and terminates the * current set of initializer values. Corner cases such as * single value initializers do not have special handling now.
49974. * E5 Section 7.3: CR LF is detected as a single line terminator for * line numbers. Here we also detect it as a single line terminator * token.
49975. We don't need to sync back thr->ptr_curr_pc here because * the bytecode executor always has a setjmp catchpoint which * does that before errors propagate to here.
49976. Previous entry was inside visited[], nothing to do.
49977. **comment:** XXX: optimize temp reg use
 label: code-design
49978.toFixed()
49979. Detect zero special case.
49980. Number and (minimum) size of bigints in the nc_ctx structure.
49981. We come here for actual aborts (like encountering . toJSON()) * but also for recursion/loop errors. Bufwriter size can be * kept because we'll probably need at least as much as we've * allocated so far.
49982. * Preliminaries: trim, sign, Infinity check * * We rely on the interned string having a NUL terminator, which will * cause a parse failure wherever it is encountered. As a result, we * don't need separate pointer checks. * * There is no special parsing for 'NaN' in the specification although * 'Infinity' (with an optional sign) is allowed in some contexts. * Some contexts allow plus/minus sign, while others only allow the * minus sign (like JSON.parse()). * * Automatic hex number detection (leading '0x' or '0X') and octal * number detection (leading '0' followed by at least one octal digit) * is done here too.
49983. duk_unicode_idp_m_ids_noabmp[]
49984. '(?:)'
49985. * Process replacer/proplist (2nd argument to JSON.stringify)
49986. may be a string constant
49987. return total chars written excluding terminator
49988. [(builtin objects) name]
49989. new_free / size <= 1 / DIV <=> new_free <= size / DIV
49990. This shouldn't happen; call sites should avoid looking up * _Finalizer "through" a Proxy, but ignore if we come here * with a Proxy to avoid finalizer re-entry.
49991. [... holder name val enum obj_key]
49992. * Create a new function object based on a "template function" which contains * compiled bytecode, constants, etc, but lacks a lexical environment. * * ECMAScript requires that each created closure is a separate object, with * its own set of editable properties. However, structured property values * (such as the formal arguments list and the variable map) are shared. * Also the bytecode, constants, and inner functions are shared. * * See E5 Section 13.2 for detailed requirements on the function objects; * there are no similar requirements for function "templates" which are an * implementation dependent internal feature. Also see function-objects.rst * for a discussion on the function instance properties provided by this * implementation. * * Notes: * * * Order of internal properties should match frequency of use, since the * properties will be linearly scanned on lookup (functions usually don't * have enough properties to warrant a hash part). * * * The created closure is independent of its template; they do share the * same 'data' buffer object, but the template object itself can be freed * even if the closure object remains reachable.
49993. E5 Section 11.13.1 (and others) step 4 never matches for prop writes -> no check
49994. exit() afterwards to satisfy "noreturn"
49995. string compare is the default (a bit oddly)
49996. '%xx%xx...%xx', p points to char after first '%'
49997. **comment:** XXX: need a toplain_ignore() which will only coerce a value to a temp * register if it might have a side effect. Side-effect free values do not * need to be coerced.
 label: code-design
49998. 2^4 -> 1/16 = 6.25% spare
49999. * reduce(), reduceRight()
50000. * Macros for accessing size fields
50001. * Resolve module identifier into canonical absolute form.
50002. leap year: sunday, monday, ...
50003. even after a detach and possible reattach
50004. tval
50005. **comment:** integer mixed endian not really used now
 label: code-design
50006. only advance if not tainted
50007. tail insert: don't disturb head in case refzero is running
50008. indexOf: NaN should cause pos to be zero. * lastIndexOf: NaN should cause pos to be +Infinity * (and later be clamped to len).
50009. 0.000...
50010. leave stack unbalanced on purpose
50011. [... func this <bound args> arg1 ... argN]
50012. Bitfield for each DUK_HBUFFEROBJECT_ELEM_xxx indicating which element types * are compatible with a blind byte copy for the TypedArray set() method (also * used for TypedArray constructor). Array index is target buffer elem type, * bitfield indicates compatible source types. The types must have same byte * size and they must be coercion compatible.
50013. **comment:** Run fake finalizer. Avoid creating unnecessary garbage.
 label: code-design
50014. * Table for hex encoding bytes
50015. [... buf func] -> [... func]
50016. build result as: (r << 32) + s: start with (BD + E + F)
50017. We only get here when doing non-standard JSON encoding
50018. Without heap pointer compression duk_hbuffer_dynamic and duk_hbuffer_external * have the same layout so checking for fixed vs. dynamic (or external) is enough.
50019. [target] -> [enum]
50020. Handle both full and partial slice (as long as covered).
50021. envrec: (declarative) record is closed

50022. **comment:** Log frontend shared helper, magic value indicates log level. Provides * frontend functions: trace(), debug(), info(), warn(), error(), fatal(). * This needs to have small footprint, reasonable performance, minimal * memory churn, etc.
label: code-design

50023. XXX: bump preventcount by one for the duration of this call?

50024. for

50025. 15: getUTCDay

50026. need_bytes may be zero

50027. register binding lookup is based on varmap (even in first pass)

50028. prototype: the only internal property lifted outside 'e' as it is so central

50029. 3 entries actually needed below

50030. end of input and last char has been processed

50031. [start length str]

50032. array

50033. fast paths for space and tab

50034. Significand ('f') padding.

50035. **comment:** not needed
label: requirement

50036. Regardless of whether property is found in entry or array part, * it may have arguments exotic behavior (array indices may reside * in entry part for abandoned / non-existent array parts).

50037. add a new pending match jump for latest finished alternative

50038. exclusive

50039. DUK_USE_BUILTIN_INITJS

50040. * If selftests enabled, run them as early as possible

50041. ivalue/ispec helpers

50042. * Determine the effective 'this' binding and coerce the current value * on the valstack to the effective one (in-place, at idx_this). * * The current this value in the valstack (at idx_this) represents either: * - the caller's requested 'this' binding; or * - a 'this' binding accumulated from the bound function chain * * The final 'this' binding for the target function may still be * different, and is determined as described in E5 Section 10.4.3. * * For global and eval code (E5 Sections 10.4.1 and 10.4.2), we assume * that the caller has provided the correct 'this' binding explicitly * when calling, i.e.: * * - global code: this=global object * - direct eval: this=copy from eval() caller's this binding * - other eval: this=global object * * Note: this function may cause a recursive function call with arbitrary * side effects, because ToObject() may be called.

50043. idx_this = idx_func + 1

50044. **comment:** XXX: for real world code, could just ignore array inheritance * and only look at array own properties.
label: code-design

50045. not needed, as we exit right away

50046. XXX: return type should probably be duk_size_t, or explicit checks are needed for * maximum size.

50047. relookup, may have changed

50048. leading digit + fractions

50049. Select appropriate escape format automatically, and set 'tmp' to a * value encoding both the escape format character and the nybble count: * * (nybble_count << 16) | (escape_char1) | (escape_char2)

50050. tc1 = true, tc2 = true

50051. See comments below on MakeTime why these are volatile.

50052. -> [val obj val]

50053. Store lexer position, restoring if quantifier is invalid.

50054. no need to compact since we already did that in duk_abandon_array_checked() * (regardless of whether an array part existed or not).

50055. marker; not actual tagged type

50056. **comment:** XXX: expensive check (also shared elsewhere - so add a shared internal API call?)
label: code-design

50057. For NaN/inf, the return value doesn't matter.

50058. curr char

50059. [regexp string]

50060. **comment:** NULL not needed here
label: code-design

50061. duk_pcall_prop() may itself throw an error, but we're content * in catching the obvious errors (like toLogString() throwing an * error).

50062. special result value handling

50063. [... Logger clog res]

50064. [... put_value varname]

50065. intermediate join to avoid valstack overflow

50066. ptr before mark-and-sweep

50067. end-of-input breaks

50068. reserve a jumps slot after instr before target spilling, used for NEXTENUM

50069. a label site has been emitted by duk_parse_stmt() automatically * (it will also emit the ENDLABEL).

50070. * Variable access

50071. **comment:** function: function must not be tail called
label: code-design

50072. third arg: absolute index (to entire valstack) of idx_bottom of new activation

50073. **comment:** * HASPROP variant used internally. * * This primitive must never throw an error, callers rely on this. * In particular, don't throw an error for prototype loops; instead, * pretend like the property doesn't exist if a prototype sanity limit * is reached. * * Does not implement proxy behavior: if applied to a proxy object, * returns key existence on the proxy object itself.
label: code-design

50074. 'out_buf' must be at least DUK_BI_DATE_ISO8601_BUFSIZE long.

50075. !DUK_USE_HSTRING_CLEN

50076. DUK_ERR_EVAL: no macros needed

50077. DUK_USE_PREFER_SIZE

50078. Inref copies, keep originals.

50079. object is a buffer object (duk_hbufferobject) (always exotic)

50080. recheck that the property still exists

50081. jump from true part to end

50082. **comment:** XXX: fastint fast path would be very useful here
label: code-design

50083. XXX: happens e.g. when evaluating: String(Buffer.prototype).

50084. 'name'

50085. eat identifier

50086. * Init stringtable: fixed variant

50087. Copy values through direct validated reads and writes.

50088. break matches always

50089. * "length" maps to number of formals (E5 Section 13.2) for function * declarations/expressions (non-bound functions). Note that 'nargs' * is NOT necessarily equal to the number of arguments.

50090. -1 for opcode

50091. * Regexp instance check, bytecode check, input coercion. ** See E5 Section 15.10.6.
50092. **comment:** hash part size or 0 if unused
label: code-design
50093. **comment:** XXX: this duplicates functionality in duk_regex.c where a similar loop is * required anyway. We could use that BUT we need to update the regexp compiler * 'nranges' too. Work this out a bit more cleanly to save space.
label: code-design
50094. helper to insert a (non-string) heap object into heap allocated list
50095. * Internal API calls which have (stack and other) semantics similar * to the public API.
50096. retval is directly usable
50097. **comment:** * Error throwing helpers ** The goal is to provide verbose and configurable error messages. Call * sites should be clean in source code and compile to a small footprint. * Small footprint is also useful for performance because small cold paths * reduce code cache pressure. Adding macros here only makes sense if there * are enough call sites to get concrete benefits.
label: code-design
50098. digit count
50099. * Hobject property set/get functionality. ** This is very central functionality for size, performance, and compliance. * It is also rather intricate; see hobject-algorithms.rst for discussion on * the algorithms and memory-management.rst for discussion on refcounts and * side effect issues. ** Notes: ** - It might be tempting to assert "refcount nonzero" for objects * being operated on, but that's not always correct: objects with * a zero refcount may be operated on by the refcount implementation * (finalization) for instance. Hence, no refcount assertions are made. ** - Many operations (memory allocation, identifier operations, etc) * may cause arbitrary side effects (e.g. through GC and finalization). * These side effects may invalidate duk_tval pointers which point to * areas subject to reallocation (like value stack). Heap objects * themselves have stable pointers. Holding heap object pointers or * duk_tval copies is not problematic with respect to side effects; * care must be taken when holding and using argument duk_tval pointers. ** - If a finalizer is executed, it may operate on the same object * we're currently dealing with. For instance, the finalizer might * delete a certain property which has already been looked up and * confirmed to exist. Ideally finalizers would be disabled if GC * happens during property access. At the moment property table realloc * disables finalizers, and all DECREFs may cause arbitrary changes so * handle DECREF carefully. ** - The order of operations for a DECREF matters. When DECREF is executed, * the entire object graph must be consistent; note that a refzero may * lead to a mark-and-sweep through a refcount finalizer.
50100. **comment:** * Compact the closure, in most cases no properties will be added later. * Also, without this the closures end up having unused property slots * (e.g. in Duktape 0.9.0, 8 slots would be allocated and only 7 used). * A better future solution would be to allocate the closure directly * to correct size (and setup the properties directly without going * through the API).
label: code-design
50101. 2/3 -> 1/8 = 12.5% min growth
50102. * Reference count updates ** Note: careful manipulation of refcounts. The top is * not updated yet, so all the activations are reachable * for mark-and-sweep (which may be triggered by decref). * However, the pointers are NULL so this is not an issue.
50103. 'hex'
50104. borrowed label name
50105. * String table algorithm: closed hashing with a probe sequence ** This is the default algorithm and works fine for environments with * minimal memory constraints.
50106. Longjmp state is kept clean in success path
50107. Longjmp callers are required to sync-and-null thr->ptr_curr_pc * before longjmp.
50108. Would indicate corrupted lists.
50109. Return invalid index; if caller uses this without checking * in another API call, the index won't map to a valid stack * entry.
50110. **comment:** XXX: faster internal way to get this
label: code-design
50111. old value is number: no refcount
50112. Using classification has smaller footprint than direct comparison.
50113. DUK_USE_ROM_GLOBAL_CLONE || DUK_USE_ROM_GLOBAL_INHERIT
50114. leading byte of match string
50115. **comment:** * Formal argument list ** We don't check for prohibited names or for duplicate argument * names here, because we don't yet know whether the function will * be strict. Function body parsing handles this retroactively.
label: code-design
50116. canonicalized by compiler
50117. [source template closure this]
50118. initial index
50119. 5 heap flags
50120. DUK_TOK_FUNCTION
50121. exponent for 'f'
50122. * Duktape includes (other than duk_features.h) ** The header files expect to be included in an order which satisfies header * dependencies correctly (the headers themselves don't include any other * includes). Forward declarations are used to break circular struct/typedef * dependencies.
50123. always 1 arg
50124. helpers
50125. * Single source autogenerated distributable for Duktape 1.8.0. ** Git commit 0a70d7e4c5227c84e3fed5209828973117d02849 (v1.8.0). * Git branch v1.8-maintenance. ** See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.
50126. ignore result
50127. DUK_JMPBUF_H_INCLUDED
50128. Newline allows module last line to contain a // comment.
50129. 'static'
50130. * sort() ** Currently qsort with random pivot. This is now really, really slow, * because there is no fast path for array parts. ** Signed indices are used because qsort() leaves and degenerate cases * may use a negative offset.
50131. extended types: compatible encoding
50132. **comment:** not allowed in strict mode, regardless of whether resolves; * in non-strict mode DELVAR handles both non-resolving and * resolving cases (the specification description is a bit confusing).
label: code-design
50133. valstack limit caller has check, prevents wrapping
50134. Note that byte order doesn't affect this test: all bytes in * 'test' will be 0xFF for two's complement.
50135. Note: number has no explicit tag (in 8-byte representation)
50136. * Calendar helpers * Some helpers are used for getters and can operate on normalized values * which can be represented with 32-bit signed integers. Other helpers are * needed by setters and operate on un-normalized double values, must watch * out for non-finite numbers etc.
50137. Unlike year, the other parts fit into 16 bits so %d format * is portable.
50138. reg/const for case value
50139. Equivalent year mapping, used to avoid DST trouble when platform * may fail to provide reasonable DST answers for dates outside the * ordinary range (e.g. 1970-2038). An equivalent year has the same * leap-year-ness as the original year and begins on the same weekday * (Jan 1). ** The year 2038 is avoided because there seem to be problems with it * on some platforms. The year 1970 is also avoided as there were * practical problems with it; an equivalent year is used for it too, * which breaks some DST computations for 1970 right now, see e.g. * test-bi-date-tzoffset-brute-fis.js.
50140. signed integer encoding needed to work with UTF-8
50141. Here val would be potentially invalid if we didn't make * a value copy at the caller.
50142. '&'
50143. jump is inserted here (variant 3)
50144. * Closing environment records. ** The environment record MUST be closed with the thread where its activation * is. In other words (if 'env' is open): ** - 'thr' must match _env.thread * - 'func' must match _env.callee * - 'regbase' must match _env.regbase * * These are not looked up from the env to minimize code size. *

* XXX: should access the own properties directly instead of using the API
50145. resume; [... initial_func undefined(= this) resume_value]
50146. ignore reclimit, not constructor
50147. '\xffCallee'
50148. is normalized
50149. Variant for writing duk_tvals so that any heap allocated values are * written out as tagged heap pointers.
50150. Coerce like boolean
50151. mark-and-sweep marking reached a recursion limit and must use multi-pass marking
50152. [(builtin objects)]
50153. **comment:** XXX: tv_func is not actually needed
 label: requirement
50154. * Source filename (or equivalent), for identifying thrown errors.
50155. '_proto_'
50156. **comment:** XXX: this is probably a useful shared helper: for a * duk_hbufferobject, get a validated buffer pointer/length.
 label: code-design
50157. parent env is the prototype
50158. get varenv for varname (callee's declarative lexical environment)
50159. Use a new environment and there's an 'arguments' object. * We need to initialize it right now.
50160. The timevalue must be in valid Ecmascript range, but since a local * time offset can be applied, we need to allow a +/- 24h leeway to * the value. In other words, although the UTC time is within the * Ecmascript range, the local part values can be just outside of it.
50161. Note: can be safely scanned as bytes (undecoded)
50162. ======
50163. No activation matches, use undefined for both .fileName and * .lineNumber (matches what we do with a _Tracedata based * no-match lookup).
50164. also goes into flags
50165. UnaryExpression
50166. **comment:** XXX: return something more useful, so that caller can throw?
 label: code-design
50167. Tailcalls are handled by back-patching the TAILCALL flag to the * already emitted instruction later (in return statement parser). * Since A and C have a special meaning here, they cannot be "shuffled".
50168. all virtual properties are non-configurable and non-writable
50169. [obj trap_result res_arr]
50170. **comment:** * The string conversion here incorporates all the necessary Ecmascript * semantics without attempting to be generic. nc_ctx->digits contains * nc_ctx->count digits (>= 1), with the topmost digit's 'position' * indicated by nc_ctx->k as follows: * * digits="123" count=3 k=0 --> 0.123 * digits="123" count=3 k=1 --> 1.23 * digits="123" count=3 k=5 --> 12300 * digits="123" count=3 k=-1 --> 0.0123 * * Note that the identifier names used for format selection are different * in Burger-Dybvig paper and Ecmascript specification (quite confusingly * so, because e.g. 'k' has a totally different meaning in each). See * documentation for discussion. * * Ecmascript doesn't specify any specific behavior for format selection * (e.g. when to use exponent notation) for non-base-10 numbers. * * The bigint space in the context is reused for string output, as there * is more than enough space for that (>1kB at the moment), and we avoid * allocating even more stack.
 label: code-design
50171. depth check is done when printing an actual type
50172. Preinc/predec for object properties.
50173. * duk_re_range_callback for generating character class ranges. * * When ignoreCase is false, the range is simply emitted as is. * We don't, for instance, eliminate duplicates or overlapping * ranges in a character class. * * When ignoreCase is true, the range needs to be normalized through * canonicalization. Unfortunately a canonicalized version of a * continuous range is not necessarily continuous (e.g. [x-{}] is * continuous but [X-{}] is not). The current algorithm creates the * canonicalized range(s) space efficiently at the cost of compile * time execution time (see doc/regexp.rst for discussion). * * Note that the ctx->nranges is a context-wide temporary value * (this is OK because there cannot be multiple character classes * being parsed simultaneously).
50174. init function state: init valstack allocations
50175. **comment:** Currently about 7*152 = 1064 bytes. The space for these * duk_bigsints is used also as a temporary buffer for generating * the final string. This is a bit awkward; a union would be * more correct.
 label: code-design
50176. Check statement type based on the first token type. * * Note: expression parsing helpers expect 'curr_tok' to * contain the first token of the expression upon entry.
50177. **comment:** * The 'in' operator requires an object as its right hand side, * throwing a TypeError unconditionally if this is not the case. * * However, lightfuncs need to behave like fully fledged objects * here to be maximally transparent, so we need to handle them * here.
 label: code-design
50178. * Number-to-string conversion. The semantics of these is very tightly * bound with the Ecmascript semantics required for call sites.
50179. **comment:** XXX: inject test
 label: code-design
50180. **comment:** SCANBUILD: with suitable dmin/dmax limits 'd' is unused
 label: code-design
50181. 0xff => 255 - 256 = -1; 0x80 => 128 - 256 = -128
50182. DUK_USE_PARANOI_ERROR
50183. Does not assume that jump_pc contains a DUK_OP_JUMP previously; this is intentional * to allow e.g. an INVALID opcode be overwritten with a JUMP (label management uses this).
50184. A 32-bit unsigned integer formats to at most 32 digits (the * worst case happens with radix == 2). Output the digits backwards, * and use a memmove() to get them in the right place.
50185. Without ROM objects "needs refcount update" == is heap allocated.
50186. **comment:** Note: 'act' is dangerous here because it may get invalidated at many * points, so we re-lookup it multiple times.
 label: code-design
50187. **comment:** XXX: Add a flag to reject an attempt to re-attach? Otherwise * the detached callback may immediately reattach.
 label: code-design
50188. side effects -> don't use tv_x, tv_y after
50189. [... enum]
50190. if gap is empty, behave as if not given at all
50191. When JX/JC not in use, the type mask above will avoid this case if needed.
50192. DUK_TOK_LT
50193. no value
50194. * HASVAR: check identifier binding from a given environment record * without traversing its parents. * * This primitive is not exposed to user code as such, but is used * internally for e.g. declaration binding instantiation. * * See E5 Sections: * 10.2.1.1 HasBinding(N) * 10.2.1.2 HasBinding(N) * * Note: strictness has no bearing on this check. Hence we don't take * a 'strict' parameter.
50195. hash probe sequence
50196. * Make the C call
50197. DUK_USE_NONSTD_FUNC_STMT
50198. has access to 'this' binding
50199. **comment:** Create a temporary enumerator to get the (non-duplicated) key list; * the enumerator state is initialized without being needed, but that * has little impact.
 label: code-design
50200. **comment:** XXX: optimize for string inputs: no need to coerce to a buffer * which makes a copy of the input.
 label: code-design

50201. initial size guess
50202. Buffer writes are often integers.
50203. env[funcname] = closure
50204. Safe to call multiple times.
50205. Default variants. Selection depends on speed/size preference. * Concretely: with gcc 4.8.1 -Os x64 the difference in final binary * is about +1kB for _FAST variants.
50206. formatted result limited
50207. [] -> [res]
50208. may be NaN
50209. 'undefined', artifact of lookup
50210. assume array part is comprehensive (contains all array indexed elements * or none of them); hence no need to check the entries part here.
50211. lJ value1: value
50212. pushes function template
50213. Identifier, i.e. don't allow reserved words
50214. **comment:** Parse an inner function, adding the function template to the current function's * function table. Return a function number to be used by the outer function. * * Avoiding O(depth^2) inner function parsing is handled here. On the first pass, * compile and register the function normally into the 'funcs' array, also recording * a lexer point (offset/line) to the closing brace of the function. On the second * pass, skip the function and return the same 'fnnum' as on the first pass by using * a running counter. * * An unfortunate side effect of this is that when parsing the inner function, almost * nothing is known of the outer function, i.e. the inner function's scope. We don't * need that information at the moment, but it would allow some optimizations if it * were used.
label: code-design
50215. [arg toLogString]
50216. out_token->lineterm set by caller
50217. note: any entries above the catchstack top are garbage and not zeroed
50218. Error code also packs a tracedata related flag.
50219. enumerator must have no keys deleted
50220. Stack size increases or stays the same.
50221. known to be number; in fact an integer
50222. avail_bytes += need_bytes
50223. **comment:** XXX: this illustrates that a C catchpoint implemented using duk_safe_call() * is a bit heavy at the moment. The wrapper compiles to ~180 bytes on x64. * Alternatives would be nice.
label: code-design
50224. temp variable to pass constants and flags to shared code
50225. char length of the atom parsed in this loop
50226. -> [... fn x y]
50227. **comment:** ToBoolean() does not require any operations with side effects so * we can do it efficiently. For footprint it would be better to use * duk_js_tobool()and then push+replace to the result slot.
label: code-design
50228. -> [...]
50229. topmost activation idx_retval is considered garbage, no need to init
50230. * Delayed activation environment record initialization (for functions * with NEWENV). * * The non-delayed initialization is handled by duk_handle_call().
50231. ref.value, ref.this.binding invalidated here by getprop call
50232. * Field accessor macros
50233. [... buf]
50234. idx_base and idx_base+1 get completion value and type
50235. integer number in range
50236. **comment:** XXX: duk_xdef_prop_index_wec() would be more appropriate * here but it currently makes some assumptions that might * not hold (e.g. that previous property is not an accessor). * * Using duk_put_prop() works incorrectly with '__proto__' * if the own property with that name has been deleted. This * does not happen normally, but a clever reviver can trigger * that, see complex reviver case in: test-bug-json-parse-__proto__.js.
label: code-design
50237. Force interrupt right away if we're paused or in "checked mode". * Step out is handled by callstack unwind.
50238. check stack before interning (avoid hanging temp)
50239. The extra (+4) is tight.
50240. Send version identification and flush right afterwards. Note that * we must write raw, unframed bytes here.
50241. DUK_TOK_STATIC
50242. No pointer compression because pointer is potentially outside of * Duktape heap.
50243. 'null'
50244. pc2line
50245. always normalized
50246. JUMP L1 omitted
50247. **comment:** writable, not configurable
label: code-design
50248. * Init/assert flags, copying them where appropriate. Some flags * (like NEWENV) are processed separately below.
50249. big endian
50250. entry_top + 3
50251. must match bytecode defines now; build autogenerate?
50252. Although NAMEBINDING is not directly needed for using * function instances, it's needed by bytecode dump/load * so copy it too.
50253. ''
50254. * Automatically generated by genbuiltins.py, do not edit!
50255. * Debug connection read primitives
50256. Note: when parsing a formal list in non-strict context, e.g. * "implements" is parsed as an identifier. When the function is * later detected to be strict, the argument list must be rechecked * against a larger set of reserved words (that of strict mode). * This is handled by duk_parse_func_body(). Here we recognize * whatever tokens are considered reserved in current strictness * (which is not always enough).
50257. **comment:** XXX: optimize with more direct internal access
label: code-design
50258. __proto__ setter returns 'undefined' on success unlike the * setPrototypeOf() call which returns the target object.
50259. 0x00-0x0f
50260. * String comparison (E5 Section 11.8.5, step 4), which * needs to compare codepoint by codepoint. * * However, UTF-8 allows us to use strcmp directly: the shared * prefix will be encoded identically (UTF-8 has unique encoding) * and the first differing character can be compared with a simple * unsigned byte comparison (which strcmp does). * * This will not work properly for non-xutf-8 strings, but this * is not an issue for compliance.
50261. DUK_TOK_BOR_EQ
50262. Argument must be a string, e.g. a buffer is not allowed.
50263. Note: val is a stable duk_tval pointer. The caller makes * a value copy into its stack frame, so 'tv_val' is not subject * to side effects here.
50264. number of formal arguments
50265. register, constant, or value
50266. Note: intentionally allow leading zeroes here, as the * actual parser will check for them.
50267. Impose a maximum buffer length for now. Restricted artificially to * ensure resize computations or adding a heap header length won't * overflow size_t and that a signed duk_int_t can hold a buffer * length. The limit should be synchronized with DUK_HSTRING_MAX_BYTELEN.
50268. h_code: held in bw_code

50269. * Forward declarations.
50270. heap pointer comparison suffices
50271. Note: 'count' is currently not adjusted by rounding (i.e. the * digits are not "chopped off". That shouldn't matter because * the digit position (absolute or relative) is passed on to the * convert-and-push function.
50272. **comment:** The variable environment for magic variable bindings needs to be * given by the caller and recorded in the arguments object. * * See E5 Section 10.6, the creation of setters/getters. * * The variable environment also provides access to the callee, so * an explicit (internal) callee property is not needed.
label: code-design
50273. All components default to 0 except day-of-month which defaults * to 1. However, because our internal day-of-month is zero-based, * it also defaults to zero here.
50274. Backtrack.
50275. **comment:** XXX: code duplication
label: code-design
50276. arg count
50277. **comment:** Switch to caller's setjmp() catcher so that if an error occurs * during error handling, it is always propagated outwards instead * of causing an infinite loop in our own handler.
label: code-design
50278. [... number] -> [... string]
50279. 'data' (and everything in it) is reachable through h_res now
50280. Fast path.
50281. called as a normal function: return new Date().toString()
50282. -----XX
50283. must be computed after realloc
50284. 'join'
50285. Get the value represented by an duk_ispec to a register or constant. * The caller can control the result by indicating whether or not: * * (1) a constant is allowed (sometimes the caller needs the result to * be in a register) * * (2) a temporary register is required (usually when caller requires * the register to be safely mutable; normally either a bound * register or a temporary register are both OK) * * (3) a forced register target needs to be used * * Bytecode may be emitted to generate the necessary value. The return * value is either a register or a constant.
50286. **comment:** XXX: FP_ZERO check can be removed, the else clause handles it * correctly (preserving sign).
label: code-design
50287. **comment:** XXX: the string shouldn't appear twice, but we now loop to the * end anyway; if fixed, add a looping assertion to ensure there * is no duplicate.
label: code-design
50288. re-lookup first char on first loop
50289. [... buf loop (proplist)]
50290. DUK_BUFOBJ_INT16ARRAY
50291. No net refcount changes.
50292. DUK_USE_DATE_FMT_STRFTIME
50293. eat the right paren
50294. **comment:** A straightforward 64-byte lookup would be faster * and cleaner, but this is shorter.
label: code-design
50295. For this to work, DATEMSK must be set, so this is not very * convenient for an embeddable interpreter.
50296. Fixed buffer; data follows struct, with proper alignment guaranteed by * struct size.
50297. **comment:** * Call handling. * * Main functions are: * * - duk_handle_call_unprotected(): unprotected call to Ecmascript or * Duktape/C function * - duk_handle_call_protected(): protected call to Ecmascript or * Duktape/C function * - duk_handle_safe_call(): make a protected C call within current * activation * - duk_handle_ecma_call_setup(): Ecmascript-to-Ecmascript calls * (not always possible), including tail calls and coroutine resume * * See 'execution.rst'. * * Note: setjmp() and local variables have a nasty interaction, * see execution.rst; non-volatile locals modified after setjmp() * call are not guaranteed to keep their value.
label: code-design
50298. -1 if disjunction is complex, char length if simple
50299. noblame_fileline
50300. Backtrack to previous slash or start of buffer.
50301. func limits
50302. This is a rare property helper; it sets the global thrower (E5 Section 13.2.3) * setter/getter into an object property. This is needed by the 'arguments' * object creation code, function instance creation code, and Function.prototype.bind().
50303. **comment:** Workaround for some exotic platforms where NAN is missing * and the expression (0.0 / 0.0) does NOT result in a NaN. * Such platforms use the global 'duk_computed_nan' which must * be initialized at runtime. Use 'volatile' to ensure that * the compiler will actually do the computation and not try * to do constant folding which might result in the original * problem.
label: code-design
50304. A -> target register (A, A+1) for call setup * (for DUK_OP_CSREGI, 'a' is indirect) * B -> register containing target function (not type checked here)
50305. Return the Buffer to allow chaining: b.fill(0x11).fill(0x22, 3, 5).toString()
50306. **comment:** code_idx: not needed
label: requirement
50307. keep func->h_funcs; inner functions are not reparsed to avoid O(depth^2) parsing
50308. load factor min 25%
50309. 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx
50310. Note: peek cannot currently trigger a detach * so the dbg_detaching == 0 assert outside the * loop is correct.
50311. Assume interrupt init/counter are properly initialized here.
50312. * Parse a case or default clause.
50313. **comment:** XXX: when this error is caused by a nonexistent * file given to duk_peval_file() or similar, the * error message is not the best possible.
label: code-design
50314. impose a reasonable exponent limit, so that exp * doesn't need to get tracked using a bigint.
50315. marker value; in E5 2^32-1 is not a valid array index (2^32-2 is highest valid)
50316. maxval
50317. valstack space allocated especially for proxy lookup which does a * recursive property lookup
50318. Accept ASCII strings which conform to identifier requirements * as being emitted without key quotes. Since we only accept ASCII * there's no need for actual decoding: 'p' is intentionally signed * so that bytes >= 0x80 extend to negative values and are rejected * as invalid identifier codepoints.
50319. Careful overflow handling. When multiplying by 10: * - 0x19999998 x 10 = 0xfffffff0: no overflow, and adding * 0..9 is safe. * - 0x19999999 x 10 = 0xffffffff: no overflow, adding * 0..5 is safe, 6...9 overflows. * - 0x1999999a x 10 = 0x10000004: always overflow.
50320. The Str(key, holder) operation. * * Stack policy: [... key] -> [...]
50321. * Bitstream encoder
50322. -> [sep ToObject(this) len str sep]
50323. Decrement PC if that was requested, this requires a PC sync.
50324. Array properties have exotic behavior but they are concrete, * so no special handling here. * * Arguments exotic behavior (E5 Section 10.6, [[GetOwnProperty]]) * is only relevant as a post-check implemented below; hence no * check here.
50325. XXX: request a "last statement is terminal" from duk_parse_stmt() and duk_parse_stmts(); * we could avoid the last RETURN if we could ensure there is no way to get here * (directly or via a jump)
50326. [... key_obj key key flags]
50327. no side effects
50328. INACTIVE: no activation, single function value on valstack
50329. Equality may be OK but >length not. Checking * this explicitly avoids some overflow cases * below.

50330. this cannot initiate a detach
50331. done so that duk_mark_heaphdr() works correctly
50332. if highest bit of a register number is set, it refers to a constant instead
50333. **comment:** Slow variants, call to a helper to reduce code size. * Can be used explicitly when size is always more important than speed.
label: code-design
50334. DUK_USE_REFERENCE_COUNTING
50335. Caller ensures space for at least DUK_JSON_MAX_ESC_LEN.
50336. byte offset to buf
50337. * The catch variable must be updated to reflect the new allocated * register for the duration of the catch clause. We need to store * and restore the original value for the varmap entry (if any).
50338. **comment:** unused now
label: code-design
50339. **comment:** stack[0] = callback * stack[1] = thisArg * stack[2] = object * stack[3] = ToUInt32(length) (unused, but avoid unnecessary pop) * stack[4] = result array (or undefined)
label: code-design
50340. still in prologue
50341. heap->dbg_processing: keep on purpose to avoid debugger re-entry in detaching state
50342. fixed-format termination conditions
50343. DUK_TOK_ELSE
50344. * Add line number to a compiler error.
50345. array of function templates: [func1, offset1, line1, func2, offset2, line2] * offset/line points to closing brace to allow skipping on pass 2
50346. Ensure copy is covered by underlying buffers.
50347. longjmp state
50348. * Note: * * - Intentionally attempt (empty) match at char_offset == k_input->clen * * - Negative char_offsets have been eliminated and char_offset is duk_uint32_t * -> no need or use for a negative check
50349. Expression, terminates at a ')'
50350. can't happen when keeping current stack size
50351. Special handling for CALL/NEW/MPUTOBJ/MPUTARR shuffling. * For each, slot B identifies the first register of a range * of registers, so normal shuffling won't work. Instead, * an indirect version of the opcode is used.
50352. DUK_TOK_FOR
50353. Miscellaneous.
50354. -> [... retval]
50355. If a setter/getter is missing (undefined), the descriptor must * still have the property present with the value 'undefined'.
50356. -> [timeval this]
50357. **comment:** Clean up stack
label: code-design
50358. **comment:** dump all allocated entries, unused entries print as 'unused', * note that these may extend beyond current 'length' and look * a bit funny.
label: code-design
50359. unwind to 'yield' caller
50360. How much stack to require on entry to object/array decode
50361. (e.g. k=3, digits=2 -> "12X")
50362. Note: changing from writable to non-writable is OK
50363. unwinds valstack, updating refcounts
50364. For JX, expressing the whole unsigned 32-bit range matters.
50365. r <- 2 * f
50366. currently active breakpoints: NULL term, borrowed pointers
50367. conversions, coercions, comparison, etc
50368. [key val] -> [key]
50369. use stack allocated buffer to ensure reachability in errors (e.g. intern error)
50370. **comment:** * Manually optimized double-to-fastint downgrade check. * * This check has a large impact on performance, especially for fastint * slow paths, so must be changed carefully. The code should probably be * optimized for the case where the result does not fit into a fastint, * to minimize the penalty for "slow path code" dealing with fractions etc. * * At least on one tested soft float ARM platform double-to-int64 coercion * is very slow (and sometimes produces incorrect results, see self tests). * This algorithm combines a fastint compatibility check and extracting the * integer value from an IEEE double for setting the tagged fastint. For * other platforms a more naive approach might be better. * * See doc/fastint.rst for details.
label: code-design
50371. * Misc support functions
50372. ch is a literal character here or -1 if parsed entity was * an escape such as "\s".
50373. Code emission flags, passed in the 'opcode' field. Opcode + flags * fit into 16 bits for now, so use duk_small_uint.t.
50374. array of formal argument names (-> _Formals)
50375. [... func this arg1 ... argN]
50376. * Simple atom * * If atom_char_length is zero, we'll have unbounded execution time for e.g. * /0*x/.exec('x'). We can't just skip the match because it might have some * side effects (for instance, if we allowed captures in simple atoms, the * capture needs to happen). The simple solution below is to force the * quantifier to match at most once, since the additional matches have no effect. * * With a simple atom there can be no capture groups, so no captures need * to be reset.
50377. key encountered as a getter
50378. **comment:** XXX: typing
label: code-design
50379. catches EOF (NUL)
50380. 'act' is no longer accessed, scanbuild fix
50381. **comment:** XXX: Array size is known before and (2 * re_ctx.nsaved) but not taken * advantage of now. The array is not compacted either, as regexp match * objects are usually short lived.
label: code-design
50382. Ecmascript activation + Duktape.Thread.yield() activation
50383. **comment:** Presence of 'fun' is config based, there's a marginal performance * difference and the best option is architecture dependent.
label: code-design
50384. [... RegExp]
50385. DUK_TOK_DECREMENT
50386. [... errhandler undefined errval]
50387. **comment:** The target for this LDCONST may need output shuffling, but we assume * that 'pc_ldconst' will be the LDCONST that we can patch later. This * should be the case because there's no input shuffling. (If there's * no catch clause, this LDCONST will be replaced with a NOP.)
label: code-design
50388. no wrap assuming h_bufobj->length is valid
50389. Nominal size check.
50390. register bound variables are non-configurable -> always false
50391. -> [obj trap handler target]
50392. reasonable output estimate
50393. Note: DST adjustment is determined using UTC time. * If 'd' is NaN, tzoffset will be 0.
50394. * Character and charcode access
50395. nargs == 2 so we can pass a callstack level to eval().

50396. First pass parse is very lenient (e.g. allows '1.2.3') and extracts a * string for strict number parsing.
50397. normal valued properties
50398. '\xffLexenv'
50399. Module loading failed. Node.js will forget the module * registration so that another require() will try to load * the module again. Mimic that behavior.
50400. **comment:** Note: for an accessor without getter, falling through to * check for "caller" exotic behavior is unnecessary as * "undefined" will never activate the behavior. But it does * no harm, so we'll do it anyway.
 label: code-design
50401. e.g. 12.3 -> digits="123" k=2 -> 1.23e1
50402. **comment:** XXX: there is room to use a shared helper here, many built-ins * check the 'this' type, and if it's an object, check its class, * then get its internal value, etc.
 label: code-design
50403. variable map for pass 2 (identifier -> register number or null (unmapped))
50404. buffer pointer is to an externally allocated buffer
50405. 'Uint32Array'
50406. 'with'
50407. **comment:** * (Re)initialize the temporary byte buffer. May be called extra times * with little impact.
 label: code-design
50408. callstack index of related activation
50409. non-standard
50410. may be safe, or non-safe depending on flags
50411. activation prevents yield (native call or "new")
50412. **comment:** XXX: expensive, but numconv now expects to see a string
 label: code-design
50413. DUK_FLD_VARINT; not relevant here
50414. **comment:** * Note: use indirect realloc variant just in case mark-and-sweep * (finalizers) might resize this same buffer during garbage * collection.
 label: code-design
50415. eat 'try'
50416. 1110 xxxx 10xx xxxx 10xx xxxx
50417. Note: it's nice if size is 2^N (at least for 32-bit platforms).
50418. Quite heavy assert: check valstack policy. Improper * shuffle instructions can write beyond valstack_top/end * so this check catches them in the act.
50419. Expects 'this' at top of stack on entry.
50420. The token in the switch has already been eaten here
50421. currently all labels accept a break, so no explicit check for it now
50422. [arg result]
50423. got lineterm preceding non-whitespace, non-lineterm token
50424. intentional overlap with earlier memzero
50425. Allow exponent
50426. '\xffSource'
50427. read-only object, in code section
50428. **comment:** Slice and accessor information. * * Because the underlying buffer may be dynamic, these may be * invalidated by the buffer being modified so that both offset * and length should be validated before every access. Behavior * when the underlying buffer has changed doesn't need to be clean: * virtual 'length' doesn't need to be affected, reads can return * zero/NaN, and writes can be ignored. * * Note that a data pointer cannot be precomputed because 'buf' may * be dynamic and its pointer unstable.
 label: code-design
50429. Shared object part.
50430. **comment:** XXX: there are language sensitive rules for the ASCII range. * If/when language/locale support is implemented, they need to * be implemented here for the fast path. There are no context * sensitive rules for ASCII range.
 label: code-design
50431. DUK_OP_RETURN flags in A
50432. DUK_TOK_ADD
50433. allocate to reg only (not const)
50434. **comment:** not strictly necessary, but avoids "uninitialized variable" warnings
 label: code-design
50435. * Augment an error at throw time; allow a user error handler (if defined) * to process/replace the error. The error to be augmented is at the * stack top.
50436. **comment:** This macro works when a regconst field is 9 bits, [0,0x1ff]. Adding * DUK_LIKELY/DUK_UNLIKELY increases code footprint and doesn't seem to * improve performance on x64 (and actually harms performance in some tests).
 label: code-design
50437. requested number of output digits; 0 = free-format
50438. * Shared readfield and writefield methods * * The readfield/writefield methods need support for endianness and field * types. All offsets are byte based so no offset shifting is needed.
50439. Gap in array; check for inherited property, * bail out if one exists. This should be enough * to support gappy arrays for all practical code.
50440. **comment:** XXX: could improve bufwriter handling to write multiple codepoints * with one ensure call but the relative benefit would be quite small.
 label: code-design
50441. **comment:** * Delete references to given hstring from the heap string cache. * * String cache references are 'weak': they are not counted towards * reference counts, nor serve as roots for mark-and-sweep. When an * object is about to be freed, such references need to be removed.
 label: code-design
50442. DUK_TOK_EOF
50443. **comment:** XXX: signalling the need to shrink check (only if unwound)
 label: code-design
50444. XXX: duk_dup_unvalidated(ctx, -2) etc.
50445. fill new entries with -unused- (required, gc reachable)
50446. -> [O toString O]
50447. accept anything, expect first value (EOF will be * caught by key parsing below.)
50448. unknown, ignore
50449. * Note: fmax() does not match the E5 semantics. E5 requires * that if -any- input to Math.max() is a NaN, the result is a * NaN. fmax() will return a NaN only if - both- inputs are NaN. * Same applies to fmin(). * * Note: every input value must be coerced with ToNumber(), even * if we know the result will be a NaN anyway: ToNumber() may have * side effects for which even order of evaluation matters.
50450. [... v1 v2 name filename]
50451. !DUK_USE_PANIC_HANDLER
50452. **comment:** XXX: inlined DECREF macro would be nice here: no NULL check, * refzero queueing but no refzero algorithm run (= no pointer * instability), inline code.
 label: code-design
50453. expected ival
50454. **comment:** A 16-bit listlen makes sense with 16-bit heap pointers: there * won't be space for 64k strings anyway.
 label: code-design
50455. * Finalize refcounts for heap elements just about to be freed. * This must be done for all objects before freeing to avoid any * stale pointer dereferences. * * Note that this must deduce the set of objects to be freed * identically to duk_sweep_heap().
50456. no sce, or sce scan not best

50457. caller must have decref'd values above new_a_size (if that is necessary)
50458. [... buf loop (proplist) (gap)]
50459. **comment:** XXX: duk_hobject_hasprop() would be correct for * non-Proxy objects too, but it is about ~20-25% * slower at present so separate code paths for * Proxy and non-Proxy now.
label: code-design
50460. Coercion may be needed, the helper handles that by pushing the * tagged values to the stack.
50461. checked by caller
50462. 'Int16Array'
50463. res->strs16[] is zeroed and zero decodes to NULL, so no NULL inits.
50464. entry part must not contain any configurable properties, or * writable properties (if is_frozen).
50465. thread; minimizes argument passing
50466. step 4.d
50467. **comment:** * The Number constructor uses ToNumber(arg) for number coercion * (coercing an undefined argument to NaN). However, if the * argument is not given at all, +0 must be used instead. To do * this, a vararg function is used.
label: code-design
50468. one operator (= assign)
50469. no need to init reg, it will be undefined on entry
50470. string objects must not created without internal value
50471. eat backslash on entry
50472. * Cached module check. * * If module has been loaded or its loading has already begun without * finishing, return the same cached value ('exports'). The value is * registered when module load starts so that circular references can * be supported to some extent.
50473. * Setup a preliminary activation and figure out nargs/nregs. * * Don't touch valstack_bottom or valstack_top yet so that Duktape API * calls work normally.
50474. **comment:** XXX: Rework to use an always-inline function?
label: code-design
50475. **comment:** * Common heap header * * All heap objects share the same flags and refcount fields. Objects other * than strings also need to have a single or double linked list pointers * for insertion into the "heap allocated" list. Strings are held in the * heap-wide string table so they don't need link pointers. * * Technically, 'h_refcount' must be wide enough to guarantee that it cannot * wrap (otherwise objects might be freed incorrectly after wrapping). This * means essentially that the refcount field must be as wide as data pointers. * On 64-bit platforms this means that the refcount needs to be 64 bits even * if an 'int' is 32 bits. This is a bit unfortunate, and compromising on * this might be reasonable in the future. * * Heap header size on 32-bit platforms: 8 bytes without reference counting, * 16 bytes with reference counting.
label: code-design
50476. We're a varargs function because we need to detect whether * initialValue was given or not.
50477. cannot be strict (never mapped variables)
50478. Avoid recursive re-enter; enter when we're attached or * detaching (to finish off the pending detach).
50479. DUK_DEBUGGER_H_INCLUDED
50480. Maximum length of CommonJS module identifier to resolve. Length includes * both current module ID, requested (possibly relative) module ID, and a * slash in between.
50481. **comment:** * Property count check. This is the only point where we ensure that * we don't get more (allocated) property space that we can handle. * There aren't hard limits as such, but some algorithms fail (e.g. * finding next higher prime, selecting hash part size) if we get too * close to the 4G property limit. * * Since this works based on allocation size (not actually used size), * the limit is a bit approximate but good enough in practice.
label: code-design
50482. This loop intentionally does not ensure characters are valid * ([0-9a-fA-F]) because the hex decode call below will do that.
50483. 0 if no used entries
50484. * Basic double / byte union memory layout.
50485. copy values (no overlap even if to_ctx == from_ctx; that's not * allowed now anyway)
50486. Use match charlen instead of bytelen, just in case the input and * match codepoint encodings would have different lengths.
50487. **comment:** Count actually used array part entries and array minimum size. * NOTE: 'out_min_size' can be computed much faster by starting from the * end and breaking out early when finding first used entry, but this is * not needed now.
label: code-design
50488. if new_p == NULL, all of these pointers are NULL
50489. -> [... obj retval/error]
50490. [... holder key] -> [... holder]
50491. 'this' binding is just before current activation's bottom
50492. **comment:** * Error handling macros, assertion macro, error codes. * * There are three level of 'errors': * * 1. Ordinary errors, relative to a thread, cause a longjmp, catchable. * 2. Fatal errors, relative to a heap, cause fatal handler to be called. * 3. Panic errors, unrelated to a heap and cause a process exit. * * Panics are used by the default fatal error handler and by debug code * such as assertions. By providing a proper fatal error handler, user * code can avoid panics in non-debug builds.
label: code-design
50493. Arguments can be: [source? filename? &comp_args] so that * nargs is 1 to 3. Call site encodes the correct nargs count * directly into flags.
50494. a
50495. Allow leading zeroes (e.g. "0123" -> "123")
50496. * Default clause.
50497. d <- (quotient (* r B) s) (in range 0...B-1)
50498. **comment:** XXX: There used to be a shared log buffer here, but it was removed * when dynamic buffer spare was removed. The problem with using * bufwriter is that, without the spare, the buffer gets passed on * as an argument to the raw() call so it'd need to be resized * (reallocated) anyway. If raw() call convention is changed, this * could be made more efficient.
label: code-design
50499. Commands initiated by debug client.
50500. next loop requires a comma
50501. 0x40...0x4f
50502. 'string'
50503. match == search string, by definition
50504. **comment:** Shared handling for encode/decode argument. Fast path handling for * buffer and string values because they're the most common. In particular, * avoid creating a temporary string or buffer when possible.
label: code-design
50505. 32-bit x 11-bit = 43-bit, fits accurately into a double
50506. Opcode only emitted by compiler when debugger * support is enabled. Ignore it silently without * debugger support, in case it has been loaded * from precompiled bytecode.
50507. eat closing brace
50508. Behavior for constructor and non-constructor call is * the same except for augmenting the created error. When * called as a constructor, the caller (duk_new()) will handle * augmentation; when called as normal function, we need to do * it here.
50509. Encode into a single opcode (18 bits can encode 1-2 bytes + length indicator)
50510. defined(DUK_USE_DATE_NOW_WINDOWS) || defined(DUK_USE_DATE_TZO_WINDOWS)
50511. without finally, the second jump slot is used to jump to end of stmt
50512. slot B is a target (default: source)
50513. fastint constants etc
50514. * Unicode codepoints above U+FFFF are encoded as surrogate * pairs here. This ensures that all CESU-8 codepoints are * 16-bit values as expected in ECMAScript. The surrogate * pairs always get a 3-byte encoding (each) in CESU-8. * See: http://en.wikipedia.org/wiki/Surrogate_pair * * 20-bit codepoint, 10 bits (A and B) per surrogate pair: * * x = 0b00000000 0000AAAA AAAAABBBBBBBB * sp1 = 0b110110AA AAAAAAAA (0xd800 + ((x >> 10) & 0x3ff)) * sp2 =

0b110111BB BBBB BBBB (0xdc00 + (x & 0x3ff)) * * Encoded into CESU-8: * * sp1 -> 0b11101101 (0xe0 + ((sp1 >> 12) & 0x0f)) * -> 0b1010AAAA (0x80 + ((sp1 >> 6) & 0x3f)) * -> 0b10AAAAAA (0x80 + (sp1 & 0x3f)) * sp2 -> 0b11101101 (0xe0 + ((sp2 >> 12) & 0x0f)) * -> 0b1011BBBB (0x80 + ((sp2 >> 6) & 0x3f)) * -> 0b10BBBBBB (0x80 + (sp2 & 0x3f)) * * Note that 0x10000 must be subtracted first. The code below * avoids the sp1, sp2 temporaries which saves around 20 bytes * of code.

50515. Duktape.modLoaded

50516. We could do this operation without caller updating bw_ctx->ptr, * but by writing it back here we can share code better.

50517. Note: if encoding ends by hitting end of input, we don't check that * the encoding is valid, we just assume it is.

50518. **comment:** XXX: call method tail call?

label: code-design

50519. w.../

50520. always provideThis=true

50521. Note: only numbered indices are relevant, so arr_idx fast reject * is good (this is valid unless there are more than 4**32-1 arguments).

50522. * Thread state and calling context checks

50523. token type (with reserved word identification)

50524. **comment:** XXX: there's no explicit recursion bound here now. For the average * qsort recursion depth O(log n) that's not really necessary: e.g. for * 2**32 recursion depth would be about 32 which is OK. However, qsort * worst case recursion depth is O(n) which may be a problem.

label: code-design

50525. verified at beginning

50526. **comment:** Fall back to the initial (original) Object.toString(). We don't * currently have pointers to the built-in functions, only the top * level global objects (like "Array") so this is now done in a bit * of a hacky manner. It would be cleaner to push the (original) * function and use duk_call_method().

label: code-design

50527. XXX: duk_push_uint_string()

50528. Built-in providers

50529. **comment:** XXX: could add flags for "is valid CESU-8" (EcmaScript compatible strings), * "is valid UTF-8", "is valid extended UTF-8" (internal strings are not, * regexp bytecode is), and "contains non-BMP characters". These are not * needed right now.

label: code-design

50530. pop final_cons

50531. **comment:** When ptr == NULL, the format argument is unused.

label: code-design

50532. To convert a heap stridx to a token number, subtract * DUK_STRIDX_START_RESERVED and add DUK_TOK_START_RESERVED.

50533. ANSI C typing

50534. These pointers are at the start of the struct so that they pack * nicely. Mixing pointers and integer values is bad on some * platforms (e.g. if int is 32 bits and pointers are 64 bits).

50535. 1=little endian

50536. * "IdentifierStart" production check.

50537. * Compiler state

50538. **comment:** XXX: this is awkward as we use an internal method which doesn't handle * extensibility etc correctly. Basically we'd want to do a [[DefineOwnProperty]] * or Object.defineProperty() here.

label: code-design

50539. **comment:** * Entries part is a bit more complex

label: code-design

50540. * On first pass, perform actual parsing. Remember valstack top on entry * to restore it later, and switch to using a new function in comp_ctx.

50541. nop: insert top to top

50542. h_match is borrowed, remains reachable through match_obj

50543. 0 = uppercase, 32 = lowercase (= 'a' - 'A')

50544. catch jump

50545. 3: toLocaleString

50546. not reachable

50547. leave 'cat' as top catcher (also works if catchstack exhausted)

50548. +1 for opcode

50549. 'varname' is in stack in this else branch, leaving an unbalanced stack below, * but this doesn't matter now.

50550. Not halfway, round to nearest.

50551. remove in_val

50552. idx_argbase

50553. Strict by default.

50554. fixed precision and zero padding would be required

50555. Ensuring (reserving) space.

50556. flags: "m"

50557. 'setPrototypeOf'

50558. return 1 to allow callers to tail call

50559. * Not found (even in global object)

50560. * Relocate memory with garbage collection, using a callback to provide * the current allocated pointer. This variant is used when a mark-and-sweep * (e.g. finalizers) might change the original pointer.

50561. 'number'

50562. label id allocation (running counter)

50563. **comment:** * Shared error messages: declarations and macros * * Error messages are accessed through macros with fine-grained, explicit * error message distinctions. Concrete error messages are selected by the * macros and multiple macros can map to the same concrete string to save * on code footprint. This allows flexible footprint/verbosity tuning with * minimal code impact. There are a few limitations to this approach: * (1) switching between plain messages and format strings doesn't work * conveniently, and (2) conditional strings are a bit awkward to handle. * * Because format strings behave differently in the call site (they need to * be followed by format arguments), they have a special prefix (DUK_STR_FMT_ * and duk_str_fmt_). * * On some compilers using explicit shared strings is preferable; on others * it may be better to use straight literals because the compiler will combine * them anyway, and such strings won't end up unnecessarily in a symbol table.

label: code-design

50564. **comment:** XXX: how to figure correct size?

label: code-design

50565. **comment:** * See the following documentation for discussion: * * doc/execution.rst: control flow details * * Try, catch, and finally "parts" are Blocks, not Statements, so * they must always be delimited by curly braces. This is unlike e.g. * the if statement, which accepts any Statement. This eliminates any * questions of matching parts of nested try statements. The Block * parsing is implemented inline here (instead of calling out). * * Finally part has a 'let scoped' variable, which requires a few kinks * here.

label: code-design

50566. op_flags

50567. # argument registers target function wants (< 0 => never for ecma calls)

50568. Different calling convention than above used because the helper * is shared.

50569. equality not actually possible

50570. **comment:** XXX: make this leniency dependent on flags or strictness?

label: code-design

50571. [... func this arg1 ... argN envobj]

50572. 32: setDate

50573. slot C is a target (default: source)
50574. -> [... lval rval new_rval]
50575. Sealed and frozen objects cannot gain any more properties, * so this is a good time to compact them.
50576. Parse a single variable declaration (e.g. "i" or "i=10"). A leading 'var' * has already been eaten. These is no return value in 'res', it is used only * as a temporary. * * When called from 'for-in' statement parser, the initializer expression must * not allow the 'in' token. The caller supply additional expression parsing * flags (like DUK__EXPR_FLAG_REJECT_IN) in 'expr_flags'. * * Finally, out_rc_varname and out_reg_varbind are updated to reflect where * the identifier is bound: * * If register bound: out_reg_varbind >= 0, out_rc_varname == 0 (ignore) * If not register bound: out_reg_varbind < 0, out_rc_varname >= 0 * * These allow the caller to use the variable for further assignment, e.g. * as is done in 'for-in' parsing.
50577. sanity
50578. deal with negative values
50579. Unlike for negative arguments, some call sites * want length to be clamped if it's positive.
50580. create bound function object
50581. maximum bytecode copies for {n,m} quantifiers
50582. [... lval rval]
50583. **comment:** Note: Boolean prototype's internal value property is not writable, * but duk_xdef_prop_stridx() disregards the write protection. Boolean * instances are immutable. * * String and buffer special behaviors are already enabled which is not * ideal, but a write to the internal value is not affected by them.
label: code-design
50584. use duk_double_union as duk_tval directly
50585. [... obj finalizer obj heapDestruct] -> [... obj retval]
50586. **comment:** * Assignments are right associative, allows e.g. * a = 5; * a += b = 9; // same as a += (b = 9) * -> expression value 14, a = 14, b = 9 * * Right associativity is reflected in the BP for recursion, * "-1" ensures assignment operations are allowed. * * XXX: just use DUK__BP_COMMA (i.e. no need for 2-step bp levels)?
label: code-design
50587. don't increase 'count'
50588. module.exports = exports
50589. simulate alloc failure on every alloc (except when mark-and-sweep is running)
50590. must relookup act in case of side effects
50591. **comment:** XXX: take advantage of val being unsigned, no need to mask
label: code-design
50592. status booleans
50593. identifier handling
50594. duk_unicode_caseconv_lc()
50595. DUK_TOK_PROTECTED
50596. target
50597. reserved words: future reserved words
50598. Ensure space for maximum multi-character result; estimate is overkill.
50599. * Getters. * * Implementing getters is quite easy. The internal time value is either * NaN, or represents milliseconds (without fractions) from Jan 1, 1970. * The internal time value can be converted to integer parts, and each * part will be normalized and will fit into a 32-bit signed integer. * * A shared native helper to provide all getters. Magic value contains * a set of flags and also packs the date component index argument. The * helper provides: * * getFullYear() * getUTCFullYear() * getMonth() * getUTCMonth() * getDate() * getUTCDate() * getDay() * getUTCDay() * getHours() * getUTCHours() * getMinutes() * getUTCMinutes() * getSeconds() * getUTCSeconds() * getMilliseconds() * getUTCMilliseconds() * getYear() * * Notes: * * - Date.prototype.getDate(): 'date' means day-of-month, and is * zero-based in internal calculations but public API expects it to * be one-based. * * - Date.prototype.getTime() and Date.prototype.valueOf() have identical * behavior. They have separate function objects, but share the same C * function (duk_bi_date_prototype_value_of).
50600. * Pre resize assertions.
50601. In-place unary operation.
50602. Note: expect that caller has already eaten the left paren
50603. implicit leading one
50604. **comment:** * Array index and length * * Array index: E5 Section 15.4 * Array length: E5 Section 15.4.5.1 steps 3.c - 3.d (array length write) * * The DUK_HSTRING_GET_ARRIDX_SLOW() and DUK_HSTRING_GET_ARRIDX_FAST() macros * call duk_js_to_arrayindex_string_helper().
label: code-design
50605. * Internal helpers for managing object 'length'
50606. biased exp == 0 -> denormal, exp -1022
50607. Oxdeadbeef in decimal
50608. strings up to this length are not cached
50609. DUK_HEAPHDR_H_INCLUDED
50610. [... key_obj key val]
50611. Checking callability of the immediate target * is important, same for constructability. * Checking it for functions down the bound * function chain is not strictly necessary * because .bind() should normally reject them. * But it's good to check anyway because it's * technically possible to edit the bound function * chain via internal keys.
50612. **comment:** * Three possible element formats: * 1)PropertyName : AssignmentExpression * 2) getPropertyName () { FunctionBody } * 3) setPropertyName (PropertySetParameterList) { FunctionBody } * * PropertyName can be IdentifierName (includes reserved words), a string * literal, or a number literal. Note that IdentifierName allows 'get' and * 'set' too, so we need to look ahead to the next token to distinguish: * * { get : 1 } * * and * * { get foo() { return 1 } } * { get get() { return 1 } } // 'get' as getter propertyname * * Finally, a trailing comma is allowed. * * Key name is coerced to string at compile time (and ends up as a * string constant) even for numeric keys (e.g. "{1:foo}"). * These could be emitted using e.g. LDINT, but that seems hardly * worth the effort and would increase code size.
label: code-design
50613. keep func->h_argnames; it is fixed for all passes
50614. num entries of new func at entry
50615. One digit octal escape, digit validated.
50616. Use result value as is.
50617. step 15.a
50618. formal arg limits
50619. Use the approach described in "Remarks" of FileTimeToLocalFileTime: * http://msdn.microsoft.com/en-us/library/windows/desktop/ms724277(v=vs.85).aspx
50620. [offset noAssert], when ftype != DUK_FLD_VARINT
50621. bufwriter for temp accumulation
50622. normal comparison; known: * - both x and y are not NaNs (but one of them can be) * - both x and y are not zero (but one of them can be) * - x and y may be denormal or infinite
50623. -1 if invalid
50624. convenience helpers
50625. * Use Object.defineProperty() helper for the actual operation.
50626. Ensure compact use of temps.
50627. * Object handling: property access and other support functions.
50628. allocated from valstack (fixed buffer)
50629. Unlike most built-ins, the internal [[PrimitiveValue]] of a Date * is mutable.
50630. DUK_TVAL_H_INCLUDED
50631. * Retry with several GC attempts. Initial attempts are made without * emergency mode; later attempts use emergency mode which minimizes * memory allocations forcibly.
50632. last component

50633. lexing (tokenization) state (contains two valstack slot indices)
50634. * Update an existing property of the base object.
50635. update length (curr points to length, and we assume it's still valid)
50636. in-place setup
50637. Currently all built-in native functions are strict. * duk_push_c_function() now sets strict flag, so * assert for it.
50638. The lo/hi indices may be crossed and hi < 0 is possible at entry.
50639. * Init compilation context
50640. add E
50641. '\xffThread'
50642. Must use memmove() because copy area may overlap (source and target * buffer may be the same, or from different slices).
50643. * Special name handling
50644. If function creation fails due to out-of-memory, the data buffer * pointer may be NULL in some cases. That's actually possible for * GC code, but shouldn't be possible here because the incomplete * function will be unwound from the value stack and never instantiated.
50645. A -> register of target object * B -> first register of key/value pair list * C -> number of key/value pairs
50646. **comment:** XXX: this is a major target for size optimization
 label: code-design
50647. 'constructor' property for all built-in objects (which have it) has attributes: * [[Writable]] = true, * [[Enumerable]] = false, * [[Configurable]] = true
50648. XXX: check num_args
50649. pop 'name'
50650. Here 'p' always points to the start of a term. * * We can also unconditionally reset q_last here: if this is * the last (non-empty) term q_last will have the right value * on loop exit.
50651. * All done, switch properties ('p') allocation to new one.
50652. * Duktape.Buffer, Node.js Buffer, and Khronos/ES6 TypedArray built-ins
50653. decode '%xx' to '%xx' if decoded char in reserved set
50654. **comment:** XXX: Finalizer lookup should traverse the prototype chain (to allow * inherited finalizers) but should not invoke accessors or proxy object * behavior.
 At the moment this lookup will invoke proxy behavior, so * caller must ensure that this function is not called if the target is * a Proxy.
 label: code-design
50655. Maximum number of breakpoints. Only breakpoints that are set are * consulted so increasing this has no performance impact.
50656. any other printable -> as is
50657. Called by duk_hstring.h macros
50658. step type: none, step into, step over, step out
50659. **comment:** XXX: faster implementation
 label: code-design
50660. cleared before entering catch part
50661. Clamp an input byte length (already assumed to be within the nominal * duk_hbufferobject 'length') to the current dynamic buffer limits to * yield a byte length limit that's safe for memory accesses. This value * can be invalidated by any side effect because it may trigger a user * callback that resizes the underlying buffer.
50662. **comment:** XXX: ignored now
 label: code-design
50663. **comment:** * Tagged type definition (duk_tval) and accessor macros. * * Access all fields through the accessor macros, as the representation * is quite tricky. * * There are two packed type alternatives: an 8-byte representation * based on an IEEE double (preferred for compactness), and a 12-byte * representation (portability). The latter is needed also in e.g. * 64-bit environments (it usually pads to 16 bytes per value). * * Selecting the tagged type format involves many trade-offs (memory * use, size and performance of generated code, portability, etc), * see doc/types.rst for a detailed discussion (especially of how the * IEEE double format is used to pack tagged values). * * NB: because macro arguments are often expressions, macros should * avoid evaluating their argument more than once.
 label: code-design
50664. * Automatically generated by extract_caseconv.py, do not edit!
50665. * Support functions for duk_heap.
50666. DUK_HCOMPILEDFUNCTION_H_INCLUDED
50667. write_to_array_part:
50668. **comment:** * Parse variant 1 or 2. The first part expression (which differs * in the variants) has already been parsed and its code emitted. * * reg_temps + 0: unused * reg_temps + 1: unused
 label: code-design
50669. E5 Section 15.5.5.1
50670. own pointer
50671. Note: although there is no 'undefined' literal, undefined * values can occur during compilation as a result of e.g. * the 'void' operator.
50672. Note: this may cause a corner case situation where a finalizer * may see a currently reachable activation whose 'func' is NULL.
50673. # total registers target function wants on entry (< 0 => never for ecma calls)
50674. bytecode has a stable pointer
50675. * Thread builts
50676. overwrite str1
50677. **comment:** XXX: exposed duk_debug_read_buffer
 label: code-design
50678. DUK_JS_BYTECODE_H_INCLUDED
50679. Convert a duk_tval number (caller checks) to a 32-bit index. Returns * DUK__NO_ARRAY_INDEX if the number is not whole or not a valid array * index.
50680. SCANBUILD: scan-build produces a NULL pointer dereference warning * below; it never actually triggers because holder is actually never * NULL.
50681. Set up pointers to the new property area: this is hidden behind a macro * because it is memory layout specific.
50682. no need for refcount update
50683. **comment:** Zero 'count' is also allowed to make call sites easier. * Arithmetic in bytes generates better code in GCC.
 label: code-design
50684. compiler ensures this
50685. [... old_result result] -> [... result]
50686. **comment:** This shouldn't be necessary, but check just in case * to avoid any chance of overruns.
 label: code-design
50687. At the moment Buffer<str> will just use the string bytes as * is (ignoring encoding), so we return the string length here * unconditionally.
50688. * concat()
50689. different memory layout, alloc size, and init
50690. t2 = (* (+ r m+) B)
50691. exponent non-negative (and thus not minimum exponent)
50692. default: NULL, length 0
50693. env is closed, should be missing _Callee, _Thread, _Regbase
50694. **comment:** The E5.1 algorithm checks whether or not a decoded codepoint * is below 0x80 and perhaps may be in the "reserved" set. * This seems pointless because the single byte UTF-8 case is * handled separately, and non-shortest encodings are rejected. * So, 'cp' cannot be below 0x80 here, and thus cannot be in * the reserved set.
 label: code-design
50695. force_flag
50696. Used for minimal 'const': initializer required.
50697. * Table for base-64 encoding
50698. Note: no allocation pressure, no need to check refcounts etc

50699. args incorrect
50700. write on next loop
50701. * Helper to format a time value into caller buffer, used by logging. * 'out_buf' must be at least DUK_BL_DATE_ISO8601_BUFSIZE long.
50702. **comment:** XXX: the returned value is exotic in ES6, but we use a * simple object here with no prototype. Without a prototype, * [[DefaultValue]] coercion fails which is abit confusing. * No callable check/handling in the current Proxy subset.
label: code-design
50703. 2nd related value (type specific)
50704. **comment:** Extraop arithmetic opcodes must have destination same as * first source. If second source matches destination we need * a temporary register to avoid clobbering the second source. * * XXX: change calling code to avoid this situation in most cases.
label: code-design
50705. cannot grow
50706. 0=indexOf, 1=lastIndexOf
50707. bump up "allocated" reg count, just in case
50708. Strict standard behavior, ignore trailing elements for * result 'length'.
50709. For errors a string coerced result is most informative * right now, as the debug client doesn't have the capability * to traverse the error object.
50710. DUK_TOK_IF
50711. **comment:** NOTE: This is a bit fragile. It's important to ensure that * duk_debug_process_messages() never throws an error or * act->curr_pc will never be reset.
label: code-design
50712. Working with the pointer and current size.
50713. object has any exotic behavior(s)
50714. DUK_USE_STRTAB_PROBE
50715. written by a previous RESUME handling
50716. **comment:** * Macro support functions (use only macros in calling code)
label: code-design
50717. string 1 of token (borrowed, stored to ctx->slot1_idx)
50718. **comment:** XXX: more accurate?
label: code-design
50719. Note: assume array part is comprehensive, so that either * the write goes to the array part, or we've abandoned the * array above (and will not come here).
50720. level string
50721. sufficient for keeping temp reg numbers in check
50722. All other object types.
50723. **comment:** Flip highest bit of each byte which changes * the bit pattern 10xxxxxx into 00xxxxxx which * allows an easy bit mask test.
label: test
50724. mixed endian
50725. min_new_size
50726. length in bytes (not counting NUL term)
50727. No finalizers for ROM objects
50728. last element that was left in the heap
50729. compact; no need to seal because object is internal
50730. 0x90...0x9f
50731. const flag for C
50732. [... obj] -> [...]
50733. number of pairs in current MPUTOBJ set
50734. escaped NonEscapeCharacter
50735. Module id requested
50736. Empty searchstring always matches; cpos must be clamped here. * (If q_blen were < 0 due to clamped coercion, it would also be * caught here.)
50737. Never executed if new size is smaller.
50738. 'protected'
50739. because arg count is 1
50740. 'if'
50741. finally free the struct itself
50742. !DUK_USE_PARANOI_ERRORS
50743. Array or Array-like
50744. coerce towards zero
50745. must not be extensible
50746. **comment:** We've ensured space for one escaped input; then * bail out and recheck (this makes escape handling * quite slow but it's uncommon).
label: code-design
50747. mark-and-sweep: finalizable (on current pass)
50748. [...] replacer match [captures] match_char_offset input]
50749. DUK_OP_TRYCATCH flags in A
50750. single string token
50751. Use byte copy.
50752. Most common cases first.
50753. no point in supporting encodings of 5 or more bytes
50754. '|'
50755. use_prev_pc
50756. We need ' nbytes' even for a failed offset; return value must be * (offset + nbytes) even when write fails due to invalid offset.
50757. Return value handling.
50758. Enable DUKFUNC exotic behavior once properties are set up.
50759. duk.bi_duk_object_yield() and duk.bi_duk_object_resume() ensure all of these are met
50760. Assumes that caller has normalized NaNs, otherwise trouble ahead.
50761. 0x00: finish (not part of number) * 0x01: continue
50762. **comment:** this is relatively expensive
label: code-design
50763. num_values and temp_start reset at top of outer loop
50764. **comment:** * Panic error * * Panic errors are not relative to either a heap or a thread, and cause * DUK_PANIC() macro to be invoked. Unless a user provides DUK_USE_PANIC_HANDLER, * DUK_PANIC() calls a helper which prints out the error and causes a process * exit. * * The user can override the macro to provide custom handling. A macro is * used to allow the user to have inline panic handling if desired (without * causing a potentially risky function call). * * Panics are only used in debug code such as assertions, and by the default * fatal error handler.
label: code-design
50765. specific assert for wrapping
50766. Could also rely on native sprintf(), but it will handle * values like NaN, Infinity, -0, exponent notation etc in * a JSON-incompatible way.
50767. [arg1 ... argN this loggerLevel loggerName buffer 'raw' buffer]
50768. **comment:** Function declaration for global/eval code is emitted even * for duplicates, because of E5 Section 10.5, step 5.e of * E5.1 (special behavior for variable bound to global object). * * DECLVAR will not re-declare a variable as such, but will * update the binding value.
label: code-design
50769. bits 2...5: type
50770. As a first approximation, buffer values are coerced to strings * for addition. This means that adding two buffers currently * results in a string.

50771. must be ecmascript
50772. 'toUTCString'
50773. Note: no need to re-lookup tv, conversion is side effect free
50774. require.id of current module
50775. low 32 bits is complete
50776. time when status/peek was last done (Date-based rate limit)
50777. [obj key value desc value]
50778. 'caller'
50779. rbp_flags
50780. 'data'
50781. 'type'
50782. **comment:** XXX: shared strings
 label: code-design
50783. -> [... source]
50784. default value
50785. output 3 bytes from 't'
50786. Special flags checks. Since these strings are always * reachable and a string cannot appear twice in the string * table, there's no need to check/set these flags elsewhere. * The 'internal' flag is set by string intern code.
50787. loop iterator init and limit changed from standard algorithm
50788. enum_flags
50789. '++' or '--' in a post-increment/decrement position, * and a LineTerminator occurs between the operator and * the preceding expression. Force the previous expr * to terminate, in effect treating e.g. "a,b\n++" as * "a,b;++" (= SyntaxError).
50790. by default, use caller's environment
50791. * Note: we currently assume that the setjmp() catchpoint is * not re-entrant (longjmp() cannot be called more than once * for a single setjmp()). * * See doc/code-issues.rst for notes on variable assignment * before and after setjmp().
50792. value resides in 'valstack_idx'
50793. Verbose errors with key/value summaries (non-paranoid) or without key/value * summaries (paranoid, for some security sensitive environments), the paranoid * vs. non-paranoid distinction affects only a few specific errors.
50794. 'typeof' must handle unresolvable references without throwing * a ReferenceError (E5 Section 11.4.3). Register mapped values * will never be unresolvable so special handling is only required * when an identifier is a "slow path" one.
50795. free inner references (these exist e.g. when external * strings are enabled)
50796. The 'this' after 'sep' will get ToString() coerced by * duk_join() automatically. We don't want to do that * coercion when providing .fileName or .lineNumber (GH-254).
50797. surrogate pairs get encoded here
50798. 11: getUTCMonth
50799. **comment:** XXX: any chance of unifying this with the 'length' key handling?
 label: code-design
50800. assert just a few critical flags
50801. correction
50802. default to false
50803. * Stack slice primitives
50804. Length in elements: take into account shift, but * intentionally don't check the underlying buffer here.
50805. [key val] -> []
50806. DUK_USE_STRHASH_DENSE
50807. * Get old and new length
50808. clipped codepoint
50809. h_new_proto may be NULL
50810. -> [... errhandler errval]
50811. this is the case for normalized numbers
50812. flags: "gim"
50813. Starting from this round, use emergency mode * for mark-and-sweep.
50814. The eval code is executed within the lexical environment of a specified * activation. For now, use global object eval() function, with the eval * considered a 'direct call to eval'. * * Callstack level for debug commands only affects scope -- the callstack * as seen by, e.g. Duktape.act() will be the same regardless.
50815. * Found existing own or inherited plain property, but original * base is a primitive value.
50816. optional callstack level
50817. e.g. duk_push_string_file_raw() pushed undefined
50818. Assume that either all memory funcs are NULL or non-NUL, mixed * cases will now be unsafe.
50819. 'error'
50820. don't throw for prototype loop
50821. (< (* r 2) s)
50822. [thisArg arg1 ... argN func] (thisArg+args == nargs total)
50823. Note: not set in template (has no "prototype")
50824. unreferenced w/o asserts
50825. if (is_repl_func)
50826. precision:shortest
50827. don't resize stringtable (but may sweep it); needed during stringtable resize
50828. Write curr_pc back for the debugger.
50829. Note: duk_dec_req_stridx() backtracks one char
50830. 'length'
50831. -> [... val retval]
50832. Object property access
50833. * The specification uses RegExp [[Match]] to attempt match at specific * offsets. We don't have such a primitive, so we use an actual RegExp * and tweak lastIndex. Since the RegExp may be non-global, we use a * special variant which forces global-like behavior for matching.
50834. 3
50835. Apply timezone offset to get the main parts in UTC
50836. * Zero sign, see misc/tcc_zerosign2.c.
50837. * Boolean built-ins
50838. LeftHandSideExpression does not allow empty expression
50839. **comment:** XXX: add proper spare handling to dynamic buffer, to minimize * reallocs; currently there is no spare at all.
 label: code-design
50840. Note: escaped characters differentiate directives
50841. defaults, E5 Section 8.6.1, Table 7
50842. * Intern the temporary byte buffer into a valstack slot * (in practice, slot1 or slot2).
50843. special RegExp literal handling after IdentifierName
50844. XXX: byte offset?
50845. 'switch'
50846. * The 'duktape.h' header provides the public API, but also handles all * compiler and platform specific feature detection, Duktape feature * resolution, inclusion of system headers, etc. These have been merged * because the public API is also dependent on e.g. detecting appropriate * C types which is quite platform/compiler

specific especially for a non-C99 * build. The public API is also dependent on the resolved feature set. * * Some actions taken by the merged header (such as including system headers) * are not appropriate for building a user application. The define * DUK_COMPILING_DUKTAPE allows the merged header to skip/include some * sections depending on what is being built.

50847. * Check the function type, handle bound function chains, and prepare * parameters for the rest of the call handling. Also figure out the * effective 'this' binding, which replaces the current value at * idx_func + 1. * * If the target function is a 'bound' one, follow the chain of 'bound' * functions until a non-bound function is found. During this process, * bound arguments are 'prepended' to existing ones, and the "this" * binding is overridden. See E5 Section 15.3.4.5.1. * * Lightfunc detection happens here too. Note that lightweight functions * can be wrapped by (non-lightweight) bound functions so we must resolve * the bound function chain first.

50848. avoid side effects

50849. result in csreg

50850. * Other heap related defines

50851. * Start doing property attributes updates. Steps 12-13. * * Start by computing new attribute flags without writing yet. * Property type conversion is done above if necessary.

50852. strict: non-deletable, non-writable

50853. * Assuming a register binds to a variable declared within this * function (a declarative binding), the 'this' for the call * setup is always 'undefined'. E5 Section 10.2.1.1.6.

50854. This may happen when forward and backward scanning disagree * (possible for non-extended-UTF-8 strings).

50855. use fallback as 'this' value

50856. Caller has already eaten the first character '(' which we don't need.

50857. r <- (* r B) * s <- s * m+ <- (* m+ B) * m- <- (* m- B) * k <- (- k 1)

50858. A -> object reg * B -> key reg/const * C -> value reg/const * * Note: intentional difference to register arrangement * of e.g. GETPROP; 'A' must contain a register-only value.

50859. C recursion check.

50860. b

50861. Misc

50862. catches EOF (0x00)

50863. **comment:** XXX: duk_get_length?
label: code-design

50864. XXX: if assertions enabled, walk through all valid PCs * and check line mapping.

50865. 'defineProperty'

50866. **comment:** XXX: awkward and bloated asm -- use faster internal accesses
label: code-design

50867. approximation, close enough

50868. marker for detecting internal "double faults", see duk_error_throw.c

50869. three flags

50870. No difference between raw/ensure because the buffer shrinks.

50871. * Determining which datetime components to overwrite based on * stack arguments is a bit complicated, but important to factor * out from setters themselves for compactness. * * If DUK_DATE_FLAG_TIMESETTER, maxnargs indicates setter type: * * 1 -> millisecond * 2 -> second, [millisecond] * 3 -> minute, [second], [millisecond] * 4 -> hour, [minute], [second], [millisecond] * * Else: * * 1 -> date * 2 -> month, [date] * 3 -> year, [month], [date] * * By comparing nargs and maxnargs (and flags) we know which * components to override. We rely on part index ordering.

50872. False if the object is NULL or the prototype 'p' is NULL. * In particular, false if both are NULL (don't compare equal).

50873. Start filling in the activation

50874. SCANBUILD: complains about use of uninitialized values. * The complaint is correct, but operating in undefined * values here is intentional in some cases and the caller * ignores the results.

50875. prevent duk_expr_led() by using a binding power less than anything valid

50876. borrowed reference; although 'tv1' comes from a register, * its value was loaded using LDCONST so the constant will * also exist and be reachable.

50877. [... proplist enum_obj key val]

50878. * Unix-like Date providers * * Generally useful Unix / POSIX / ANSI Date providers.

50879. Push the current 'this' binding; throw TypeError if binding is not object * coercible (CheckObjectCoercible).

50880. All strings beginning with 0xff are treated as "internal", * even strings interned by the user. This allows user code to * create internal properties too, and makes behavior consistent * in case user code happens to use a string also used by Duktape * (such as string has already been interned and has the 'internal' * flag set).

50881. 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx [31 bits]

50882. shrink failure is not fatal

50883. !(l < p)

50884. valstack index of loop detection object

50885. limit is quite low: one array entry is 8 bytes, one normal entry is 4+1+8+4 = 17 bytes (with hash entry)

50886. yes -> move back to heap allocated

50887. strict flag for putvar comes from our caller (currently: fixed)

50888. catchstack limits

50889. * Manually optimized number-to-double conversion

50890. [ToObject(this) ToUint32(length) lowerValue upperValue]

50891. if a tail call: * - an Ecmascript activation must be on top of the callstack * - there cannot be any active catchstack entries

50892. [... enum_target res trap_result val true]

50893. Helper for component getter calls: check 'this' binding, get the * internal time value, split it into parts (either as UTC time or * local time), push a specified component as a return value to the * value stack and return 1 (caller can then tail call us).

50894. important for callers

50895. fail

50896. [sep ToObject(this) len]

50897. DUK_TOK_PERIOD

50898. Enumeration keys are checked against the enumeration target (to see * that they still exist). In the proxy enumeration case _Target will * be the proxy, and checking key existence against the proxy is not * required (or sensible, as the keys may be fully virtual).

50899. Easiest way to implement the search required by the specification * is to do a RegExp test() with lastIndex forced to zero. To avoid * side effects on the argument, "clone" the RegExp if a RegExp was * given as input. * * The global flag of the RegExp should be ignored; setting lastIndex * to zero (which happens when "cloning" the RegExp) should have an * equivalent effect.

50900. idx_reviver

50901. **comment:** Require a lot of stack to force a value stack grow/shrink. * Recursive mark-and-sweep is prevented by allocation macros * so this won't trigger another mark-and-sweep.
label: code-design

50902. signed shift

50903. Impose a string maximum length, need to handle overflow * correctly.

50904. **comment:** XXX: casts could be improved, especially for GET/SET DATA
label: code-design

50905. bytes left

50906. **comment:** XXX: these last tricks are unnecessary if the function is made * a genuine native function.
label: code-design

50907. No locale specific formatter; this is OK, we fall back * to ISO 8601.

50908. failed

50909. -> [... key]

50910. **comment:** Tested: not faster on x64
 label: code-design
50911. clamp to 10 chars
50912. * "LineTerminator" production check.
50913. A lightfunc might also inherit a .toJSON() so just bail out.
50914. **comment:** Note: this is very tricky; we must never 'overshoot' the * correction downwards.
 label: code-design
50915. **comment:** XXX: inefficient; block insert primitive
 label: code-design
50916. * Exposed helper for setting up heap longjmp state.
50917. in non-packed representation we don't care about which NaN is used
50918. **comment:** Period followed by a digit can only start DecimalLiteral * (handled in slow path). We could jump straight into the * DecimalLiteral handling but should avoid goto to inside * a block.
 label: code-design
50919. zero
50920. Handle change in value stack top. Respect value stack * initialization policy: 'undefined' above top. Note that * DECREF may cause a side effect that reallocates valstack, * so must relookup after DECREF.
50921. **comment:** XXX: tostring?
 label: code-design
50922. 'func' wants stack "as is"
50923. * Debug message processing
50924. Bottom of valstack for this activation, used to reset * valstack_bottom on return; index is absolute. Note: * idx_top not needed because top is set to 'nregs' always * when returning to an EcmaScript activation.
50925. * Sweep heap
50926. Object.keys
50927. map is reachable through obj
50928. **comment:** XXX: this needs to be reworked so that we never shrink the value * stack on function entry so that we never need to grow it here. * Needing to grow here is a sandboxing issue because we need to * allocate which may cause an error in the error handling path * and thus propagate an error out of a protected call.
 label: code-design
50929. **comment:** * Note: since this is an exposed API call, there should be * no way a mark-and-sweep could have a side effect on the * memory allocation behind 'ptr'; the pointer should never * be something that Duktape wants to change. * * Thus, no need to use DUK_REALLOC_INDIRECT (and we don't * have the storage location here anyway).
 label: code-design
50930. required for rehash to succeed, equality not that useful
50931. DUK_TOK_CLASS
50932. **comment:** unused for label
 label: code-design
50933. 'this' binding shouldn't matter here
50934. **comment:** remove old value
 label: code-design
50935. **comment:** get rid of the strings early to minimize memory use before intern
 label: code-design
50936. always in entry part, no need to look up parents etc
50937. * Raw intern and lookup
50938. Copy the property table verbatim; this handles attributes etc. * For ROM objects it's not necessary (or possible) to update * refcounts so leave them as is.
50939. CONDITIONAL EXPRESSION
50940. swap elements; deal with non-existent elements correctly
50941. [... closure/error]
50942. * Create a new property in the original object. * * Exotic properties need to be reconsidered here from a write * perspective (not just property attributes perspective). * However, the property does not exist in the object already, * so this limits the kind of exotic properties that apply.
50943. Initial value for highest_idx is -1 coerced to unsigned. This * is a bit odd, but (highest_idx + 1) will then wrap to 0 below * for out_min_size as intended.
50944. ignore
50945. **comment:** XXX: combine all the integer conversions: they share everything * but the helper function for coercion.
 label: code-design
50946. vararg function, thisArg needs special handling
50947. Assert for value stack initialization policy.
50948. **comment:** * Shallow fast path checks for accessing array elements with numeric * indices. The goal is to try to avoid coercing an array index to an * (interned) string for the most common lookups, in particular, for * standard Array objects. * * Interning is avoided but only for a very narrow set of cases: * - Object has array part, index is within array allocation, and * value is not unused (= key exists) * - Object has no interfering exotic behavior (e.g. arguments or * string object exotic behaviors interfere, array exotic * behavior does not). * * Current shortcoming: if key does not exist (even if it is within * the array allocation range) a slow path lookup with interning is * always required. This can probably be fixed so that there is a * quick fast path for non-existent elements as well, at least for * standard Array objects.
 label: code-design
50949. curr_token slot1 (matches 'lex' slot1_idx)
50950. eat 'break' or 'continue'
50951. zero-based -> one-based
50952. roughly 512 bytes
50953. * indexOf() and lastIndexOf()
50954. DUK_USE_HEAPPTR16
50955. regexp literal must not follow this token
50956. qmin and qmax will be 0 or 1
50957. same as during entry
50958. Have a catch variable.
50959. Note: in strict mode the compiler should reject explicit * declaration of 'eval' or 'arguments'. However, internal * bytecode may declare 'arguments' in the function prologue. * We don't bother checking (or asserting) for these now.
50960. Object property allocation layout
50961. Node.js accepts only actual Arrays.
50962. [requested_id require require.id resolved_id last_comp Duktape.modLoaded undefined]
50963. **comment:** XXX: Shrink the stacks to minimize memory usage? May not * be worth the effort because terminated threads are usually * garbage collected quite soon.
 label: code-design
50964. * Actual Object.defineProperty() default algorithm.
50965. t1 = milliseconds within day (fits 32 bit) * t2 = day number from epoch (fits 32 bit, may be negative)
50966. 'class'
50967. compare: similar to string comparison but for buffer data.
50968. Note: if 'x' is zero, x->n becomes 0 here
50969. Debugger protocol version is defined in the public API header.
50970. not undefined

50971. Endianness indicator
50972. number of activation records in callstack preventing a yield
50973. DUK_TOK_IMPLEMENT
50974. 4'294'967'295
50975. XXX: switch cast?
50976. 'trace'
50977. array 'length' is always a number, as we coerce it
50978. NULL is allowed, no output
50979. * Convert duk_compiler_func to a function template
50980. Update duk_tval in-place if pointer provided and the * property is writable. If the property is not writable * (immutable binding), use duk_hobject_putprop() which * will respect mutability.
50981. Better equivalent algorithm. If the compiler is compliant, C and * Ecmascript semantics are identical for this particular comparison. * In particular, NaNs must never compare equal and zeroes must compare * equal regardless of sign. Could also use a macro, but this inlines * already nicely (no difference on gcc, for instance).
50982. Is whole and within 32 bit range. If the value happens to be 0xFFFFFFFF, * it's not a valid array index but will then match DUK__NO_ARRAY_INDEX.
50983. * PLAIN: x1 * ARITH: x1 <op> x2 * PROP: x1.x2 * VAR: x1 (name)
50984. eat closing slash
50985. * Regexp range tables
50986. Value stack intentionally mixed size here.
50987. Computation must not wrap; this limit works for 32-bit size_t: * >>> srclen = 3221225469 * >>> '%x' % ((srclen + 2) / 3 * 4) * 'fffffc'
50988. **comment:** * Special post-tweaks, for cases not covered by the init data format. * * - Set Date.prototype.toGMTString to Date.prototype.toUTCString. * toGMTString is required to have the same Function object as * toUTCString in E5 Section B.2.6. Note that while Smjs respects * this, V8 does not (the Function objects are distinct). * * - Make DoubleError non-extensible. * * - Add info about most important effective compile options to Duktape. * * - Possibly remove some properties (values or methods) which are not * desirable with current feature options but are not currently * conditional in init data.
label: code-design
50989. hobject property layout
50990. **comment:** UNUSED, intentionally empty
label: code-design
50991. double quote or backslash
50992. track utf-8 non-continuation bytes
50993. [hobject props enum(props)]
50994. 22 to 31
50995. **comment:** XXX: very similar to DUK_IVAL_ARITH - merge?
label: code-design
50996. **comment:** XXX: optional check, match_caps is zero if no regexp, * so dollar will be interpreted literally anyway.
label: code-design
50997. * Internal helper to define a property with specific flags, ignoring * normal semantics such as extensibility, write protection etc. * Overwrites any existing value and attributes unless caller requests * that value only be updated if it doesn't already exists. * * Does not support: * - virtual properties (error if write attempted) * - getter/setter properties (error if write attempted) * - non-default (!= WEC) attributes for array entries (error if attempted) * - array abandoning: if array part exists, it is always extended * - array 'length' updating * * Stack: [... in_val] -> [] * * Used for e.g. built-in initialization and environment record * operations.
50998. -> [... enum key enum_target val]
50999. constants arbitrary, chosen for small loads
51000. reset 'allow_in' for parenthesized expression
51001. **comment:** Internal class is Object: Object.prototype.toString.call(new Buffer(0)) * prints "[object Object]".
label: code-design
51002. * The entire allocated buffer area, regardless of actual used * size, is kept zeroed in resizes for simplicity. If the buffer * is grown, zero the new part.
51003. XXX: integrate support for this into led() instead? * Similar issue as post-increment/post-decrement.
51004. E5 Sections 11.8.2, 11.8.5; x > y --> y < x
51005. if present, forces 16-byte duk_tval
51006. * 'arguments' binding is special; if a shadowing argument or * function declaration exists, an arguments object will * definitely not be needed, regardless of whether the identifier * 'arguments' is referenced inside the function body.
51007. exports (this binding)
51008. disabled
51009. **comment:** XXX: other variants like uint, u32 etc
label: code-design
51010. rely on interning, must be this string
51011. Does not allow e.g. 2**31-1, but one more would allow overflows of u32.
51012. Note: cannot be a bound function either right now, * this would be easy to relax though.
51013. Match labels starting from latest; once label_id no longer matches, we can * safely exit without checking the rest of the labels (only the topmost labels * are ever updated).
51014. The 'else' ambiguity is resolved by 'else' binding to the innermost * construct, so greedy matching is correct here.
51015. Short circuit if is safe: if act->curr_pc != NULL, 'fun' is * guaranteed to be a non-NULL Ecmascript function.
51016. read-only values 'lifted' for ease of use
51017. return value
51018. 'source'
51019. 'clog'
51020. digits
51021. * Statement value handling. * * Global code and eval code has an implicit return value * which comes from the last statement with a value * (technically a non- "empty" continuation, which is * different from an empty statement). * * Since we don't know whether a later statement will * override the value of the current statement, we need * to coerce the statement value to a register allocated * for implicit return values. In other cases we need * to coerce the statement value to a plain value to get * any side effects out (consider e.g. "foo.bar;").
51022. is_frozen
51023. **comment:** Downgrade checks are not made everywhere, so 'length' is not always * a fastint (it is a number though). This can be removed once length * is always guaranteed to be a fastint.
label: code-design
51024. if duk_uint32_t is exactly 32 bits, this is a NOP
51025. * Not found as a concrete property, check whether a String object * virtual property matches.
51026. * Function formal arguments, always bound to registers * (there's no support for shuffling them now).
51027. off >= step, and step >= 1
51028. zero-width joiner
51029. Rethrow error to calling state.
51030. ""
51031. next temporary register to allocate
51032. * For bound objects, [[HasInstance]] just calls the target function * [[HasInstance]]. If that is again a bound object, repeat until * we find a non-bound Function object.
51033. Pointer and buffer primitive values are treated like other * primitives values which have a fully fledged object counterpart: * promote to an object value. Lightfuncs are coerced with * ToObject() even they could also be returned as is.
51034. **comment:** XXX: option to fix opcode length so it lines up nicely

label: code-design

51035. xxx -> DUK_HBUFFEROBJECT_ELEM_UINT16

51036. relookup after side effects (no side effects currently however)

51037. Note: this order matters (final value before deleting map entry must be done)

51038. 'arguments'

51039. * Fast path for bufferobject getprop/putprop

51040. XXX: shrink array allocation on entries compaction here?

51041. inline arithmetic check for constant values

51042. pre-incremented, points to first jump slot

51043. * Helpers for duk_compiler_func.

51044. Note: need to re-lookup because ToNumber() may have side effects

51045. expr_flags

51046. **comment:** XXX: exposed duk_debug_read_hbuffer

label: code-design

51047. accept string

51048. unreferenced without assertions

51049. **comment:** XXX: macro checks for array index flag, which is unnecessary here

label: code-design

51050. **comment:** XXX: this algorithm could be optimized quite a lot by using e.g. * a logarithm based estimator for 'k' and performing B^n multiplication * using a lookup table or using some bit-representation based exp * algorithm. Currently we just loop, with significant performance * impact for very large and very small numbers.

label: code-design

51051. 'toString'

51052. first character has been matched

51053. 0xffff0 is -Infinity

51054. **comment:** XXX: don't want to shrink allocation here

label: code-design

51055. -> [... error]

51056. Lightfunc coerces to a Function instance with concrete * properties. Since 'length' is virtual for Duktape/C * functions, don't need to define that. * * The result is made extensible to mimic what happens to * strings: * > Object.isExtensible(Object('foo')) * true

51057. curr_token follows 'function'

51058. * Update preventcount

51059. **comment:** These are not needed to implement quantifier capture handling, * but might be needed at some point.

label: requirement

51060. 'toJSON'

51061. function executes as a constructor (called via "new")

51062. heapdr: * - is not reachable * - is an object * - is not a finalized object * - has a finalizer

51063. DUK_ERROR_H_INCLUDED

51064. Number/string-or-buffer -> coerce string to number (e.g. "'1.5' == 1.5" -> true).

51065. XXX: There are several limitations in the current implementation for * strings with $\geq 0x80000000UL$ characters. In some cases one would need * to be able to represent the range [-0xffffffff,0xffffffff] and so on. * Generally character and byte length are assumed to fit into signed 32 * bits ($< 0x80000000UL$). Places with issues are not marked explicitly * below in all cases, look for signed type usage (duk_int_t etc) for * offsets/lengths.

51066. * Reset function state and perform register allocation, which creates * 'varmap' for second pass. Function prologue for variable declarations, * binding value initializations etc is emitted as a by-product. * * Strict mode restrictions for duplicate and invalid argument * names are checked here now that we know whether the function * is actually strict. See: test-dev-strict-mode-boundary.js. * * Inner functions are compiled during pass 1 and are not reset.

51067. if 1, doing a fixed format output (not free format)

51068. We want to do a straight memory copy if possible: this is * an important operation because .set() is the TypedArray * way to copy chunks of memory. However, because set() * conceptually works in terms of elements, not all views are * compatible with direct byte copying. * * If we do manage a direct copy, the "overlap issue" handled * below can just be solved using memmove() because the source * and destination element sizes are necessarily equal.

51069. borrowed; NULL if no step state (NULLed in unwind)

51070. new buffer with string contents

51071. Allow very small lenience because some compilers won't parse * exact IEEE double constants (happened in matrix testing with * Linux gcc-4.8 -m32 at least).

51072. We could fit built-in index into magic but that'd make the magic * number dependent on built-in numbering (genbuiltins.py doesn't * handle that yet). So map both class and prototype from the * element type.

51073. * Allocate a new thread. * * Leaves the built-ins array uninitialized. The caller must either * initialize a new global context or share existing built-ins from * another thread.

51074. no regexp instance should exist without a non-configurable bytecode property

51075. original input stack before nargs/nregs handling must be * intact for 'arguments' object

51076. replaces top of stack with new value if necessary

51077. 33: setUTCDate

51078. Notes: * - only numbered indices are relevant, so arr_idx fast reject is good * (this is valid unless there are more than $4^{**}32-1$ arguments). * - since variable lookup has no side effects, this can be skipped if * DUK_GETDESC_FLAG_PUSH_VALUE is not set.

51079. Automatic semicolon insertion is allowed if a token is preceded * by line terminator(s), or terminates a statement list (right curly * or EOF).

51080. eat 'for'

51081. init 'curr_token'

51082. -> [... new_global new_globalenv new_global]

51083. Index clamping is a bit tricky, we must ensure that we'll only iterate * through elements that exist and that the specific requirements from E5.1 * Sections 15.4.4.14 and 15.4.4.15 are fulfilled; especially: * * - indexOf: clamp to [-len,len], negative handling -> [0,len], * if clamped result is len, for-loop bails out immediately * * - lastIndexOf: clamp to [-len-1, len-1], negative handling -> [-1, len-1], * if clamped result is -1, for-loop bails out immediately * * If fromIndex is not given, ToInteger(undefined) = 0, which is correct * for indexOf() but incorrect for lastIndexOf(). Hence special handling, * and why lastIndexOf() needs to be a vararg function.

51084. implicit attributes

51085. allow a constant to be returned

51086. prop index in 'entry part', < 0 if not there

51087. add_auto_proto

51088. [... ToObject(this) ToUint32(length) val]

51089. Load bytecode instructions.

51090. unsupported: would consume multiple args

51091. DUK_USE_DEBUGGER_THROW_NOTIFY

51092. * Same type? * * Note: since number values have no explicit tag in the 8-byte * representation, need the awkward if + switch.

51093. There's intentionally no check for * current underlying buffer length.

51094. follow parent chain

51095. xxx -> DUK_HBUFFEROBJECT_ELEM_FLOAT64

51096. **comment:** XXX: this should be an assert

label: test

51097. without explicit non-BMP support, assume non-BMP characters * are always accepted as letters.

51098. No debugger support, just pop values.

51099. The lookup byte is intentionally sign extended to (at least) * 32 bits and then ORed. This ensures that is at least 1 byte * is negative, the highest bit of 't' will be set at the end * and we don't need to check every byte.

51100. [hobject props enum(props) key desc value? getter? setter?]

51101. start match from beginning

51102. DUK_HOBJECT_FLAG_ARRAY_PART: don't care

51103. Validity checks for various fraction formats ("0.1", ".1", "1.", ".").

51104. DUK_USE_SECTION_B

51105. !DUK_USE_NONSTD_FUNC_CALLER_PROPERTY

51106. **comment:** XXX: better to get base and walk forwards?
label: code-design

51107. 'ownKeys'

51108. **comment:** Call this.raw(msg); look up through the instance allows user to override * the raw() function in the instance or in the prototype for maximum * flexibility.
label: code-design

51109. restore stack top

51110. DUK_USE_BASE64_FASTPATH

51111. Valstack should suffice here, required on function valstack init

51112. duk_hnativefunction specific fields.

51113. r <- (* f b 2) [if b==2 -> (* f 4)] * s <- (* (expt b (- 1 e)) 2) == b^(1-e) * 2 [if b==2 -> b^(2-e)] * m+ <- b == 2 * m- <- 1 * k <- 0 * B <- B * low_ok <- round * high_ok <- round

51114. Insert bytes in the middle of the buffer from an external buffer.

51115. identifier names (E5 Section 7.6)

51116. * Set up the resolution input which is the requested ID directly * (if absolute or no current module path) or with current module * ID prepended (if relative and current module path exists). * Suppose current module is 'foo/bar' and relative path is './quux'. * The 'bar' component must be replaced so the initial input here is * 'foo/bar/../quux'.

51117. A non-Array object should not have an array part in practice. * But since it is supported internally (and perhaps used at some * point), check and abandon if that's the case.

51118. keep heap->dbg_detached_cb

51119. For tv1 == tv2, both pointing to stack top, the end result * is same as duk_pop(ctx).

51120. If input_offset were assigned a negative value, it would * result in a large positive value. Most likely it would be * larger than input_length and be caught here. In any case * no memory unsafe behavior would happen.

51121. During parsing, month and day are one-based; set defaults here.

51122. **comment:** * Logger arguments are: * * magic: log level (0-5) * this: logger * stack: plain log args * * We want to minimize memory churn so a two-pass approach * is used: first pass formats arguments and computes final * string length, second pass copies strings either into a * pre-allocated and reused buffer (short messages) or into a * newly allocated fixed buffer. If the backend function plays * nice, it won't coerce the buffer to a string (and thus * intern it).
label: code-design

51123. Non-zero refcounts should not happen because we refcount * finalize all unreachable objects which should cancel out * refcounts (even for cycles).

51124. token allows automatic semicolon insertion (eof or preceded by newline)

51125. Use special opcodes to load short strings

51126. * Add .fileName and .lineNumber to an error on the stack top.

51127. * CommonJS require() and modules support

51128. * Lexing helpers

51129. * Create built-in objects by parsing an init bitstream generated * by genbuiltins.py.

51130. explicit PropertyList

51131. Special handling for call setup instructions. The target * is expressed indirectly, but there is no output shuffling.

51132. * Create mantissa

51133. NOTE: act may be NULL if an error is thrown outside of any activation, * which may happen in the case of, e.g. syntax errors.

51134. ToNumber() for a fastint is a no-op.

51135. [key] -> [undefined] (default value)

51136. traceback depth fits into 16 bits

51137. [thisArg arg1 ... argN func boundFunc argArray]

51138. * Parser duk__advance() token eating functions

51139. **comment:** cleanup varmap from any null entries, compact it, etc; returns number * of final entries after cleanup.
label: code-design

51140. **comment:** * Needs a save of multiple saved[] entries depending on what range * may be overwritten. Because the regexp parser does no such analysis, * we currently save the entire saved array here. Lookaheads are thus * a bit expensive. Note that the saved array is not needed for just * the lookahead sub-match, but for the matching of the entire sequel. * * The temporary save buffer is pushed on to the valstack to handle * errors correctly. Each lookahead causes a C recursion and pushes * more stuff on the value stack. If the C recursion limit is less * than the value stack spare, there is no need to check the stack. * We do so regardless, just in case.
label: code-design

51141. **comment:** XXX: split into separate functions for each field type?
label: code-design

51142. Same as above but for unsigned int range.

51143. comma after a value, expected

51144. duplicate key

51145. Given a day number, determine year and day-within-year.

51146. **comment:** Vararg function: must be careful to check/require arguments. * The JSON helpers accept invalid indices and treat them like * non-existent optional parameters.
label: code-design

51147. Success path handles

51148. **comment:** XXX: getter with class check, useful in built-ins
label: code-design

51149. already updated

51150. checking for DUK__DELETED_MARKER is not necessary here, but helper does it now

51151. match: keep saves

51152. DUK_USE_PARANOIAD_MATH

51153. 'boolean'

51154. * Debugging disabled

51155. fixed offset in valstack

51156. DUK_TOK_ENUM

51157. when key is NULL, value is garbage so no need to set

51158. **comment:** * Error during call. The error value is at heap->lj.value1. * * The very first thing we do is restore the previous setjmp catcher. * This means that any error in error handling will propagate outwards * instead of causing a setjmp() re-entry above.
label: code-design

51159. Standard behavior for map(): trailing non-existent * elements don't invoke the user callback and are not * counted towards result 'length'.

51160. * Append a Unicode codepoint to the temporary byte buffer. Performs * CESU-8 surrogate pair encoding for codepoints above the BMP. * Existing surrogate pairs are allowed and also encoded into CESU-8.

51161. * Proxy helpers

51162. * Found; post-processing (Function and arguments objects)
51163. maps to finalizer 2nd argument
51164. DUK_USE_AVOID_PLATFORM_FUNCPTRS
51165. 'roundpos' is relative to nc_ctx->k and increases to the right * (opposite of how 'k' changes).
51166. key encountered as a plain property
51167. e.g. DUK_OP_POSTINCV
51168. Allow empty string to be interpreted as 0
51169. Math.pow(-0,y) where y<0 should be: * -Infinity if y<0 and an odd integer * - Infinity otherwise * NetBSD pow() returns -Infinity for all finite y<0. The * if-clause also catches y == -Infinity (which works even * without the fix).
51170. **comment:** XXX: add temporary duk_p pointer here too; sharing
label: code-design
51171. not minimum exponent
51172. * Avoid nested calls. Concretely this happens during debugging, e.g. * when we eval() an expression. * * Also don't interrupt if we're currently doing debug processing * (which can be initiated outside the bytecode executor) as this * may cause the debugger to be called recursively. Check required * for correct operation of throw intercept and other "exotic" halting * scenarios.
51173. Because new_size > duk_count_used_probe(heap), guaranteed to work
51174. bump refcount to prevent refzero during finalizer processing
51175. * Pointers to function data area for faster access. Function * data is a buffer shared between all closures of the same * "template" function. The data buffer is always fixed (non- * dynamic, hence stable), with a layout as follows: * * constants (duk_tval) * inner functions (duk_hobject *) * bytecode (duk_instr_t) * * Note: bytecode end address can be computed from 'data' buffer * size. It is not strictly necessary functionally, assuming * bytecode never jumps outside its allocated area. However, * it's a safety/robustness feature for avoiding the chance of * executing random data as bytecode due to a compiler error. * * Note: values in the data buffer must be incref'd (they will * be decref'd on release) for every compiledfunction referring * to the 'data' element.
51176. **comment:** * Advance lookup window by N characters, filling in new characters as * necessary. After returning caller is guaranteed a character window of * at least DUK_LEXER_WINDOW_SIZE characters. * * The main function duk_advance_bytes() is called at least once per every * token so it has a major lexer/compiler performance impact. There are two * variants for the main duk_advance_bytes() algorithm: a sliding window * approach which is slightly faster at the cost of larger code footprint, * and a simple copying one. * * Decoding directly from the source string would be another lexing option. * But the lookup window based approach has the advantage of hiding the * source string and its encoding effectively which gives more flexibility * going forward to e.g. support chunked streaming of source from flash. * * Decodes UTF-8/CESU-8 leniently with support for code points from U+0000 to * U+10FFFF, causing an error if the input is unparseable. Leniency means: * * * Unicode code point validation is intentionally not performed, * except to check that the codepoint does not exceed 0x10ffff. * * * In particular, surrogate pairs are allowed and not combined, which * allows source files to represent all SourceCharacters with CESU-8. * Broken surrogate pairs are allowed, as Ecmascript does not mandate * their validation. * * * Allows non-shortest UTF-8 encodings. * * Leniency here causes few security concerns because all character data is * decoded into Unicode codepoints before lexer processing, and is then * re-encoded into CESU-8. The source can be parsed as strict UTF-8 with * a compiler option. However, Ecmascript source characters include -all- * 16-bit unsigned integer codepoints, so leniency seems to be appropriate. * * Note that codepoints above the BMP are not strictly SourceCharacters, * but the lexer still accepts them as such. Before ending up in a string * or an identifier name, codepoints above BMP are converted into surrogate * pairs and then CESU-8 encoded, resulting in 16-bit Unicode data as * expected by Ecmascript. * * An alternative approach to dealing with invalid or partial sequences * would be to skip them and replace them with e.g. the Unicode replacement * character U+FFFD. This has limited utility because a replacement character * will most likely cause a parse error, unless it occurs inside a string. * Further, Ecmascript source is typically pure ASCII. * * See: * * <http://en.wikipedia.org/wiki=UTF-8> * <http://en.wikipedia.org/wiki/CESU-8> * <http://tools.ietf.org/html/rfc3629> * http://en.wikipedia.org/wiki/UTF-8#Invalid_byte_sequences * * Future work: * * * Reject other invalid Unicode sequences (see Wikipedia entry for examples) * in strict UTF-8 mode. * * * Size optimize. An attempt to use a 16-byte lookup table for the first * byte resulted in a code increase though. * * * Is checking against maximum 0x10ffff really useful? 4-byte encoding * imposes a certain limit anyway. * * * Support chunked streaming of source code. Can be implemented either * by streaming chunks of bytes or chunks of codepoints.
label: code-design
51177. duk_push_this() + CheckObjectCoercible() + duk_to_string()
51178. pointers for scanning
51179. array part size (entirely gc reachable)
51180. Steps 12 and 13: reorganize elements to make room for itemCount elements
51181. number of tokens parsed
51182. Tear down state.
51183. Don't check for Infinity unless the context allows it. * 'Infinity' is a valid integer literal in e.g. base-36: * * parseInt('Infinity', 36) * 1461559270678
51184. zero handled by caller
51185. Don't accept relative indices now.
51186. [holder name val] -> [holder]
51187. res_obj
51188. * XUTF-8 and CESU-8 encoding/decoding
51189. 1110 xxxx 10xx xxxx 10xx xxxx [16 bits]
51190. function declaration
51191. **comment:** XXX: disable error handlers for duration of compaction?
label: code-design
51192. * PUTVAR * * See E5 Sections: * 11.1.2 Identifier Reference * 10.3.1 Identifier Resolution * 11.13.1 Simple Assignment [example of where the Reference is PutValue'd] * 8.7.2 PutValue (V,W) [see especially step 3.b, undefined -> automatic global in non-strict mode] * 8.12.4 [[CanPut]] (P) * 8.12.5 [[Put]] (P) * * Note: may invalidate any valstack (or object) duk_tval pointers because * putting a value may reallocate any object or any valstack. Caller beware.
51193. ?
51194. state == 3
51195. Maximum traversal depth for "bound function" chains.
51196. **comment:** XXX: magic for getter/setter? use duk_def_prop()?
label: code-design
51197. xxx -> DUK_HBUFFEROBJECT_ELEM_INT32
51198. [thread value]
51199. * GETPROP: Ecmascript property read.
51200. **comment:** * The __FILE__ and __LINE__ information is intentionally not used in the * creation of the error object, as it isn't useful in the tracedata. The * tracedata still contains the function which returned the negative return * code, and having the file/line of this function isn't very useful.
label: code-design
51201. object is a thread (duk_hthread)
51202. 'info'
51203. add a new pending split to the beginning of the entire disjunction
51204. duk_push_this() + CheckObjectCoercible() + duk_to_object()
51205. [... arr jsonx(arr) res] -> [... res jsonx(arr)]
51206. this may have side effects, so re-lookup act
51207. Fall through to handle the rest.
51208. [... error lineNumber fileName]
51209. * Number built-ins
51210. **comment:** XXX: this compiles to over 500 bytes now, even without special handling * for an array part. Uses signed ints so does not handle full array range correctly.
label: code-design
51211. -> [... closure template env]
51212. Note: if the reg_temp load generated shuffling * instructions, we may need to rewind more than * one instruction, so use explicit PC computation.
51213. 6: toUTCString

51214. **comment:** note: ctx-wide temporary
label: code-design

51215. value popped by call

51216. Note: this may fail, caller should protect the call if necessary

51217. Leaves new timevalue on stack top and returns 1, which is correct * for part setters.

51218. stack[0] = compareFn * stack[1] = ToObject(this) * stack[2] = ToUint32(length)

51219. 4

51220. Boolean/any -> coerce boolean to number and try again. If boolean is * compared to a pointer, the final comparison after coercion now always * yields false (as pointer vs. number compares to false), but this is * not special cased.

51221. s <- 2 * b

51222. native functions: 149

51223. * Range parsing is done with a special lexer function which calls * us for every range parsed. This is different from how rest of * the parsing works, but avoids a heavy, arbitrary size intermediate * value type to hold the ranges. * * Another complication is the handling of character ranges when * case insensitive matching is used (see docs for discussion). * The range handler callback given to the lexer takes care of this * as well. * * Note that duplicate ranges are not eliminated when parsing character * classes, so that canonicalization of * * [0-9a-fA-Fx-{}] * * creates the result (note the duplicate ranges): * * [0-9A-FA-FX-Z-{}-{} * * where [x-{} is split as a result of canonicalization. The duplicate * ranges are not a semantics issue: they work correctly.

51224. Although we can allow non-BMP characters (they'll decode * back into surrogate pairs), we don't allow extended UTF-8 * characters; they would encode to URIs which won't decode * back because of strict UTF-8 checks in URI decoding. * (However, we could just as well allow them here.)

51225. exponent 1070

51226. no need to reset temps, as we're finished emitting code

51227. DUK_ERR_ERROR: no macros needed

51228. **comment:** XXX: assumed to fit for now
label: code-design

51229. [... fallback retval]

51230. **comment:** XXX: Migrate bufwriter and other read/write helpers to its own header?
label: code-design

51231. recursive call for a primitive value (guaranteed not to cause second * recursion).

51232. Getter might have arbitrary side effects, * so bail out.

51233. * The traceback format is pretty arcane in an attempt to keep it compact * and cheap to create. It may change arbitrarily from version to version. * It should be decoded/accessed through version specific accessors only. * * See doc/error-objects.rst.

51234. The internal _Target property is kept pointing to the original * enumeration target (the proxy object), so that the enumerator * 'next' operation can read property values if so requested. The * fact that the _Target is a proxy disables key existence check * during enumeration.

51235. Reject a proxy object as the handler because it would cause * potentially unbounded recursion. (ES6 has no such restriction)

51236. absolute req_digits; e.g. digits = 1 -> last digit is 0, * but add an extra digit for rounding.

51237. -> [... voidp voidp]

51238. convert duk_compiler_func into a function template, leaving the result * on top of stack.

51239. r <- r (updated above: r <- (remainder (* r B) s) * s <- s * m+ <- m+ (updated above: m+ <- (* m+ B) * m- <- m- (updated above: m- <- (* m- B) * B, low_ok, high_ok are fixed

51240. special helper for emitting u16 lists (used for character ranges for built-in char classes)

51241. [... re_obj input bc saved_buf]

51242. 'reg_varbind' is the operation result and can also * become the expression value for top level assignments * such as: "var x; x += y;".

51243. **comment:** * Three possible outcomes: * * A try or finally catcher is found => resume there. * (or) * * The error propagates to the bytecode executor entry * level (and we're in the entry thread) => rethrow * with a new longjmp(), after restoring the previous * catchpoint. * * The error is not caught in the current thread, so * the thread finishes with an error. This works like * a yielded error, except that the thread is finished * and can no longer be resumed. (There is always a * resumer in this case.) * * Note: until we hit the entry level, there can only be * Ecmascript activations.
label: code-design

51244. * Number-to-string and string-to-number conversions. * * Slow path number-to-string and string-to-number conversion is based on * a Dragon4 variant, with fast paths for small integers. Big integer * arithmetic is needed for guaranteeing that the conversion is correct * and uses a minimum number of digits. The big number arithmetic has a * fixed maximum size and does not require dynamic allocations. * * See: doc/number-conversion.rst.

51245. Parser separator indices.

51246. Regexp

51247. k > 0 -> k was too low, and cannot be too high

51248. 'func' in the algorithm

51249. **comment:** * Note: zero works as a "no update" marker because the new length * can never be zero after a new property is written. * * Note: must re-lookup because calls above (e.g. duk_alloc_entry_checked()) * may realloc and compact properties and hence change e_idx.
label: code-design

51250. Outside any activation -> look up from global.

51251. Decrease by 25% every round

51252. For manual testing only.

51253. **comment:** Execute the fast path in a protected call. If any error is thrown, * fall back to the slow path. This includes e.g. recursion limit * because the fast path has a smaller recursion limit (and simpler, * limited loop detection).
label: code-design

51254. * Main heap structure

51255. * Argument exotic [[Delete]] behavior (E5 Section 10.6) is * a post-check, keeping arguments internal 'map' in sync with * any successful deletes (note that property does not need to * exist for delete to 'succeed'). * * Delete key from 'map'. Since 'map' only contains array index * keys, we can use arr_idx for a fast skip.

51256. E5 Section 9.2

51257. Backtrack utf-8 input and return a (possibly canonicalized) input character.

51258. Cause an interrupt after executing one instruction.

51259. Compiler

51260. at least errobj must be on stack

51261. 16 bits

51262. * Default panic handler

51263. thr->heap->dbg_detaching may be != 0 if a debugger write outside * the message loop caused a transport error and detach1() to run.

51264. 'modSearch'

51265. **comment:** The condition could be more narrow and check for the * copy area only, but there's no need for fine grained * behavior when the underlying buffer is misconfigured.
label: code-design

51266. **comment:** XXX: for a memory-code tradeoff, remove 'func' and make it's access either a function * or a macro. This would make the activation 32 bytes long on 32-bit platforms again.
label: code-design

51267. Function name is an Identifier (not IdentifierName), but we get * the raw name (not recognizing keywords) here and perform the name * checks only after pass 1.

51268. does not fit into 16 bits

51269. ensures no overflow

51270. [... input buffer]

51271. execution resumes in bytecode executor

51272. Standard JSON omits functions

51273. 7 bits

51274. * Value stack resizing. ** This resizing happens above the current "top": the value stack can be * grown or shrunk, but the "top" is not affected. The value stack cannot * be resized to a size below the current "top". ** The low level reallocation primitive must carefully recompute all value * stack pointers, and must also work if ALL pointers are NULL. The resize * is quite tricky because the valstack realloc may cause a mark-and-sweep, * which may run finalizers. Running finalizers may resize the valstack * recursively (the same value stack we're working on). So, after realloc * returns, we know that the valstack "top" should still be the same (there * should not be live values above the "top"), but its underlying size and * pointer may have changed.

51275. * Objects have internal references. Must finalize through * the "refzero" work list.

51276. different thread

51277. number of values in current MPUTARR set

51278. steps 13-14

51279. Compute result length and validate argument buffers.

51280. DUK_BUILTINS_H_INCLUDED

51281. copy bytecode instructions one at a time

51282. **comment:** E5.1 Section 15.2.4.6, step 3.a, lookup proto once before compare. * Prototype loops should cause an error to be thrown.
label: code-design

51283. [... RegExp val] -> [... res]

51284. load factor max 75%

51285. [value offset noAssert], when ftype != DUK_FLD_VARINT

51286. * Resizing

51287. token closes expression, e.g. ')', ']'

51288. fileName

51289. * Get prototype object for an integer error code.

51290. '\xffNext'

51291. PC/line semantics here are: * - For callstack top we're conceptually between two * opcodes and current PC indicates next line to * execute, so report that (matches Status). * - For other activations we're conceptually still * executing the instruction at PC-1, so report that * (matches error stacktrace behavior). * - See: <https://github.com/svaaraal/duktape/issues/281>

51292. Don't allow negative zero as it will cause trouble with * LDINT+LDINTX. But positive zero is OK.

51293. borrow literal Infinity from builtin string

51294. **comment:** XXX: This now coerces an identifier into a GETVAR to a temp, which * causes an extra LDREG in call setup. It's sufficient to coerce to a * unary ivalue?
label: code-design

51295. prefer direct execution

51296. **comment:** Longjmp state is cleaned up by error handling
label: code-design

51297. **comment:** spaces[] must be static to allow initializer with old compilers like BCC
label: code-design

51298. * Object.prototype.hasOwnProperty() and Object.prototype.propertyIsEnumerable().

51299. Ordinary gap, undefined encodes to 'null' in * standard JSON (and no JX/JC support here now).

51300. fall-through jump to next code of next case (backpatched)

51301. already one-based

51302. **comment:** XXX: could use at least one fewer loop counters
label: code-design

51303. Wrap checked above.

51304. Create function 'data' buffer but don't attach it yet.

51305. * Function built-ins

51306. * No match, E5 Section 15.10.6.2, step 9.a.i - 9.a.ii apply, regardless * of 'global' flag of the RegExp. In particular, if lastIndex is invalid * initially, it is reset to zero.

51307. **comment:** hacky helper for String.prototype.split()
label: code-design

51308. [(builtin objects) name func]

51309. We request a tail call, but in some corner cases * call handling can decide that a tail call is * actually not possible. * See: test-bug-tailcall-preventyield-assert.c.

51310. toString() conditions

51311. Write bytecode executor's curr_pc back to topmost activation (if any).

51312. [] -> [val]

51313. **comment:** XXX: Extensibility check for target uses IsExtensible(). If we * implemented the isExtensible trap and didn't reject proxies as * proxy targets, it should be respected here.
label: code-design

51314. **comment:** Technically return value is not needed because INVLHS will * unconditionally throw a ReferenceError. Coercion is necessary * for proper semantics (consider ToNumber() called for an object). * Use DUK_EXTRAOP_UNP with a dummy register to get ToNumber().
label: code-design

51315. decl type

51316. -> [sep TObject(this) len sep str]

51317. Shared lenient buffer length clamping helper. Indices are treated as * element indices (though output values are byte offsets) which only * really matters for TypedArray views as other buffer object have a zero * shift. Negative indices are counted from end of input slice; crossed * indices are clamped to zero length; and final indices are clamped * against input slice. Used for e.g. ArrayBuffer slice().

51318. XXX: perhaps refactor this to allow caller to specify some parameters, or * at least a 'compact' flag which skips any spare or round-up .. useful for * emergency gc.

51319. [... enum_target res handler trap]

51320. XXX: Decrementing and restoring act->curr_pc works now, but if the * debugger message loop gains the ability to adjust the current PC * (e.g. a forced jump) restoring the PC here will break. Another * approach would be to use a state flag for the "decrement 1 from * topmost activation's PC" and take it into account whenever dealing * with PC values.

51321. Create a fresh object environment for the global scope. This is * needed so that the global scope points to the newly created RAM-based * global object.

51322. x <- (1<<y)

51323. * HASPROP: Ecmascript property existence check ("in" operator). ** Interestingly, the 'in' operator does not do any coercion of * the target object.

51324. Any 'res' will do.

51325. If neutered must return 0; length is zeroed during * neutering.

51326. XXX: assert idx_base

51327. **comment:** XXX: refactor into internal helper, duk_clone_hobject()
label: code-design

51328. * Get starting character offset for match, and initialize 'sp' based on it. ** Note: lastIndex is non-configurable so it must be present (we check the * internal class of the object above, so we know it is). User code can set * its value to an arbitrary (garbage) value though; E5 requires that lastIndex * be coerced to a number before using. The code below works even if the * property is missing: the value will then be coerced to zero. ** Note: lastIndex may be outside Uint32 range even after ToInteger() coercion. * For instance, ToInteger(+Infinity) = +Infinity. We track the match offset * as an integer, but pre-check it to be inside the 32-bit range before the loop. * If not, the check in E5 Section 15.10.6.2, step 9.a applies.

51329. avoid side effect issues

51330. * Setup call: target and 'this' binding. Three cases: * * 1. Identifier base (e.g. "foo()") * 2. Property base (e.g. "foo.bar()") * 3. Register base (e.g. "foo()"); i.e. when a return value is a function

51331. SHIFT EXPRESSION

51332. overflow, fall through

51333. restored if ecma-to-ecma setup fails

51334. Match labels starting from latest label because there can be duplicate empty * labels in the label set.
51335. If debugger is paused, garbage collection is disabled by default.
51336. DUK_ERR_REFERENCE_ERROR: no macros needed
51337. XXX: shared helper for duk_push_hobject_or_undefined()
51338. * Strings have no internal references but do have "weak" * references in the string cache. Also note that strings * are not on the heap_allocated list like other heap * elements.
51339. unconditionally
51340. **comment:** XXX: At the moment Duktape accesses internal keys like _Finalizer using a * normal property set/get which would allow a proxy handler to interfere with * such behavior and to get access to internal key strings. This is not a problem * as such because internal key strings can be created in other ways too (e.g. * through buffers). The best fix is to change Duktape internal lookups to * skip proxy behavior. Until that, internal property accesses bypass the * proxy and are applied to the target (as if the handler did not exist). * This has some side effects, see test-bi-proxy-internal-keys.js.
label: code-design
51341. Get local time offset (in seconds) for a certain (UTC) instant 'd'.
51342. **comment:** Can be called multiple times with no harm. Mark the transport * bad (dbg_read_cb == NULL) and clear state except for the detached * callback and the udata field. The detached callback is delayed * to the message loop so that it can be called between messages; * this avoids corner cases related to immediate debugger reattach * inside the detached callback.
label: code-design
51343. resume
51344. * Parse a function body or a function-like expression, depending * on flags.
51345. actually used, non-NULL entries
51346. Fast path check.
51347. length check
51348. Do decrefs only with safe pointers to avoid side effects * disturbing e_idx.
51349. '{"_ninf":true}'
51350. shared checks for all descriptor types
51351. There's no need to check for buffer validity status for the * target here: the property access code will do that for each * element. Moreover, if we did check the validity here, side * effects from reading the source argument might invalidate * the results anyway.
51352. * Process space (3rd argument to JSON.stringify)
51353. Difference to non-strict/strict comparison is that NaNs compare * equal and signed zero signs matter.
51354. DUK_USE_PANIC_HANDLER
51355. avoid array abandoning which interns strings
51356. traits are separate; in particular, arguments not an array
51357. duk_get_min_grow_a() is always >= 1
51358. 9: getUTCFullYear
51359. [key getter this key] -> [key retval]
51360. Source end clamped silently to available length.
51361. First character has already been eaten and checked by the caller. * We can scan until a NUL in stridx string because no built-in strings * have internal NULs.
51362. * Proxy built-in (ES6)
51363. Native function, no relevant lineNumber.
51364. * Limited functionality bigint implementation. * * Restricted to non-negative numbers with less than 32 * DUK__BI_MAX_PARTS bits, * with the caller responsible for ensuring this is never exceeded. No memory * allocation (except stack) is needed for bigint computation. Operations * have been tailored for number conversion needs. * * Argument order is "assignment order", i.e. target first, then arguments: * x <- y * z --> duk__bi_mul(x, y, z);
51365. * InitJS code - Ecmascript code evaluated from a built-in source * which provides e.g. backward compatibility. User can also provide * JS code to be evaluated at startup.
51366. 'res' must be a plain ivalue, and not register-bound variable.
51367. function helpers
51368. 'Null'
51369. internal properties
51370. custom
51371. * XXX: shrink array allocation or entries compaction here?
51372. ToInteger(lastIndex)
51373. * Encoding/decoding helpers
51374. default: NaN
51375. covers -Infinity
51376. Error instance, use augmented error data directly
51377. t1 = (+ r m+)
51378. [... func arg1 ... argN]
51379. **comment:** XXX: unnecessary string coercion for array indices, * intentional to keep small; some overlap with string * handling.
label: code-design
51380. Some contexts don't allow fractions at all; this can't be a * post-check because the state ('f' and expt) would be incorrect.
51381. variable value (function, we hope, not checked here)
51382. return 'res_obj'
51383. E5 Section 10.4.2
51384. DUK_FORWDECL_H_INCLUDED
51385. avoid unary minus on unsigned
51386. Just flip the single bit.
51387. unicode code points, window[0] is always next, points to 'buffer'
51388. probe
51389. may be set to 0 by duk_debugger_attach() inside callback
51390. Easy to get wrong, so assert for it.
51391. mask out flags not actually stored
51392. 'with' target must be created first, in case we run out of memory
51393. **comment:** Slow writing of value using either 64-bit arithmetic * or IEEE doubles if 64-bit types not available. There's * no special sign handling when writing varints.
label: code-design
51394. Important main primitive.
51395. no throw
51396. [... lval rval(func)]
51397. **comment:** XXX: 'this' will be ToObject() coerced twice, which is incorrect * but should have no visible side effects.
label: code-design
51398. **comment:** XXX: The duk_to_number() cast followed by integer coercion * is platform specific so NaN, +/- Infinity, and out-of-bounds * values result in platform specific output now. * See: test-bi-nodejs-buffer-proto-varint-special.js
label: code-design
51399. 'lastIndex'
51400. 0xd0-0xdf
51401. * Format tag parsing. Since we don't understand all the * possible format tags allowed, we just scan for a terminating * specifier and keep track of relevant modifiers that we do * understand. See man 3 printf.
51402. pop varname

51403. **comment:** XXX: this should probably just operate on the stack top, because it * needs to push stuff on the stack anyway...

label: code-design

51404. 'advtok' indicates how much to advance and which token id to assign * at the end. This shared functionality minimizes code size. All * code paths are required to set 'advtok' to some value, so no default * init value is used. Code paths calling DUK_ERROR() never return so * they don't need to set advtok.

51405. debugger

51406. * Automatically generated by extract_chars.py, do not edit!

51407. * Copy keys and values in the entry part (compacting them at the same time).

51408. expensive init, just want to fill window

51409. 'toLogString'

51410. split2: prefer jump execution (not direct)

51411. Zero special case: fake requested number of zero digits; ensure * no sign bit is printed. Relative and absolute fixed format * require separate handling.

51412. skip line info

51413. heap destruction ongoing, finalizer rescue no longer possible

51414. E5 Section 9.1

51415. value stack indices for tracking objects

51416. two encoding attempts suffices

51417. DUK_ISPEC_XXX

51418. **comment:** XXX: append primitive

label: code-design

51419. current lexical environment (may be NULL if delayed)

51420. explicit NULL inits

51421. [... closure template env]

51422. use a temp: decref only when valstack reachable values are correct

51423. Helper for string conversion calls: check 'this' binding, get the * internal time value, and format date and/or time in a few formats. * Return value allows tail calls.

51424. 0 or -1

51425. **comment:** * Allocate heap struct * * Use a raw call, all macros expect the heap to be initialized

label: code-design

51426. Object.defineProperty() equivalent C binding.

51427. skip jump slots

51428. We can directly access value stack here.

51429. DUK_TOK_TYPEOF

51430. mark as complex

51431. XXX: handling of timestamps outside Windows supported range. * How does Windows deal with dates before 1600? Does windows * support all Ecmascript years (like -200000 and +200000)? * Should equivalent year mapping be used here too? If so, use * a shared helper (currently integrated into timeval-to-parts).

51432. **comment:** m+ != m- (very rarely)

label: code-design

51433. * Duktape debugger

51434. Ecmascript activation + Duktape.Thread.resume() activation

51435. \"

51436. [... str]

51437. -> [... value]

51438. Use temporaries and update lex_ctx only when finished.

51439. mark-and-sweep not running -> must be empty

51440. **comment:** * Normal error * * Normal error is thrown with a longjmp() through the current setjmp() * catchpoint record in the duk_heap. The 'curr_thread' of the duk_heap * identifies the throwing thread. * * Error formatting is usually unnecessary. The error macros provide a * zero argument version (no formatting) and separate macros for small * argument counts. Variadic macros are not used to avoid portability * issues and avoid the need for stash-based workarounds when they're not * available. Vararg calls are avoided for non-formatted error calls * because vararg call sites are larger than normal, and there are a lot * of call sites with no formatting. * * Note that special formatting provided by debug macros is NOT available. * * The _RAW variants allow the caller to specify file and line. This makes * it easier to write checked calls which want to use the call site of the * checked function, not the error macro call inside the checked function.

label: code-design

51441. **comment:** Fast canonicalization lookup at the cost of 128kB footprint.

label: code-design

51442. DUK_TOK_EXPORT

51443. array of active label names

51444. * Helper for C function call negative return values.

51445. entry_top + 0

51446. * Thread switching * * To switch heap->curr_thread, use the macro below so that interrupt counters * get updated correctly. The macro allows a NULL target thread because that * happens e.g. in call handling.

51447. entry allocation updates hash part and increases the key * refcount; may need a props allocation resize but doesn't * 'recheck' the valstack.

51448. Not necessary to unwind catchstack: break/continue * handling will do it. The finally flag of 'cat' is * no longer set. The catch flag may be set, but it's * not checked by break/continue handling.

51449. Shared helper.

51450. x1 must be a string

51451. [... value? getter? setter?]

51452. **comment:** XXX: the insert here is a bit expensive if there are a lot of items. * It could also be special cased in the outermost for loop quite easily * (as the element is dup()'d anyway).

label: code-design

51453. Gather in big endian

51454. **comment:** SCANBUILD: warning about 'thr' potentially being NULL here, * warning is incorrect because thr != NULL always here.

label: code-design

51455. format is bit packed

51456. See test-bug-netbsd-math-pow.js: NetBSD 6.0 on x86 (at least) does not * correctly handle some cases where x=+-0. Specific fixes to these * here.

51457. parse args starting from "next temp", reg_target + 1

51458. Note: multiple threads may be simultaneously in the RUNNING * state, but not in the same "resume chain".

51459. * Named function expression, name needs to be bound * in an intermediate environment record. The "outer" * lexical/variable environment will thus be: * * a) { funcname: <func>, __prototype: outer_lex_env } * b) { funcname: <func>, __prototype: <globalenv> } (if outer_lex_env missing)

51460. * x is finite and non-zero * * -1.6 -> floor(-1.1) -> -2 * -1.5 -> floor(-1.0) -> -1 (towards +Inf) * -1.4 -> floor(-0.9) -> -1 * -0.5 -> -0.0 (special case) * -0.1 -> -0.0 (special case) * +0.1 -> +0.0 (special case) * +0.5 -> floor(+1.0) -> 1 (towards +Inf) * +1.4 -> floor(+1.9) -> 1 * +1.5 -> floor(+2.0) -> 2 (towards +Inf) * +1.6 -> floor(+2.1) -> 2

51461. * Context init

51462. **comment:** XXX: rework

label: code-design

51463. Call stack, [0,callstack_top[is GC reachable.

51464. DUK_NUMCONV_H_INCLUDED

51465. is a function declaration (as opposed to function expression)

51466. Precomputed pointers when using 16-bit heap pointer packing.

51467. The E5.1 Section 15.4.4 algorithm doesn't set the length explicitly * in the end, but because we're operating with an internal value which * is known to be an array, this should be equivalent.

51468. XXX: Allow customizing the pause and notify behavior at runtime * using debugger runtime flags. For now the behavior is fixed using * config options.
51469. = IsAccessorDescriptor(Desc)
51470. 36: setFullYear
51471. Object is currently being finalized.
51472. **comment:** not covered, return all zeroes
label: test
51473. * Unicode range matcher * * Matches a codepoint against a packed bitstream of character ranges. * Used for slow path Unicode matching.
51474. DUK_HTHREAD_H_INCLUDED
51475. DUK_USE_ES6_PROXY
51476. Located in duk_heap.hdr h_extra16. Subclasses of duk_hobject (like * duk_hcompiledfunction) are not free to use h_extra16 for this reason.
51477. When looking for .fileName/.lineNumber, blame compilation * or C call site unless flagged not to do so.
51478. Pretend like we got EOM
51479. [regexp source bytecode]
51480. catch depth at point of definition
51481. -> [... cons target]
51482. Grow entry part allocation for one additional entry.
51483. **comment:** * Calls. * * Protected variants should avoid ever throwing an error.
label: code-design
51484. **comment:** XXX: better multiline
label: code-design
51485. object literal key tracking flags
51486. curr is accessor -> cannot be in array part
51487. stmt has explicit/implicit semicolon terminator
51488. lexing
51489. [... result/error]
51490. catches EOF (NUL) and initial comma
51491. **comment:** If no explicit message given, put error code into message field * (as a number). This is not fully in keeping with the EcmaScript * error model because messages are supposed to be strings (Error * constructors use ToString() on their argument). However, it's * probably more useful than having a separate 'code' property.
label: code-design
51492. The #ifdef clutter here needs to handle the three cases: * (1) JX+JC, (2) JX only, (3) JC only.
51493. currently running thread
51494. temproots
51495. Tail call check: if last opcode emitted was CALL(I), and * the context allows it, change the CALL(I) to a tail call. * This doesn't guarantee that a tail call will be allowed at * runtime, so the RETURN must still be emitted. (Duktape * 0.10.0 avoided this and simulated a RETURN if a tail call * couldn't be used at runtime; but this didn't work * correctly with a thread yield/resume, see * test-bug-tailcall-thread-yield-resume.js for discussion.) * * In addition to the last opcode being CALL, we also need to * be sure that 'rc_val' is the result register of the CALL(I). * For instance, for the expression 'return 0, (function () { return 1; })', 2' the last opcode emitted is CALL (no * bytecode is emitted for '2') but 'rc_val' indicates * constant '2'. Similarly if '2' is replaced by a register * bound variable, no opcodes are emitted but tail call would * be incorrect. * * This is tricky and easy to get wrong. It would be best to * track enough expression metadata to check that 'rc_val' came * from that last CALL instruction. We don't have that metadata * now, so we check that 'rc_val' is a temporary register result * (not a constant or a register bound variable). There should * be no way currently for 'rc_val' to be a temporary for an * expression following the CALL instruction without emitting * some opcodes following the CALL. This proxy check is used * below. * * See: test-bug-comma-expr-gh131.js. * * The non-standard 'caller' property disables tail calls * because they pose some special cases which haven't been * fixed yet.
51496. * Helper: handle Array object 'length' write which automatically * deletes properties, see E5 Section 15.4.5.1, step 3. This is * quite tricky to get right. * * Used by duk_hobject_putprop().
51497. vararg
51498. Use 'res' as the expression value (it's side effect * free and may be a plain value, a register, or a * constant) and write it to the LHS binding too.
51499. Allow NULL 'msg'
51500. borrowed reference
51501. [... s1 s2] -> [... s1+s2]
51502. DUK_TOK_DIV
51503. **comment:** XXX: lithuanian not implemented
label: requirement
51504. Explicit zero size check to avoid NULL 'tv1'.
51505. * Get holder object
51506. XXX: can shift() / unshift() use the same helper? * shift() is (close to?) <-> splice(0, 1) * unshift is (close to?) <-> splice(0, 0, [items])?
51507. lead zero + 'digits' fractions + 1 for rounding
51508. stack discipline consistency check
51509. * Update the interrupt counter
51510. these non-standard properties are copied for convenience
51511. still reachable
51512. * Forward declarations for all Duktape structures.
51513. [... obj key]
51514. **comment:** * Object.preventExtensions() and Object.isExtensible() (E5 Sections 15.2.3.10, 15.2.3.13) * * Not needed, implemented by macros DUK_HOBJECT_{HAS,CLEAR,SET}_EXTENSIBLE * and the Object built-in bindings.
label: code-design
51515. * Macros for property handling
51516. val2 = min count, val1 = max count
51517. find and remove string from stringtable; caller must free the string itself
51518. Inner functions recursively.
51519. == DUK__MAX_TEMPS is OK
51520. Don't free h->resumer because it exists in the heap. * Callstack entries also contain function pointers which * are not freed for the same reason.
51521. use bigint area as a temp
51522. 'n'
51523. * Allocate initial stacks for a thread. Note that 'thr' must be reachable * as a garbage collection may be triggered by the allocation attempts. * Returns zero (without leaking memory) if init fails.
51524. step 4.b
51525. (Number.MAX_VALUE).toString(2).length == 1024, + spare
51526. * Check whether already declared. * * We need to check whether the binding exists in the environment * without walking its parents. However, we still need to check * register-bound identifiers and the prototype chain of an object * environment target object.
51527. Catch stack. [0,catchstack_top[is GC reachable.
51528. DUK_USE_DATE_TZOWINDOWS
51529. -> [... res_obj]
51530. * E5 Section 15.10.2.6. The previous and current character * should -not- be canonicalized as they are now. However, * canonicalization does not affect the result of IsWordChar() * (which depends on Unicode characters never canonicalizing * into ASCII characters) so this does not matter.
51531. flags is unsigned
51532. Already declared but no initializer value * (e.g. 'var xyz;'), no-op.
51533. 0

51534. **comment:** XXX: better place for this
label: code-design

51535. Zero encoded pointer is required to match NULL

51536. Given a (year, month, day-within-month) triple, compute day number. * The input triple is un-normalized and may contain non-finite values.

51537. upper limit, assuming no whitespace etc

51538. Copy slice, respecting underlying buffer limits; remainder * is left as zero.

51539. **comment:** not necessary to init, disabled for faster parsing
label: code-design

51540. We already ate 'x', so backup one byte.

51541. maximum token count before error (sanity backstop)

51542. * For ASCII strings, the answer is simple.

51543. 'super'

51544. **comment:** XXX: This won't be shown in practice now * because this code is not run when builtins * are in ROM.
label: code-design

51545. 'JSON'

51546. combine left->x1 and res->x1 (right->x1, really) -> (left->x1 OP res->x1)

51547. We can't reliably pop anything here because the stack input * shape is incorrect. So we throw an error; if the caller has * no catch point for this, a fatal error will occur. Another * alternative would be to just return an error. But then the * stack would be in an unknown state which might cause some * very hard to diagnose problems later on. Also note that even * if we did not throw an error here, the underlying call handler * might STILL throw an out-of-memory error or some other internal * fatal error.

51548. marker: copy t if not changed

51549. thread has terminated

51550. * Context management

51551. Return codes for protected calls (duk_safe_call(), duk_pcall())

51552. RangeError

51553. set/clear writable

51554. DUK_USE_64BIT_OPS

51555. Ecmascript date range is 100 million days from Epoch: * > 100e6 * 24 * 60 * 60 * 1000 // 100M days in millisecs * 8640000000000000 * (= 8.64e15)

51556. * Coercion operations: in-place coercion, return coerced value where * applicable. If index is invalid, throw error. Some coercions may * throw an expected error (e.g. from a toString() or valueOf() call) * or an internal error (e.g. from out of memory).

51557. packed tvl

51558. (internal) catch compilation errors

51559. * END PUBLIC API

51560. E == 0x7ff, topmost four bits of F != 0 => assume NaN

51561. APIError

51562. * BEGIN PUBLIC API

51563. timeval breakdown: internal time value NaN -> RangeError (toISOString)

51564. internal flag: external buffer

51565. enumerate internal properties (regardless of enumerability)

51566. Ecmascript E5 specification error codes

51567. * Debugger (debug protocol)

51568. lstring

51569. timeval breakdown: internal time value NaN -> zero

51570. include time part in string conversion result

51571. prefer number

51572. Note: parentheses are required so that the comma expression works in assignments.

51573. no error if file does not exist

51574. internal flag value: throw if mask doesn't match

51575. set getter (given on value stack)

51576. * Memory management * * Raw functions have no side effects (cannot trigger GC).

51577. Coercion hints

51578. is_copy

51579. * Variable access

51580. weekday: 0 to 6, 0=sunday, 1=monday, etc

51581. DUK_DBUNION_H_INCLUDED

51582. lightweight function pointer

51583. * String manipulation

51584. getter: subtract 1900 from year when getting year part

51585. Value mask types, used by e.g. duk_get_type_mask()

51586. Ecmascript undefined

51587. file

51588. Log levels

51589. Ecmascript boolean: 0 or 1

51590. * Public API specific typedefs * * Many types are wrapped by Duktape for portability to rare platforms * where e.g. 'int' is a 16-bit type. See practical typing discussion * in Duktape web documentation.

51591. flags

51592. Assertion Error

51593. only enumerate array indices

51594. * Stack management

51595. Flags for duk_def_prop() and its variants

51596. NOTE: when writing a Date provider you only need a few specific * flags from here, the rest are internal. Avoid using anything you * don't need.

51597. * Date provider related constants * * NOTE: These are "semi public" - you should only use these if you write * your own platform specific Date provider, see doc/datetime.rst.

51598. internal: request fixed buffer result

51599. Number of value stack entries (in addition to actual call arguments) * guaranteed to be allocated on entry to a Duktape/C function.

51600. * Indexes of various types with respect to big endian (logical) layout

51601. no error (e.g. from duk_get_error_code())

51602. LICENSE.txt

51603. SyntaxError

51604. * Duktape/C function magic value

51605. internal: request dynamic buffer result

51606. * Avoid C++ name mangling

51607. **comment:** XXX: replace with TypeError?
label: code-design

51608. **comment:** XXX: to be removed?
label: code-design

51609. * ===== * Duktape license * ===== * * (<http://opensource.org/licenses/MIT>) * * Copyright (c) 2013-2017 by Duktape authors (see AUTHORS.rst) * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the

"Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

51610. Git commit, describe, and branch for Duktape build. Useful for * non-official snapshot builds so that application code can easily log * which Duktape snapshot was used. Not available in the Ecmascript * environment.

51611. DUK_API_PUBLIC_H_INCLUDED

51612. Compilation flags for duk_compile() and duk_eval()

51613. * Duktape public API for Duktape 1.8.0. * * See the API reference for documentation on call semantics. * The exposed API is inside the DUK_API_PUBLIC_H_INCLUDED * include guard. Other parts of the header are Duktape * internal and related to platform/compiler/feature detection. * * Git commit 0a70d7e4c5227c84e3fed5209828973117d02849 (v1.8.0). * Git branch v1.8-maintenance. * * See Duktape AUTHORS.rst and LICENSE.txt for copyright and * licensing information.

51614. timeval breakdown: convert month and day-of-month parts to one-based (default is zero-based)

51615. Duktape version, (major * 10000) + (minor * 100) + patch. Allows C code * to #ifdef against Duktape API version. The same value is also available * to Ecmascript code in Duktape.version. Unofficial development snapshots * have 99 for patch level (e.g. 0.10.99 would be a development version * after 0.10.0 but before the next official release).

51616. * Buffer

51617. set enumerable (effective if DUK_DEFPROP_HAVE_ENUMERABLE set)

51618. set configurable (effective if DUK_DEFPROP_HAVE_CONFIGURABLE set)

51619. don't walk prototype chain, only check own properties

51620. UnimplementedError

51621. (internal) omit eval result

51622. **comment:** * If no variadic macros, __FILE__ and __LINE__ are passed through globals * which is ugly and not thread safe.
label: requirement

51623. last value is func pointer, arguments follow in parens

51624. set/clear configurable

51625. Return codes for C functions (shortcut for throwing an error)

51626. * Pop operations

51627. * Require operations: no coercion, throw error if index or type * is incorrect. No defaulting.

51628. * Bytecode load/dump

51629. compile function code (instead of global code)

51630. enumerate a proxy object itself without invoking proxy behavior

51631. **comment:** XXX: native 64-bit byteswaps when available
label: code-design

51632. **comment:** XXX: These calls are incomplete and not usable now. They are not (yet) * part of the public API.
label: code-design

51633. Error

51634. DUK_USE_FILE_IO

51635. ReferenceError

51636. timeval breakdown: replace year with equivalent year in the [1971,2037] range for DST calculations

51637. * Ecmascript operators

51638. args

51639. * Some defines forwarded from feature detection

51640. (internal) take strlen() of src_buffer (avoids double evaluation in macro)

51641. set/clear enumerable

51642. internal use

51643. * Error handling

51644. EvalError

51645. * Object prototype

51646. setter: perform 2-digit year fixup (00...99 -> 1900...1999)

51647. nop: no need to normalize

51648. * Compilation and evaluation

51649. or is a normalized NaN

51650. AllocError

51651. month: 0 to 11

51652. **comment:** XXX: Enough space to hold internal suspend/resume structure. * This is rather awkward and to be fixed when the internal * structure is visible for the public API header.
label: code-design

51653. internal flag: create backing ArrayBuffer; keep in one byte

51654. additional values begin at bit 12

51655. * Constants

51656. **comment:** XXX: replace with plain Error?
label: code-design

51657. * Object finalizer

51658. Reverse operation is the same.

51659. duk_context is now defined in duk_config.h because it may also be * referenced there by prototypes.

51660. * C++ name mangling

51661. UncaughtError

51662. Ecmascript year range: * > new Date(100e6 * 24 * 3600e3).toISOString() * '+275760-09-13T00:00:00.000Z' * > new Date(-100e6 * 24 * 3600e3).toISOString() * '-271821-04-20T00:00:00.000Z'

51663. create a new global environment

51664. * Global object

51665. DUKTAPE_H_INCLUDED

51666. E == 0x7ff, F != 0 => NaN

51667. DUK_USE_PACKED_TVAL

51668. * Logging

51669. safe variants of a few coercion operations

51670. Indicates that a native function does not have a fixed number of args, * and the argument stack should not be capped/extended at all.

51671. * Get operations: no coercion, returns default value for invalid * indices and invalid value types. * * duk_get_undefined() and duk_get_null() would be pointless and * are not included.

51672. **comment:** use locale specific formatting if available
label: code-design

51673. External duk_config.h provides platform/compiler/OS dependent * typedefs and macros, and DUK_USE_xxx config options so that * the rest of Duktape doesn't need to do any feature detection.

51674. internal: don't care about fixed/dynamic nature

51675. * Property access ** The basic function assumes key is on stack. The _string variant takes * a C string as a property name, while the _index variant takes an array * index as a property name (e.g. 123 is equivalent to the key "123").

51676. * Debugging

51677. 64-bit byteswap, same operation independent of target endianness.

51678. * Type checks ** duk_is_none(), which would indicate whether index is outside of stack, * is not needed; duk_is_valid_index() gives the same information.

51679. sort array indices, use with DUK_ENUM_ARRAY_INDICES_ONLY

51680. force change if possible, may still fail for e.g. virtual properties

51681. Millisecond count constants.

51682. used by packed duk_tval, assumes sizeof(void *) == 4

51683. set value (given on value stack)

51684. Used to represent invalid index; if caller uses this without checking, * this index will map to a non-existent stack entry. Also used in some * API calls as a marker to denote "no value".

51685. * Module helpers: put multiple function or constant properties

51686. raw void pointer

51687. Flags for duk_push_thread_raw()

51688. * ===== * Duktape authors * ===== * Copyright * ===== * * Duktape copyrights are held by its authors. Each author has a copyright * to their contribution, and agrees to irrevocably license the contribution * under the Duktape ``LICENSE.txt``. * * Authors * ===== * * Please include an e-mail address, a link to your GitHub profile, or something * similar to allow your contribution to be identified accurately. * * The following people have contributed code, website contents, or Wiki contents, * and agreed to irrevocably license their contributions under the Duktape * ``LICENSE.txt`` (in order of appearance): * * * Sami Vaarala <sami.vaarala@iki.fi> * * Niki Dobrev * * Andreas \u00d6man <andreas@lonelycoder.com> * * L\u00e1szl\u00f3 Lang\u00f3 <llango.u-szeged@partner.samsung.com> * * Legimet <legimet.calc@gmail.com> * * Karl Skomski <karl@skomski.com> * * Bruce Pascoe <fatcerberus1@gmail.com> * * Ren\u00e9 Hollander <rene@rene8888.at> * * Julien Hamaide (<https://github.com/crazyjul>) * * Sebastian G\u00f6ttfert (<https://github.com/jaseg>) * * Other contributions * ===== * * The following people have contributed something other than code (e.g. reported * bugs, provided ideas, etc; roughly in order of appearance): * * * Greg Burns * * Anthony Rabine * * Carlos Costa * * Aur\u00e9lien Bouilland * * Preet Desai (Pris Matic) * * judofyr (<http://www.reddit.com/user/judofyr>) * * Jason Woofenden * * Micha\u0142 Przyby\u015b * * Anthony Howe * * Conrad Pankoff * * Jim Schimpf * * Rajaran Gaunker (<https://github.com/zimbabao>) * * Andreas \u00d6man * * Doug Sanden * * Josh Engebreton (<https://github.com/JoshEngebreton>) * * Remo Eichenberger (<https://github.com/remoe>) * * Mamod Mehyar (<https://github.com/mamod>) * * David Demelier (<https://github.com/markand>) * * Tim Caswell (<https://github.com/creationix>) * * Mitchell Blank Jr (<https://github.com/mitchblank>) * * <https://github.com/yushli> * Seo Sanghyeon (<https://github.com/sanxiyn>) * * Han ChoongWoo (<https://github.com/tunz>) * * Joshua Peek (<https://github.com/josh>) * * Bruce E. Pascoe (<https://github.com/fatcerberus>) * * <https://github.com/Kelledin> * * <https://github.com/sstruchtrup> * * Michael Drake (<https://github.com/tlsa>) * * <https://github.com/chris-y> * * Laurent Zubiaur (<https://github.com/lzubiaur>) * * Ole Andr\u00e9 Vadla Ravn\u00e5s (<https://github.com/oleavr>) * * If you are accidentally missing from this list, send me an e-mail * (`sami.vaarala@iki.fi`) and I'll fix the omission.

51689. InternalError

51690. Value types, used by e.g. duk_get_type()

51691. **comment:** Duktape specific error codes (must be 8 bits at most, see duk_error.h)

label: code-design

51692. no value, e.g. invalid index

51693. * Stack manipulation (other than push/pop)

51694. Byteswap an IEEE double in the duk_double_union from host to network * order. For a big endian target this is a no-op.

51695. prefer string

51696. external use

51697. Flags for duk_push_string_file_raw()

51698. Duktape debug protocol version used by this build.

51699. **comment:** DUK_COMPILE_xxx bits 0-2 are reserved for an internal 'nargs' argument * (the nargs value passed is direct stack arguments + 1 to account for an * internal extra argument).

label: code-design

51700. fixed or dynamic, garbage collected byte buffer

51701. * Thread management

51702. * Double NaN manipulation macros related to NaN normalization needed when * using the packed duk_tval representation. NaN normalization is necessary * to keep double values compatible with the duk_tval format. * * When packed duk_tval is used, the NaN space is used to store pointers * and other tagged values in addition to NaNs. Actual NaNs are normalized * to a specific quiet NaN. The macros below are used by the implementation * to check and normalize NaN values when they might be created. The macros * are essentially NOPs when the non-packed duk_tval representation is used. * * A FULL check is exact and checks all bits. A NOTFULL check is used by * the packed duk_tval and works correctly for all NaNs except those that * begin with 0x7ff0. Since the 'normalized NaN' values used with packed * duk_tval begin with 0x7ff8, the partial check is reliable when packed * duk_tval is used. The 0x7ff8 prefix means the normalized NaN will be a * quiet NaN regardless of its remaining lower bits. * * The ME variant below is specifically for ARM byte order, which has the * feature that while doubles have a mixed byte order (32107654), unsigned * long long values has a little endian byte order (76543210). When writing * a logical double value through a ULL pointer, the 32-bit words need to be * swapped; hence the #ifdefs below for ULL writes with DUK_USE_DOUBLE_ME. * This is not full ARM support but suffices for some environments.

51703. either not a NaN

51704. convert time value to local time

51705. not directly applicable, byte order differs from a double

51706. end 'extern "C"' wrapper

51707. * Object operations

51708. set writable (effective if DUK_DEFPROP_HAVE_WRITABLE set)

51709. UnsupportedError

51710. **comment:** Internal API call flags, used for various functions in this file. * Certain flags are used by only certain functions, but since the flags * don't overlap, a single flags value can be passed around to multiple * functions. * * The unused top bits of the flags field are also used to pass values * to helpers (duk_get_part_helper() and duk_set_part_helper()). * * (Must be in-sync with genbuiltins.py)

label: code-design

51711. There are currently no native functions to yield/resume, due to the internal * limitations on coroutine handling. These will be added later.

51712. Concrete macros for NaN handling used by the implementation internals. * Chosen so that they match the duk_tval representation: with a packed * duk_tval, ensure NaNs are properly normalized; with a non-packed duk_tval * these are essentially NOPs.

51713. (internal) no source string on stack

51714. * Function (method) calls

51715. plain

51716. ECMAScript string: CESU-8 / extended UTF-8 encoded

51717. internal flag: don't zero allocated buffer

51718. use strict (outer) context for global, eval, or function code

51719. in non-packed representation we don't care about which NaN is used

51720. * Union for accessing double parts, also serves as packed duk_tval

51721. include date part in string conversion result

51722. internal flag: dynamic buffer

51723. ECMAScript null

51724. enumerate non-enumerable properties in addition to enumerable

51725. **comment:** XXX: replace with RangeError?

label: code-design

51726. * Push operations * * Push functions return the absolute (relative to bottom of frame) * position of the pushed value for convenience. * * Note: duk_dup() is technically a push.

51727. string
51728. * ROM pointer compression
51729. * Union to access IEEE double memory representation, indexes for double * memory representation, and some macros for double manipulation. ** Also used by packed duk_tval. Use a union for bit manipulation to minimize aliasing issues in practice. The C99 standard does not guarantee that this should work, but it's a very widely supported * practice for low level manipulation. ** IEEE double format summary: ** seeeeeeee eeeeffff ffffffff ffffffff ffffffff ffffffff * A B C D E F G H *'s sign bit * eee... exponent field * fff... fraction ** See http://en.wikipedia.org/wiki/Double_precision_floating-point_format. ** NaNs are represented as exponent 0x7ff and mantissa != 0. The NaN is a * signaling NaN when the highest bit of the mantissa is zero, and a quiet * NaN when the highest bit is set. ** At least three memory layouts are relevant here: ** A B C D E F G H Big endian (e.g. 68k) DUK_USE_DOUBLE_BE * H G F E D C B A Little endian (e.g. x86) DUK_USE_DOUBLE_LE * D C B A H G F E Mixed/cross endian (e.g. ARM) DUK_USE_DOUBLE_ME ** ARM is a special case: ARM double values are in mixed/cross endian * format while ARM duk_uint64_t values are in standard little endian * format (H G F E D C B A). When a double is read as a duk_uint64_t * from memory, the register will contain the (logical) value * E F G H A B C D. This requires some special handling below. ** Indexes of various types (8-bit, 16-bit, 32-bit) in memory relative to * the logical (big endian) order: ** byte order duk_uint8_t duk_uint16_t duk_uint32_t * BE 01234567 0123 01 * LE 76543210 3210 10 * ME (ARM) 32107654 1032 01 ** Some processors may alter NaN values in a floating point load+store. ** For instance, on X86 a FLD + FSTP may convert a signaling NaN to a * quiet one. This is catastrophic when NaN space is used in packed * duk_tval values. See: misc/clang_aliasing.c.

51730. * Helper macros for reading/writing memory representation parts, used * by duk_numconv.c and duk_tval.h.
51731. string conversion: use 'T' instead of '' as a separator
51732. Ecmascript number: double
51733. TypeError
51734. Support array for ROM pointer compression. Only declared when ROM * pointer compression is active.
51735. **comment:** Although extra/top could be an unsigned type here, using a signed type * makes the API more robust to calling code calculation errors or corner * cases (where caller might occasionally come up with negative values). * Negative values are treated as zero, which is better than casting them * to a large unsigned number. (This principle is used elsewhere in the * API too.)
label: code-design
51736. * Misc conversion
51737. Ecmascript object: includes objects, arrays, functions, threads
51738. URIError
51739. all doubles are considered normalized
51740. AUTHORS.rst
51741. day within month: 0 to 30
51742. setter: call is a time setter (affects hour, min, sec, ms); otherwise date setter (affects year, month, day-in-month)
51743. (internal) no filename on stack
51744. compile eval code (instead of global code)
51745. One problem with this macro is that expressions like the following fail * to compile: "(void) duk_error(...)". But because duk_error() is noreturn, * they make little sense anyway.
51746. set setter (given on value stack)
51747. prefer number, unless input is a Date, in which * case prefer string (E5 Section 8.12.8)
51748. Part indices for internal breakdowns. Part order from DUK_DATE_IDX_YEAR * to DUK_DATE_IDX_MILLISECOND matches argument ordering of Ecmascript API * calls (like Date constructor call). Some functions in duk_bi_date.c * depend on the specific ordering, so change with care. 16 bits are not * enough for all parts (year, specifically). * * (Must be in-sync with genbuiltins.py.)
51749. E == 0x7ff, F == 8 => normalized NaN
51750. Enumeration flags for duk_enum()
51751. year
51752. * Other state related functions
51753. user-id of owner
51754. time of last data modification
51755. file size, in bytes
51756. device type, for special file inode
51757. ** Factory of directory iterators
51758. ** Get symbolic link information using lstat.
51759. blocks allocated for file
51760. **
51761. push member value and return
51762. next entry
51763. lkmode valid values are: LK_LOCK Locks the specified bytes. If the bytes cannot be locked, the program immediately tries again after 1 second. If, after 10 attempts, the bytes cannot be locked, the constant returns an error. LK_NBLCK Locks the specified bytes. If the bytes cannot be locked, the constant returns an error. LK_NBRLCK Same as _LK_NBLCK. LK_RLCK Same as _LK_LOCK. LK_UNLCK Unlocks the specified bytes, which must have been previously locked. Regions should be locked only briefly and should be unlocked before closing a file or exiting the program. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_locking.asp
51764. ** Utility functions
51765. device inode resides on
51766. ** Convert the inode protection mode to a string.
51767. ** Creates a link. ** @param #1 Object to link to. ** @param #2 Name of link. ** @param #3 True if link is symbolic (optional).
51768. Lua 5.3
51769. group-id of owner
51770. ** Convert the inode protection mode to a permission list.
51771. **comment:** number of hard links to the file
label: documentation
51772. first entry
51773. ** Removes a directory. ** @param #1 Directory path.
51774. ** Assumes the table is on top of the stack.
51775. MAX_PATH seems to be 260. Seems kind of small. Is there a better one?
51776. ** Get file information using stat.
51777. time of last access
51778. ** Creates a directory. ** @param #1 Directory path.
51779. Passing (NULL, 0) is not guaranteed to work. Use a temp buffer and size instead.
51780. ** Closes directory iterators
51781. Linux, Solaris and HP-UX
51782. ** Check if the given element on the stack is a file and returns it.
51783. inode protection mode
51784. AIX
51785. time of last file status change
51786. ** Creates directory metatable.
51787. ** This function changes the working (current) directory
51788. Define 'getcwd' for systems that do not implement it
51789. For MAXPATHLEN:
51790. ** Locks a file. ** @param #1 File handle. ** @param #2 String with lock mode ('w'rite, 'r'ead). ** @param #3 Number with start position (optional). ** @param #4 Number with length (optional).

51791. stores all members in table on top of the stack
51792. ** Set access time and modification values for file
51793. ** Luaf.FileSystem ** Copyright Kepler Project 2003 (<http://www.keplerproject.org/luafilesystem>) *** File system manipulation library. ** This library offers these functions: ** lfs.attributes (filepath [, attributename]) ** lfs.chdir (path) ** lfs.currentdir () ** lfs.dir (path) ** lfs.lock (fh, mode) ** lfs.lock_dir (path) ** lfs.mkdir (path) ** lfs.rmdir (path) ** lfs.setmode (filepath, mode) ** lfs.symlinkattributes (filepath [, attributename]) -- thanks to Sam Roberts ** lfs.touch (filepath [, atime [, mtime]]) ** lfs.unlock (fh) *** \$Id: lfs.c,v 1.61 2009/07/04 02:10:16 mascarenhas Exp \$
51794. ** Directory iterator
51795. ** Get file or symbolic link information
51796. creates a table if none is given
51797. ** Creates lock metatable.
51798. optimal file system I/O blocksize
51799. no more entries => close directory
51800. Method table
51801. inode's number
51802. member not found
51803. Define 'strerror' for systems that do not implement it
51804. permissions string
51805. ** This function returns the current directory ** If unable to get the current directory, it returns nil ** and a string describing the error
51806. Metamethods
51807. ** Unlocks a file. ** @param #1 File handle. ** @param #2 Number with start position (optional). ** @param #3 Number with length (optional).
51808. set to current date/time
51809. ** Luaf.FileSystem ** Copyright Kepler Project 2003 (<http://www.keplerproject.org/luafilesystem>) *** \$Id: lfs.h,v 1.5 2008/02/19 20:08:23 mascarenhas Exp \$
51810. Define 'chdir' for systems that do not implement it
51811. ***** * lssqlite3 * * Copyright (C) 2002-2016 Tiago Dionizio, Doug Currie * *
All rights reserved. ** Author : Tiago Dionizio <tiago.dionizio@ist.utl.pt> ** Author : Doug Currie <doug.currie@alum.mit.edu> * * Library : lssqlite3 - an SQLite 3 database binding for Lua 5 * * * Permission is hereby granted, free of charge, to any person obtaining * * a copy of this software and associated documentation files (the * * "Software"), to deal in the Software without restriction, including * * without limitation the rights to use, copy, modify, merge, publish, * * distribute, sublicense, and/or sell copies of the Software, and to * * permit persons to whom the Software is furnished to do so, subject to * * the following conditions: * * * The above copyright notice and this permission notice shall be * * included in all copies or substantial portions of the Software. * * * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND. * * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF * * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.* * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY * * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, * * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE * * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. * *****
51812. push error message
51813. db sql svm_ud reg[db_lud] svm_lud db --
51814. compile time features
51815. error codes
51816. Version number of this library
51817. safety measures for userdata field to be present in the stack
51818. From: Wolfgang Oertl When using lssqlite3 in a multithreaded environment, each thread has a separate Lua environment, but full userdata structures can't be passed from one thread to another. This is possible with lightuserdata, however. See: lssqlite_open_ptr().
51819. not yet created? - shouldn't happen in finalize function
51820. use result if there was no error
51821. make sure we have an userdata field (even if nil)
51822. backup structure
51823. register (local) sqlite metatable
51824. top SQL function being called
51825. Source database name
51826. i think it is OK to use assume that using a light user data ** as an entry on LUA REGISTRY table will be unique
51827. save user data
51828. things done properly (SQLite + Lua SQLite) ** this should never happen
51829. set update_hook handler
51830. clear busy handler
51831. column names not known at this point
51832. 'trap' userdata - nil extra parameters
51833. userdata is a no-no as it will be garbage-collected
51834. disable extension loading
51835. push metatable
51836. rollback_hook callback
51837. ** ===== ** Virtual Machine - generic info **
=====
51838. sqlite3_step info
51839. push nil
51840. sql functions stack usage
51841. metatable.__index = metatable
51842. sdb_vm *svm =
51843. enable extension loading
51844. context table
51845. !
51846. set sqlite's metatable to itself - set as readonly (__newindex)
51847. failed to open database
51848. clear commit_hook handler
51849. true when step succeeds
51850. so, assert(load_extension(...)) works
51851. #if !defined(LSQLITE OMIT_UPDATE_HOOK) || !LSQLITE OMIT_UPDATE_HOOK
51852. virtual machine
51853. target db
51854. ** Params: db, sql ** returns: code, compiled length or error message
51855. to prevent referenced databases from being garbage collected while bu is live
51856. set trace handler
51857. lua_setopt(L,3) above means we don't need: lua_pushvalue(L,3);
51858. unpacked version of db:rows
51859. information about database
51860. Lua 5.3 introduced an integer type, but depending on the implementation, it could be 32 ** or 64 bits (or something else?). This helper macro tries to do "the right thing."
51861. reuse context userdata value
51862. stack: ** 3: callback function ** 4: userdata ** 5: column names ** 6: reusable column values

51863. set commit_hook handler
51864. save registered function in db function list
51865. add source and target dbs to table at indices 1 and 2
51866. stepvm may change svm->vm if re-prepare is needed
51867. finalize and check for errors
51868. key: vm; val: sql text
51869. virtual machine information
51870. free allocated memory
51871. update_hook callback
51872. luaL_typerror always used with arg at ndx == NULL
51873. put table in registry with key lightuserdata bu
51874. { "__tostring", dbbu_tostring },
51875. not yet created?
51876. push context - used to set results
51877. else ignore if already finished
51878. column types
51879. create table from registry
51880. ** When creating database handles, always creates a `closed` database handle ** before opening the actual database; so, if there is a memory error, the ** database is not left opened. ** ** Creates a new 'table' and leaves it in the stack
51881. db,sql is on top of stack for call to newvm
51882. ** trace callback: ** Params: database, callback function, userdata ** ** callback function: ** Params: userdata, sql
51883. setup lua callback call
51884. set context
51885. clear update_hook handler
51886. Authorizer Action Codes
51887. ** Register a normal function ** Params: db, function name, number arguments, [callback | step, finalize], user data ** Returns: true on sucess ** ** Normal function: ** Params: context, params ** ** Aggregate function: ** Params of step: context, params ** Params of finalize: context
51888. already finished
51889. **comment:** cleanup context
 label: code-design
51890. column values table
51891. clear trace handler
51892. **comment:** if there was a timeout callback registered, it is now ** invalid/useless. free any references we may have
 label: code-design
51893. save the setup/normal function callback
51894. call lua function
51895. clear rollback_hook handler
51896. ** update_hook callback: ** Params: database, callback function, userdata ** ** callback function: ** Params: userdata, {one of SQLITE_INSERT, SQLITE_DELETE, or SQLITE_UPDATE}, ** database name, table name (containing the affected row), rowid of the row
51897. ===== ** User Defined Functions - Context Methods **
=====
51898. ** Params: db, sql, callback, user ** returns: code [, errmsg] ** ** Callback: ** Params: user, number of columns, values, names ** Returns: 0 to continue, other value will cause abort
51899. ignore closed vms
51900. sqlite3_db_filename may return NULL, in that case Lua pushes nil...
51901. ===== ** Database (internal management) **
=====
51902. invalidate context
51903. terminator
51904. close database
51905. ignore any error generated by this function
51906. i think it is OK to use assume that using a light user data ** as an entry on LUA REGISTRY table will be unique
51907. Destination database name
51908. get callback
51909. associated lua state
51910. conversion warning: int64 -> luaNumber
51911. create and leave in stack
51912. ===== ** Virtual Machine - Bind **
=====
51913. database handle is currently `closed'
51914. number of opcodes
51915. associated database handle
51916. ===== ** Database Methods **
=====
51917. pop vm table
51918. ===== ** Virtual Machine - getters **
=====
51919. function to call
51920. create_collation; contributed by Thomas Lauer
51921. number of columns in result
51922. get callback user data
51923. minimal mem usage
51924. remove entry in database table - no harm if not present in the table
51925. **comment:** cleanup temporary only tables?
 label: code-design
51926. vm already in the stack
51927. scalar function to be called ** callback params: context, values...
51928. db sql svm_ud reg[db_lud] svm_lud --
51929. luaL_openlib always used with name == NULL
51930. **comment:** maybe an alternative way to allocate memory should be used/avoided
 label: code-design
51931. set progress callback
51932. references
51933. sqlite database handle
51934. to use as C user data so i know what function sqlite is calling
51935. called with db,sql text on the lua stack
51936. commit_hook callback
51937. 'free' all references

51938. ensure there is enough space in the stack
51939. set metatable
51940. to keep track of 'open' virtual machines
51941. ** busy handler: ** Params: database, callback function, userdata ** ** callback function: ** Params: userdata, number of tries ** returns: 0 to return immediately and return SQLITE_BUSY, non-zero to try again
51942. busy callback
51943. (reg[db_lud])[svm_lud] = db ; set the db for this vm
51944. columns names
51945. this should not happen since sqlite3_prepare_v2 will not set ->vm on error
51946. set rollback_hook handler
51947. #if !defined(SQLITE_OMIT_PROGRESS_CALLBACK) || !SQLITE_OMIT_PROGRESS_CALLBACK
51948. free associated memory with created functions
51949. Online Backup API
51950. progress handler
51951. sqlite_step specific return values
51952. From: Wolfgang Oertl When using lssqlite3 in a multithreaded environment, each thread has a separate Lua environment, but full userdata structures can't be passed from one thread to another. This is possible with lightuserdata, however. See: db_get_ptr().
51953. add constants to global table
51954. temporary vm used in db:rows
51955. references to associated lua values
51956. **comment:** clean things up
 label: code-design
51957. Thanks to Wolfgang Oertl...
51958. ** ===== ** Database Virtual Machine Operations **
=====

51959. database handle already in the stack - return it
51960. trace callback
51961. valid as soon as statement prepared
51962. ** Lua 5.2
51963. no callbacks
51964. return
51965. safety measure
51966. ignore closed databases
51967. close all used handles
51968. compatibility names (added by request)
51969. default args to nil, and exclude extras
51970. ** progress handler: ** Params: database, number of opcodes, callback function, userdata ** ** callback function: ** Params: userdata ** returns: 0 to return immediately and return SQLITE_ABORT, non-zero to continue
51971. user data
51972. luaL_register used once, so below expansion is OK for this case
51973. db sql svm_ud reg[db_lud] --
51974. end of Online Backup API
51975. Source database handle
51976. set busy handler
51977. ** ===== ** General library functions **
=====

51978. abort by default
51979. push params
51980. Destination database handle
51981. ** Registering SQL functions:
51982. ** commit_hook callback: ** Params: database, callback function, userdata ** ** callback function: ** Params: userdata ** Returned value: Return false or nil to continue the COMMIT operation normally. ** return true (non false, non nil), then the COMMIT is converted into a ROLLBACK.
51983. remove entry in lua registry table
51984. ** rollback hook callback: ** Params: database, callback function, userdata ** ** callback function: ** Params: userdata
51985. remove metatable from stack
51986. ** Lua 5.3
51987. =====
51988. db sql svm_ud db_lud --
51989. remove it from registry
51990. ** callback functions used when calling registered sql functions
51991. traced sql statement
51992. add an entry on the database table: svm -> db to keep db live while svm is live
51993. source db
51994. column values
51995. file open flags
51996. db sql svm_ud --
51997. free associated virtual machines
51998. ** ===== ** Register functions **
=====

51999. update_hook database name
52000. This is the only API function that runs sqlite3SafetyCheck regardless of * SQLITE_ENABLE_API_ARMOR and does almost nothing (without an SQL * statement)
52001. remove table from registry
52002. register metatable functions
52003. should be defined in rockspec, but just in case...
52004. leave key in the stack
52005. total number of rows in result
52006. Last change: revised for Lua 5.1.5
52007. EOF
52008. \$Id: lua.man,v 1.11 2006/01/06 16:03:34 lhf Exp \$
52009. www.lua.org
52010. EOF
52011. \$Id: luac.man,v 1.28 2006/01/06 16:03:34 lhf Exp \$
52012. www.lua.org
52013. (S)
52014. =====
52015. (I)
52016. private part

52017. \$Id: manual.of,v 1.49.1.2 2012/01/13 20:23:26 roberto Exp \$
52018. (n)
52019. (u) number of upvalues
52020. Last change: revised for Lua 5.1.5
52021. get global 'f'
52022. * all.c -- Lua core, libraries and interpreter in a single file
52023. lua.hpp Lua header files for C++ <<extern "C">> not supplied automatically because Lua also compiles as C++
52024. * min.c -- a minimal Lua interpreter * loads stdin only with minimal error handling. * no interaction, and no standard library, only a "print" function.
52025. * The code below can be used to make a Lua core that does not contain the * parsing modules (lcode, llex, lparser), which represent 35% of the total core. * You'll only be able to load binary files and strings, precompiled with luac. * (Of course, you'll have to build luac with the original parsing modules!) * * To use this module, simply compile it ("make noparser" does that) and list * its object file before the Lua libraries. The linker should then not load * the parsing modules. To try it, do "make luab". * * If you also want to avoid the dump module (ldump.o), define NODUMP. * #define NODUMP
52026. GC values are expressed in Kbytes: #bytes/2^10
52027. function to be called
52028. stack overflow
52029. data to `f_Ccall'
52030. may call tag method
52031. ** push functions (C -> stack)
52032. explicit test for incompatible code
52033. invalid option
52034. ensure that true is 1
52035. ** access functions (stack -> C)
52036. conversion failed?
52037. ** `load' and `call' functions (run Lua code)
52038. signal it
52039. pop value
52040. ** set functions (stack -> Lua)
52041. use global table as environment
52042. previous call may reallocate the stack
52043. push function
52044. ** Execute a protected C call.
52045. pop index and value
52046. `subtract' index (index is negative)
52047. else n == 1; nothing to do
52048. ** Garbage-collection function
52049. remove key
52050. `luaV_tostring' may create a new string
52051. pseudo-indices
52052. ** \$Id: lapi.c,v 2.55.1.5 2008/07/04 18:41:18 roberto Exp \$ ** Lua API ** See Copyright Notice in lua.h
52053. ** get functions (Lua -> stack)
52054. no enclosing function?
52055. data to `f_call'
52056. ** Execute a protected call.
52057. **comment:** to avoid warnings
 label: code-design
52058. push empty string
52059. no more elements
52060. ** miscellaneous functions
52061. ** basic stack manipulation
52062. push only argument
52063. end of cycle?
52064. function upvalue?
52065. ** \$Id: lapi.h,v 2.2.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Auxiliary functions from Lua API ** See Copyright Notice in lua.h
52066. name already in use?
52067. remove previous table
52068. remove both metatables
52069. is there a numeric field `n'?
52070. push prefix
52071. get registry.name
52072. else try sizes[t]
52073. remove table and value
52074. no metafield?
52075. remove it from list
52076. use `sizes'
52077. remove this nil
52078. no stack frame?
52079. put buffer before new value
52080. free list of references
52081. remove only metatable
52082. binary file?
52083. }=====

52084. push last suffix
52085. no metatable?
52086. ** \$Id: lauxlib.c,v 1.159.1.3 2008/01/21 13:20:51 roberto Exp \$ ** Auxiliary functions for building Lua libraries ** See Copyright Notice in lua.h
52087. metatable(N).__mode = "kv"
52088. copy upvalues to the top
52089. This file uses only the official API of Lua. ** Any function declared here could be written as an application function.
52090. remove it from stack
52091. sizes[t] = n
52092. put nothing on stack
52093. move library table to below upvalues
52094. add new value into B stack
52095. t[ref] = t[FREELIST_REF]
52096. get first free element
52097. remove from stack
52098. return problematic part of the name
52099. reopen in binary mode
52100. convert a stack index to positive

52101. do not count `self'
52102. ref = t[FREELIST_REF]
52103. leave previous value on top, but return 0
52104. get info about it
52105. get _LOADED[libname]
52106. no free elements
52107. create metatable
52108. index of filename on the stack
52109. store in register
52110. use it
52111. value is a userdata?
52112. **comment:** to avoid warnings
 label: code-design
52113. continue after `p'
52114. field has a non-table value?
52115. else error
52116. set new table into field
52117. remove `sizes'
52118. remove _LOADED table
52119. check function at level
52120. put it there
52121. does it have the correct mt?
52122. (t[FREELIST_REF] = t[ref])
52123. new table for field
52124. check whether lib already exists
52125. Unix exec. file?
52126. create it
52127. push replacement in place of pattern
52128. number of levels to concat
52129. _LOADED[libname] = new table
52130. no `size' table?
52131. ignore results from `lua_load'
52132. any free element?
52133. does it have a metatable?
52134. try global variable (and create one if it does not exist)
52135. remove nil
52136. ** {===== ** Generic Buffer manipulation **
=====**
52137. try t.n
52138. registry.name = metatable
52139. ** {===== ** Load functions **
=====**
52140. t[FREELIST_REF] = ref
52141. skip first line
52142. fit into buffer?
52143. remove upvalues
52144. `size' will be its own metatable
52145. remove metatable and metafield
52146. create new reference
52147. remove previous result
52148. close file (even in case of errors)
52149. get correct metatable
52150. ** {===== ** Error-report functions **
=====**
52151. else, no information available...
52152. error is in the self argument itself?
52153. `nil' has a unique fixed reference
52154. not found?
52155. ** {===== ** getn-setn: size for arrays **
=====**
52156. no such field?
52157. is there info?
52158. **comment:** avoid extra test when d is not 0
 label: test
52159. skip eventual `#!...'
52160. current position in buffer
52161. extra error code for `luaL_load'
52162. number of strings in the stack (level)
52163. ** \$Id: lauxlib.h,v 1.88.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Auxiliary functions for building Lua libraries ** See Copyright Notice in lua.h
52164. compatibility with ref system
52165. }=====**
52166. no op!
52167. compatibility only
52168. pre-defined references
52169. ** {===== ** Generic Buffer manipulation **
=====**
52170. ** ===== ** some useful macros **
=====**
52171. function to be called
52172. change environment of current thread
52173. set global _VERSION
52174. and initial value
52175. weaktable[m] = true
52176. skip trailing spaces
52177. metatable(w).__mode = "kv"
52178. return nil plus error message
52179. return status + all results
52180. number of elements

52181. error object is a string?
52182. =====
52183. set global _G
52184. `newproxy' needs a weaktable as upvalue
52185. put error function under function to be called
52186. 'normal' (it resumed another coroutine)
52187. add extra info
52188. pop result
52189. move function to top
52190. create proxy
52191. running
52192. standard conversion
52193. metatable is valid; get it
52194. function, eventual name, plus one reserved slot
52195. ** \$Id: lbaselib.c,v 1.191.1.6 2008/02/14 16:46:22 roberto Exp \$ ** Basic library ** See Copyright Notice in lua.h
52196. return false + error message
52197. move yielded values
52198. no metatable
52199. open lib into global table
52200. it is running
52201. new table 'w'
52202. return the thread's global env.
52203. set global `newproxy'
52204. **comment:** to avoid warnings
 label: code-design
52205. value to print
52206. ... and mark `m' as a valid metatable
52207. `ipairs' and `pairs' need auxiliary functions as upvalues
52208. else not a number
52209. save string in a reserved stack slot
52210. push arg[i] (avoiding overflow problems)
52211. return generator,
52212. move error message
52213. error flag
52214. to check if weaktable[metatable(u)] == true
52215. state,
52216. next value
52217. propagate error
52218. returns either __metatable field (if present) or metatable
52219. is a C function?
52220. at least one valid digit?
52221. create a 2nd argument if there isn't one
52222. put before error message
52223. call it
52224. some error occurred
52225. return true + `resume' returns
52226. move function from L to NL
52227. suspended
52228. ** Reader for generic `load' function: `lua_load' uses the ** stack for internal stuff, so the reader cannot change the ** stack top. Instead, it keeps its resulting string in a ** reserved slot inside the stack.
52229. n <= 0 means arith. overflow
52230. empty range
52231. remove value
52232. no invalid trailing characters?
52233. **comment:** ** If your system does not support `stdout', you can just remove this function. ** If you need, you can define your own `print' function, following this ** model but changing `fputs' to put the strings at a proper place ** (a console window or a log file, for instance).
 label: code-design
52234. initial state
52235. OK?
52236. is there a metafield?
52237. push arg[i + 1...e]
52238. does it have frames?
52239. number of arguments
52240. create a new metatable `m' ...
52241. ===== ** Coroutine library **
=====
52242. `w' will be its own metatable
52243. add extra information?
52244. use its value
52245. main thread is not a coroutine
52246. get result
52247. get function
52248. position after whole expression
52249. operand must be on the `stack'
52250. operand must be on the 'stack'
52251. insert last jump in `f' list
52252. find last element
52253. free registers with list values
52254. invalid var kind to store
52255. jump to default target
52256. put this jump in 't' list
52257. insert last jump in 't' list
52258. else go through
52259. exp is already in a register
52260. end of list
52261. turn offset into absolute position
52262. cannot happen
52263. always false; do nothing

52264. not found
52265. remove previous OP_NOT
52266. put new instruction in code array
52267. interchange true and false lists
52268. constant not found; create a new entry
52269. nothing to do...
52270. constant fit in RK operand?
52271. cannot patch other instructions
52272. those instructions may be jump targets
52273. position of an eventual LOAD true
52274. positions are already clean
52275. there is one value available (somewhere)
52276. o1 <==> o2
52277. cannot use nil as key; instead use table itself to represent nil
52278. do not attempt to divide by 0
52279. point to itself represents end of list
52280. reg. is not a local?
52281. default
52282. no jumps to current position?
52283. ** returns current 'pc' and marks it as a jump target (to avoid wrong ** optimizations with consecutive instructions not in the same basic block).
52284. save corresponding line information
52285. do not attempt to produce NaN
52286. exchange args to replace by '<' or '<='
52287. expression is an open function call?
52288. else no optimization
52289. cannot operate on non-numeric constants
52290. ** check whether list has any jump that do not produce a value ** (or produce an inverted value)
52291. position of an eventual LOAD false
52292. keep them on hold
52293. constant fit in argC?
52294. save list of jumps to here
52295. not a constant in the right range: put it in a register
52296. always true; do nothing
52297. no constant folding for 'len'
52298. function start?
52299. 'pc' will change
52300. can connect both?
52301. no register to put value or register already has the value
52302. put value on it
52303. can relocate its simple result
52304. pc of last jump
52305. list must be closed
52306. cannot operate on constants
52307. ** \$Id: lcode.c,v 2.25.1.5 2011/01/31 14:53:16 roberto Exp \$ ** Code generator for Lua ** See Copyright Notice in lua.h
52308. ** grep "ORDER OPR" if you change these enums
52309. ** Marks the end of a patch list. It is an invalid value both as an absolute ** address, and as a list link (would link an element to itself).
52310. ** \$Id: lcode.h,v 1.48.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Code generator for Lua ** See Copyright Notice in lua.h
52311. set hooks
52312. external hook?
52313. set new hook
52314. out of range?
52315. level 0 may be this own function
52316. size of the first part of the stack
52317. **comment:** too many levels
 label: code-design
52318. level out of range
52319. message is not a string
52320. cannot touch C upvalues from Lua
52321. turn off hooks
52322. no metatable
52323. remove eventual returns
52324. return table
52325. size of the second part of the stack
52326. **comment:** no more than 'LEVELS2' more levels?
 label: code-design
52327. ** \$Id: ldblib.c,v 1.104.1.4 2009/08/04 18:50:18 roberto Exp \$ ** Interface from Lua to its debug API ** See Copyright Notice in lua.h
52328. remove hook table
52329. is there a name?
52330. C function or tail call
52331. find last levels
52332. keep going
52333. still before eventual '...'!
52334. get hook
52335. main?
52336. do the jump
52337. only active lua functions have current-line information
52338. level found?
52339. a Lua function?
52340. affect all registers above base
52341. change register `a'
52342. move from `b' to `a'
52343. level is of a lost tail call?
52344. does it jump?
52345. global index
52346. is there an error handling function?
52347. invalid option
52348. invalid instruction after an open call
52349. Lua function?

52350. not full check and jump is forward and do not skip `lastpc'?
52351. is Lua code?
52352. pop value
52353. not found
52354. add file:line information
52355. skip lost tail calls
52356. skip the '>'
52357. calling function
52358. no tail call?
52359. function is not a Lua function?
52360. push function
52361. points to final return (a 'neutral' instruction)
52362. pop function
52363. is 'n' inside 'ci' stack?
52364. space for results
52365. calling function is not Lua (or is unknown)
52366. check skip
52367. is a local?
52368. set registers from 'a' to 'b'
52369. at least two operands
52370. turn off hooks?
52371. at least one result (control variable)
52372. b = num, returns
52373. check that it does not jump to a setlist count; this is tricky, because the count from a previous setlist may have the same value of an invalid setlist; so, we must go all the way back to the first of them (if any)
52374. c = num, returns
52375. ** \$Id: ldebug.c,v 2.29.1.6 2008/05/08 16:56:26 roberto Exp \$ ** Debug Interface ** See Copyright Notice in lua.h
52376. upvalue index
52377. no useful name can be found
52378. move argument
52379. try symbolic execution
52380. get name for 'b'
52381. key index
52382. ** this function can be called asynchronous (e.g. during a signal)
52383. if 'j' is even, previous value is not a setlist (even if it looks like one)
52384. ======
52385. do not 'execute' these pseudo-instructions
52386. affect all regs above its base
52387. handled by lua_getinfo
52388. only ANSI way to check whether a pointer points to an array
52389. no useful name found
52390. no such level
52391. is a local variable in a Lua function
52392. call it
52393. check its jump
52394. ** ====== ** Symbolic Execution and code checker **
======
52395. go through
52396. first operand is wrong
52397. stores position of last instruction that changed 'reg'
52398. tracing?
52399. ** \$Id: ldebug.h,v 2.3.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Auxiliary functions from Debug Interface module ** See Copyright Notice in lua.h
52400. error while handing stack error
52401. tag method is the new function to be called
52402. ** ====== ** Error-recovery functions **
======
52403. **comment:** compat. with old-style vararg?
 label: code-design
52404. number of extra arguments
52405. add 'arg' parameter
52406. correct 'pc'
52407. yielding?
52408. 0 iff wanted == LUA_MULTRET
52409. 'func' is not a function?
52410. move results to correct place
52411. double size is enough?
52412. ** Execute a protected parser.
52413. initialize eventual upvalues
52414. an error occurred?
52415. **comment:** put extra arguments into 'arg' table
 label: code-design
52416. close eventual pending closures
52417. tail call; no debug information about it
52418. Lua function?
52419. finish interrupted execution of 'OP_CALL'
52420. final position of first argument
52421. do the actual call
52422. Open a hole inside the stack at 'func'
52423. next call may change stack
52424. tail calls
52425. start coroutine?
52426. restore savedpc
52427. hooks assume 'pc' is already incremented
52428. protect stack slots below
52429. ** Call a function (C or Lua). The function to be called is at *func. ** The arguments are on the stack, right after the function. ** When returns, all the results are on the stack, starting at the original ** function position.
52430. restore old error handler
52431. res == final position of 1st result

52432. error message on current top
52433. can `undo' overflow?
52434. move fixed parameters to final position
52435. starting point
52436. error?
52437. ensure minimum stack size
52438. yielded inside a hook; just continue its execution
52439. mark thread as `dead'
52440. first fixed argument
52441. overflow while handling overflow?
52442. error code
52443. now `enter' new function
52444. restore base
52445. vararg function
52446. is a Lua function?
52447. ** \$Id: ldo.c,v 2.38.1.4 2012/01/18 02:27:10 roberto Exp \$ ** Stack and Call structure of Lua ** See Copyright Notice in lua.h
52448. create `arg' table
52449. =====
52450. store counter in field `n'
52451. previous call may change stack
52452. resuming from previous yield
52453. complete it...
52454. data to `f_parser'
52455. buffer to be used by the scanner
52456. `common' yield?
52457. Lua function? prepare its call
52458. chain new error handler
52459. there was an overflow?
52460. check the `function' tag method
52461. no varargs?
52462. chain list of long jump buffers
52463. call it
52464. if is a C function, call it
52465. and correct top if not multiple results
52466. previous call may change the stack
52467. cannot call hooks inside a hook
52468. results from luaD_precall
52469. ** \$Id: ldo.h,v 2.7.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Stack and Call structure of Lua ** See Copyright Notice in lua.h
52470. did a call to a C function
52471. **comment:** type of protected functions, to be ran by `runprotected'
 label: code-design
52472. initiated a call to a Lua function
52473. C function yielded
52474. ** \$Id: ldump.c,v 2.8.1.1 2007/12/27 13:02:25 roberto Exp \$ ** save precompiled Lua chunks ** See Copyright Notice in lua.h
52475. cannot happen
52476. ** dump Lua function as precompiled chunk
52477. include trailing '\0'
52478. **comment:** is it dead?
 label: code-design
52479. current value lives in the stack
52480. found a corresponding upvalue?
52481. is it open?
52482. not found: create a new one
52483. free upvalue
52484. chain it in the proper position
52485. ** \$Id: lfunc.c,v 2.12.1.2 2007/12/28 14:58:43 roberto Exp \$ ** Auxiliary functions to manipulate prototypes and closures ** See Copyright Notice in lua.h
52486. remove from `uvhead' list
52487. ** Look for n-th local variable at line `line' in function `func'. ** Returns NULL if not found.
52488. is variable active?
52489. not found
52490. resurrect it
52491. remove from open list
52492. double link it in `uvhead' list
52493. remove from `open' list
52494. now current value lives here
52495. link upvalue into `gcroot' list
52496. ** \$Id: lfunc.h,v 2.4.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Auxiliary functions to manipulate prototypes and closures ** See Copyright Notice in lua.h
52497. remove entry from table
52498. closed upvalues need barrier
52499. restore threshold
52500. clear bits
52501. mark nested protos
52502. is there a weak mode?
52503. keep it gray
52504. link `curr' at the end of `tmudata' list
52505. mark basic metatables (again)
52506. end sweep phase
52507. ** The next function tells whether a key or value can be cleared from ** a weak table. Non-collectable objects are never removed from weak ** tables. Strings
 behave as `values', so are never removed too. for ** other objects: if really collected, cannot keep them; for userdata ** being finalized, keep them in keys, but not
 in values
52508. strings are `values', so are never weak
52509. make global table be traversed before main stack
52510. free all string lists
52511. ** traverse one gray object, turning it to black. ** Returns `quantity' traversed.
52512. mask to collect all elements
52513. first estimate
52514. creates a circular list
52515. - lim/g->gcstepmul;

52516. must keep invariant?
52517. ** \$Id: lgc.c,v 2.38.1.2 2011/03/18 18:05:38 roberto Exp \$ ** Garbage Collector ** See Copyright Notice in lua.h
52518. don't need finalization
52519. **comment:** table is too big
 label: code-design
52520. may be marked, if left from previous GC
52521. **comment:** remark occasional upvalues of (maybe) dead threads
 label: code-design
52522. do not touch the stacks
52523. mark running thread
52524. sweep open upvalues of each thread
52525. mark as white just to avoid other barriers
52526. ** Call all GC tag methods
52527. remark gray again
52528. adjust first
52529. number of 'ci' in use
52530. start a new collection
52531. total size of userdata to be finalized
52532. move 'dead' udata that need finalization to list 'tmudata'
52533. no more 'gray' objects
52534. is really weak?
52535. don't mind
52536. value was collected?
52537. must erase 'curr'
52538. must be cleared after GC, ...
52539. open upvalues are never black
52540. remove value ...
52541. mark its upvalues
52542. dead key; remove it
52543. ... so put in the appropriate list
52544. mark root set
52545. end sweep-string phase
52546. reset sweep marks to sweep all elements (returning them to white)
52547. traverse objects caught by write barrier and by 'remarkupvals'
52548. make it white (for next cycle)
52549. make table gray (again)
52550. link upvalue into 'rootgc' list
52551. must call its gc method
52552. return it to 'root' list
52553. finish any pending sweep phase
52554. finish mark phase
52555. end collection
52556. flip current white
52557. mark upvalue names
52558. check size of buffer
52559. ** clear collected entries from weaktables
52560. separate userdata to be finalized
52561. handling overflow?
52562. remove collected objects from weak tables
52563. restore hooks
52564. list is empty?
52565. **comment:** not dead?
 label: code-design
52566. is the first element of the list?
52567. get first element
52568. nothing more to sweep?
52569. non-empty entry?
52570. ** All marks are conditional because a GC may happen while the ** prototype is still being created
52571. still big enough...
52572. remove value
52573. no limit
52574. closed?
52575. check size of string hash
52576. reset other collector lists
52577. mark 'preserved' userdata
52578. remark weak tables
52579. avoid GC steps
52580. stop debug hooks during GC tag method
52581. mark literals
52582. remove empty entries
52583. udata are never gray
52584. mark local-variable names
52585. buffer too big?
52586. restore invariant
52587. table is weak?
52588. don't bother with them
52589. remove udata from 'tmudata'
52590. part of stack in use
52591. lua_checkmemory(L);
52592. last element?
52593. sweep phase: sweep it (turning it into white)
52594. remark, to propagate 'preserveness'
52595. **comment:** ** some useful bit tricks
 label: code-design
52596. ** \$Id: lgc.h,v 2.15.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Garbage Collector ** See Copyright Notice in lua.h
52597. **comment:** ** Layout for bit use in 'marked' field: ** bit 0 - object is white (type 0) ** bit 1 - object is white (type 1) ** bit 2 - object is black ** bit 3 - for userdata: has been finalized ** bit 3 - for tables: has weak keys ** bit 4 - for tables: has weak values ** bit 5 - object is fixed (should not be collected) ** bit 6 - object is "super" fixed (only the main thread)

label: code-design

52598. ** Possible states of the Garbage Collector
52599. ** \$Id: linit.c,v 1.14.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Initialization of libraries for lua.c ** See Copyright Notice in lua.h
52600. create (and set) default files
52601. ** {===== ** READ **
=====

52602. calls to Lua API may change this value
52603. how much to read
52604. close it
52605. ** \$Id: liolib.c,v 2.73.1.4 2010/05/14 15:33:51 roberto Exp \$ ** Standard I/O (and system) library ** See Copyright Notice in lua.h
52606. pop 'popen'
52607. push nil instead
52608. not a file
52609. until end of count or eof
52610. file is already closed?
52611. open library
52612. do not include `eol'
52613. file methods
52614. ** function to (not) close the standard files stdin, stdout, and stderr
52615. line
52616. "result" to be removed
52617. to return 1 result
52618. copy environment
52619. ensure stack space for all results and for auxlib's buffer
52620. error
52621. EOF
52622. close/not close file when finished
52623. will iterate over default input
52624. create environment for 'popen'
52625. file handle is currently 'closed'
52626. pop environment for default files
52627. return current value
52628. ** When creating file handles, always creates a `closed' file handle ** before opening the actual file; so, if there is a memory error, the ** file is not left opened.
52629. close function for default files
52630. create metatable for file handles
52631. no arguments?
52632. number
52633. generator created file?
52634. push metatable
52635. set fenv for 'popen'
52636. eof?
52637. number of chars actually read
52638. read MAX_SIZE_T chars
52639. ======

52640. create (private) environment (with fields IO_INPUT, IO_OUTPUT, __close)
52641. read at least an `eol'
52642. read fails
52643. ** function to close regular files
52644. always success
52645. ignore closed files
52646. remove last result
52647. metatable.__index = metatable
52648. set it
52649. make sure argument is a file
52650. file
52651. still have to read `n' chars
52652. ** this function has a separated environment, which defines the ** correct __close for 'popen' files
52653. try to read that much each time
52654. optimization: could be done exactly as for strings
52655. ** function to close 'popen' files
52656. close buffer
52657. cannot read more than asked
52658. check whether read something
52659. check that it's a valid file handle
52660. read next token
52661. skip 2nd `]'
52662. LUA_NUMBER
52663. skip 2nd `['
52664. ORDER RESERVED
52665. reserved word?
52666. `E'?
52667. read first char
52668. ...
52669. \xxx
52670. skip delimiter
52671. follow locale for decimal point
52672. reserved word
52673. handles `\\', `\" , `\' , and `\\?`
52674. try updated decimal separator
52675. reserved words are never collected
52676. optional exponent sign
52677. else is a comment
52678. identifier or reserved word
52679. ..
52680. try to update decimal point separator
52681. else short comment
52682. initialize buffer
52683. undo change (for error message)

52684. string starts with a newline?
52685. no look-ahead token
52686. ** \$Id: llex.c,v 2.20.1.2 2009/11/23 14:58:22 roberto Exp \$ ** Lexical Analyzer ** See Copyright Notice in lua.h
52687. will raise an error next loop
52688. skip '\n\r' or '\r\n'
52689. make sure `str' will not be collected
52690. **comment:** to avoid warnings
 label: code-design
52691. **comment:** avoid warnings when `cont' is not used
 label: code-design
52692. entry for `str'
52693. format error: try to update decimal point separator
52694. single-char tokens (+ - ...)
52695. long comment
52696. skip it
52697. is there a look-ahead token?
52698. use this one
52699. format error with correct decimal point: no more options
52700. do not save the `\'
52701. avoid wasting space
52702. `skip_sep' may dirty the buffer
52703. and discharge it
52704. go through
52705. ** ===== ** LEXICAL ANALYZER **
=====

52706. skip '\n' or '\r'
52707. format error?
52708. input line counter
52709. input stream
52710. current token
52711. current source name
52712. * WARNING: if you change the order of this enumeration, * grep "ORDER RESERVED"
52713. semantics information
52714. other terminal symbols
52715. ** \$Id: llex.h,v 1.58.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Lexical Analyzer ** See Copyright Notice in lua.h
52716. buffer for tokens
52717. locale decimal point
52718. look ahead token
52719. line of last token `consumed'
52720. number of reserved words
52721. current character (charint)
52722. array with token `names'
52723. terminal symbols denoted by reserved words
52724. 'FuncState' is private to the parser
52725. maximum length of a reserved word
52726. ** \$Id: llimits.h,v 1.69.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Limits, basic types, and some other `installation-dependent' definitions ** See Copyright Notice in lua.h
52727. type to ensure maximum alignment
52728. minimum size for string buffer
52729. ** conversion of pointer to integer ** this is for hashing only; there is no problem if the integer ** cannot hold the whole pointer value
52730. **comment:** ** macro to control inclusion of some hard tests on stack reallocation
 label: test
52731. ** type for virtual-machine instructions ** must be an unsigned with (at least) 4 bytes (see details in lopcodes.h)
52732. maximum stack for a Lua function
52733. minimum size for the string table (must be power of 2)
52734. chars used as small naturals (so that `char' is reserved for characters)
52735. result of a `usual argument conversion' over lua_Number
52736. maximum value of an int (-2 for safety)
52737. **comment:** to avoid warnings
 label: code-design
52738. internal assertions for in-house debugging
52739. only upper limit
52740. number of arguments
52741. Number between 0 and 1
52742. no arguments
52743. ** Open math library
52744. ** \$Id: lmthlib.c,v 1.67.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Standard mathematical library ** See Copyright Notice in lua.h
52745. lower and upper limits
52746. check number of arguments
52747. int between `l' and `u'
52748. int between 1 and `u'
52749. the `%' avoids the (rare) case of r==1, and is needed also because on some systems (SunOS!) `rand()' may return a value larger than RAND_MAX
52750. cannot double it?
52751. cannot grow even a little?
52752. update only when everything else is OK
52753. minimum size
52754. ** generic allocation routine.
52755. still have at least one free place
52756. ** \$Id: lmem.c,v 1.70.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Interface to Memory Manager ** See Copyright Notice in lua.h
52757. **comment:** to avoid warnings
 label: code-design
52758. ** About the realloc function: ** void * frealloc (void *ud, void *ptr, size_t osize, size_t nszie); ** ('osize' is the old size, `nszie' is the new size) *** ** Lua ensures that (ptr == NULL) iff (osize == 0). *** ** * frealloc(ud, NULL, 0, x) creates a new block of size `x' *** ** * frealloc(ud, p, x, 0) frees the block `p' ** (in this specific case, frealloc must return NULL). ** particularly, frealloc(ud, NULL, 0, 0) does nothing ** (which is equivalent to free(NULL) in ANSI C) *** ** frealloc returns NULL if it cannot create or reallocate the area ** (any reallocation to an equal or smaller size cannot fail!)

52759. **comment:** +1 to avoid warnings
 label: code-design
52760. ** \$Id: lmem.h,v 1.31.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Interface to Memory Manager ** See Copyright Notice in lua.h

52761. no errors?
52762. remove file name
52763. get _LOADED[modname]
52764. _LOADED[name] = true
52765. root not found
52766. unable to load library
52767. loader returned error message?
52768. create new metatable
52769. not found
52770. ** {===== ** This is an implementation of loadlib based on the dlfcn interface. ** The dlfcn interface is available in Linux, SunOS, Solaris, IRIX, FreeBSD, ** NetBSD, AIX 4.2, HPUX 11, and probably most other Unix flavors, at least ** as an emulation layer on top of native functions. **
=====

52771. is table an initialized module?
52772. package is already loaded
52773. did it find module?
52774. check loops
52775. fill it with pre-defined loaders
52776. unable to find function
52777. use default
52778. library loaded successfully
52779. open failed
52780. }=====

52781. module did not set a value?
52782. no entry yet; create one
52783. does file exist and is readable?
52784. look for last dot in module name
52785. non-nil return?
52786. get a loader
52787. return nil, error message, and where
52788. check whether table already has a _NAME field
52789. this would be a rare case, but prevents crashing if it happens
52790. ** __gc tag method: calls library's `ll_unloadlib' function with the lib ** handle
52791. try to open file
52792. return the loaded function
52793. error codes for ll_loadfunc
52794. index of _LOADED table
52795. set field `preload'
52796. open lib into global table
52797. ** {===== ** Fallback for other systems **
=====

52798. return that file name
52799. _LOADED table will be at index 2
52800. mt._index = _G
52801. **comment:** to avoid warnings
 label: code-design
52802. put it in field `loaders'
52803. module._M = module
52804. error accumulator
52805. auxiliary mark (for internal use)
52806. module loaded successfully
52807. _LOADED[name] = sentinel
52808. function not found
52809. skip separators
52810. template
52811. Mac appends a '_' before C function names
52812. is it there?
52813. _LOADED[name] = returned value
52814. set field `cpath'
52815. get calling function
52816. library not found in this path
52817. find next separator
52818. replace ";" by ";AUXMARK;" and then AUXMARK by default path
52819. module
52820. return function
52821. run loaded module
52822. error message accumulator
52823. no; initialize it
52824. pass name as argument to module
52825. real error
52826. create new type _LOADLIB
52827. else must load it; iterate over available loaders
52828. add entry to possible error message
52829. ** \$Id: loadlib.c,v 1.52.1.4 2009/09/09 13:17:16 roberto Exp \$ ** Dynamic library loader for Lua ** See Copyright Notice in lua.h ** ** This module contains an implementation of loadlib for Unix systems ** that have dlfcn, an implementation for Darwin (Mac OS X), an ** implementation for Windows, and a stub for other systems.
52830. no more templates
52831. create 'package' table
52832. try global variable (and create one if it does not exist)
52833. check library in registry?
52834. ** {===== ** 'require' function **
=====

52835. set field `path'
52836. remove path template
52837. no environment variable?
52838. error; error message is on stack top
52839. store config information
52840. use true as result

52841. set field `loaded'
52842. call it
52843. separator for open functions in C libraries
52844. _LOADED[modname] = new table
52845. remove previous result
52846. get option (a function)
52847. remove original string
52848. is root
52849. prefix for open functions in C libraries
52850. extra copy to be returned
52851. set _PACKAGE as package name (full module name minus last part)
52852. ** {===== ** Native Mac OS X / Darwin Implementation **
=====**
52853. mark library as closed
52854. not found?
52855. ** {===== ** This is an implementation of loadlib for Windows using
native functions. ** =====**
52856. return 'package' table
52857. ** {===== ** 'module' function **
=====**
52858. is there an entry?
52859. remove 'gsub' result
52860. accumulate it
52861. create 'loaders' table
52862. maybe an hexadecimal constant?
52863. skip the `@'
52864. should be enough space for a `%'
52865. remove first char
52866. ** converts an integer to a "floating point byte", represented as ** (eeeeexxx), where the real value is (1xxx) * 2^(eeee - 1) if ** eeeee != 0 and (xxx) otherwise.
52867. conversion failed
52868. most common case
52869. converts back
52870. out = [string "string"]
52871. out = "source", or "...source"
52872. boolean true must be 1 !!
52873. this function handles only `%d', `%c', %f, %p, and `%' formats
52874. exponent
52875. must truncate?
52876. ensures null termination
52877. invalid trailing characters?
52878. get last part of file name
52879. ** \$Id: lobject.c,v 2.22.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Some generic functions over Lua objects ** See Copyright Notice in lua.h
52880. stop at first newline
52881. ensures maximum alignment for strings
52882. tags for values visible from Lua
52883. the value (when closed)
52884. information about local variables
52885. Macros to access values
52886. Macros to test type
52887. Macros to set values
52888. constants used by the function
52889. to new object
52890. to table
52891. ensures maximum alignment for 'local' udata
52892. functions defined inside the function
52893. ** \$Id: lobject.h,v 2.20.1.2 2008/08/06 13:29:48 roberto Exp \$ ** Type definitions for Lua objects ** See Copyright Notice in lua.h
52894. **comment:** first point where variable is dead
 label: code-design
52895. ** for internal debug only
52896. index to stack elements
52897. double linked list (when open)
52898. ** Common header in struct form
52899. ** Union of all collectable objects
52900. ** String headers for string table
52901. size of 'p'
52902. log2 of size of `node' array
52903. upvalue names
52904. from table to same table
52905. from stack to (same) stack
52906. masks for new-style vararg
52907. 1<<p means tagmethod(p) is not present
52908. map from opcodes to source lines
52909. size of `array' array
52910. for chaining
52911. array part
52912. ** Tables
52913. size of 'k'
52914. ** 'module' operation for hashing (size is always a power of 2)
52915. ** Common Header for all collectable objects (in macro form, to be ** included in other objects)
52916. ** Tagged Values
52917. ** Extra tags for non-values
52918. ** Upvalues
52919. ** Union of all Lua values
52920. first point where variable is active
52921. ** different types of sets, according to destination
52922. number of upvalues
52923. any free position is before this position
52924. ** Closures

52925. to stack (not from same stack)
 52926. ** Function Prototypes
 52927. points to stack or to its own value
 52928. OP_LOADNIL
 52929. OP_UNM
 52930. OP_MOVE
 52931. OP_SELF
 52932. OP_VARARG
 52933. OP_RETURN
 52934. OP_JMP
 52935. OP_FORPREP
 52936. OP_CLOSURE
 52937. OP_LE
 52938. OP_DIV
 52939. OP_NOT
 52940. OP_LEN
 52941. OP_EQ
 52942. OP_SETUPVAL
 52943. OP_POW
 52944. OP_LT
 52945. OP_SETGLOBAL
 52946. OP_GETGLOBAL
 52947. OP_TFORLOOP
 52948. OP_SETLIST
 52949. T A B C mode opcode
 52950. ORDER OP
 52951. OP_CONCAT
 52952. OP_SETTABLE
 52953. OP_MUL
 52954. OP_LOADK
 52955. OP_GETTABLE
 52956. OP_LOADBOOL
 52957. OP_FORLOOP
 52958. OP_MOD
 52959. OP_NEWTABLE
 52960. OP_TEST
 52961. OP_TAILCALL
 52962. ** \$Id: lopcodes.c,v 1.37.1.1 2007/12/27 13:02:25 roberto Exp \$ ** See Copyright Notice in lua.h
 52963. OP_CLOSE
 52964. OP_ADD
 52965. OP_CALL
 52966. OP_SUB
 52967. OP_GETUPVAL
 52968. OP_TESTSET
 52969. **comment:** argument is not used
 label: code-design
 52970. ** invalid register that fits in 8 bits
 52971. number of list items to accumulate before a SETLIST instruction
 52972. A B C R(A) := R(B)[RK(C)]
 52973. A B C if ((RK(B) == RK(C)) ~= A) then pc++
 52974. A C if not (R(A) <=> C) then pc++
 52975. A B C if (R(B) <=> C) then R(A) := R(B) else pc++
 52976. ** masks for instruction properties. The format is: ** bits 0-1: op mode ** bits 2-3: C arg mode ** bits 4-5: B arg mode ** bit 6: instruction set register A ** bit 7: operator is a test
 52977. test whether value is a constant
 52978. A close all variables in the stack up to (\geq) R(A)
 52979. A B R(A), R(A+1), ..., R(A+B-1) = vararg
 52980. A Bx Gbl[Kst(Bx)] := R(A)
 52981. ** R(x) - register ** Kst(x) - constant (in constant table) ** RK(x) == if ISK(x) then Kst(INDEXK(x)) else R(x)
 52982. ===== We assume that instructions are unsigned numbers. All
 instructions have an opcode in the first 6 bits. Instructions can have the following fields: `A' : 8 bits `B' : 9 bits `C' : 9 bits `Bx' : 18 bits ('B' and 'C' together)`sBx'
 : signed Bx A signed argument is represented in excess K; that is, the number value is the unsigned value minus K. K is exactly the maximum value for that
 argument (so that -max is represented by 0, and +max is represented by 2^{max}), which is half the maximum for the corresponding unsigned argument.
 =====
 52983. A B C R(A+1) := R(B); R(A) := R(B)[RK(C)]
 52984. A Bx R(A) := Gbl[Kst(Bx)]
 52985. A B C return R(A)(R(A+1), ... ,R(A+B-1))
 52986. this bit 1 means constant (0 means register)
 52987. A Bx R(A) := Kst(Bx)
 52988. ** Macros to operate RK indices
 52989. basic instruction format
 52990. A B C R(A) := RK(B) ^ RK(C)
 52991. ** the following macros help to manipulate instructions
 52992. A B C R(A)[RK(B)] := RK(C)
 52993. A C R(A+3), ... ,R(A+2+C) := R(A)(R(A+1), R(A+2)); if R(A+3) ~= nil then R(A+2)=R(A+3) else pc++
 52994. ** \$Id: lopcodes.h,v 1.125.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Opcodes for Lua virtual machine ** See Copyright Notice in lua.h
 52995. A B C R(A) := RK(B) - RK(C)
 52996. A sBx R(A)+=R(A+2); if R(A) <= R(A+1) then { pc+=sBx; R(A+3)=R(A) }
 52997. A B C R(A)[(C-1)*FPF+i] := R(A+i), 1 <= i <= B
 52998. `sBx' is signed
 52999. A B C R(A) := {} (size = B,C)
 53000. A B UpValue[B] := R(A)
 53001. A sBx R(A)-=R(A+2); pc+=sBx
 53002. ----- name args description -----
 53003. sBx pc+=sBx
 53004. A B C R(A) := RK(B) / RK(C)
 53005. ** size and position of opcode arguments.
 53006. A B R(A) := not R(B)

53007. A B R(A) := ... := R(B) := nil
 53008. A B C R(A) := RK(B) * RK(C)
 53009. ** limits for opcode arguments. ** we use (signed) int to manipulate most arguments, ** so they must fit in LUAI_BITSINT-1 bits (-1 for sign)
 53010. argument is used
 53011. A B R(A) := length of R(B)
 53012. A B R(A) := R(B)
 53013. opcode names
 53014. ** grep "ORDER OP" if you change these enums
 53015. argument is a register or a jump offset
 53016. A B C R(A) := (Bool)B; if (C) pc++
 53017. gets the index of the constant
 53018. argument is a constant or register/constant
 53019. A B C if ((RK(B) <= RK(C)) ~= A) then pc++
 53020. A B return R(A), ... ,R(A+B-2) (see note)
 53021. A B C R(A) := RK(B) % RK(C)
 53022. A Bx R(A) := closure(KPROTO[Bx], R(A), ... ,R(A+n))
 53023. A B R(A) := -R(B)
 53024. A B R(A) := UpValue[B]
 53025. A B C R(A) := R(B)..R(C)
 53026. A B C if ((RK(B) < RK(C)) ~= A) then pc++
 53027. A B C R(A), ... ,R(A+C-2) := R(A)(R(A+1), ... ,R(A+B-1))
 53028. code a constant index as a RK value
 53029. ===== Notes: (*) In OP_CALL, if (B == 0) then B = top. C is the number of returns - 1, and can be 0: OP_CALL then sets `top' to last_result+1, so next open instruction (OP_CALL, OP_RETURN, OP_SETLIST) may use `top'. (*) In OP_VARARG, if (B == 0) then use actual number of varargs and set top (like in OP_CALL with C == 0). (*) In OP_RETURN, if (B == 0) then return up to `top' (*) In OP_SETLIST, if (B == 0) then B = `top'; if (C == 0) then next 'instruction' is real C (*) For comparisons, A specifies what condition the test should accept (true or false). (*) All `skips' (pc++) assume that next instruction is a jump
 =====
 53030. creates a mask with `n' 1 bits at position `p'
 53031. creates a mask with `n' 0 bits at position `p'
 53032. A B C R(A) := RK(B) + RK(C)
 53033. UTC?
 53034. ** ===== Time/Date operations ** { year=%Y, month=%m, day=%d, hour=%H, min=%M, sec=%S, ** wday=%w+1, yday=%j, isdst=? } ** =====
 53035. 9 = number of fields
 53036. calls to Lua API may change this value
 53037. skip '!'
 53038. undefined?
 53039. should be big enough for any conversion result
 53040. if NULL push nil
 53041. get current time
 53042. no conversion specifier?
 53043. called without args?
 53044. make sure table is at the top
 53045. invalid date?
 53046. does not set field
 53047. ** \$Id: loslib.c,v 1.19.1.3 2008/01/18 16:38:18 roberto Exp \$ ** Standard Operating System library ** See Copyright Notice in lua.h
 53048. }=====
 53049. pretend that `OP_FOR' starts the loop
 53050. a block either controls scope or breaks (never both)
 53051. body -> `(` parlist `)' chunk END
 53052. stat -> assignment
 53053. end of scope for declared variables
 53054. read first token
 53055. assignment -> `,' primaryexp assignment
 53056. param -> `...'
 53057. `:' NAME funcargs
 53058. not found
 53059. `[' exp1 `]'
 53060. expand while operators have priorities higher than 'limit'
 53061. block -> chunk
 53062. stat -> forstat
 53063. no value (yet)
 53064. finish scope...
 53065. fix it at stack top (for gc)
 53066. skip WHILE
 53067. free registers
 53068. =====
 53069. stat -> func | assignment
 53070. no more levels?
 53071. reserve register for parameters
 53072. ** structure to chain all variables in the left-hand side of an ** assignment
 53073. forlist -> NAME {,NAME} IN explist1 forbodystat
 53074. `falses' are all equal here
 53075. control variables
 53076. left priority for each binary operator
 53077. no upvalues?
 53078. skip DO
 53079. local function?
 53080. remove table and prototype from the stack
 53081. forstat -> FOR (fornum | forlist) END
 53082. stat -> RETURN explist
 53083. cond -> exp
 53084. create control variables
 53085. chain
 53086. last list item read
 53087. simpleexp -> NUMBER | STRING | NIL | true | false | ... | constructor | FUNCTION body | primaryexp
 53088. only one single value?

53089. funcargs
53090. constructor -> ??
53091. scope block
53092. skip BREAK
53093. logical (and/or)
53094. debug information will only see the variable after this point!
53095. and repeat
53096. may be listfields or recfields
53097. close the loop
53098. return no values
53099. includes call itself
53100. open call
53101. funcstat -> FUNCTION funcname body
53102. equality and inequality
53103. set initial array size
53104. tail call?
53105. may be needed for error messages
53106. last exp. provides the difference
53107. ** \$Id: lp parser.c,v 2.42.1.4 2011/10/21 19:31:42 roberto Exp \$ ** Lua Parser ** See Copyright Notice in lua.h
53108. linked list of funcstates
53109. ** nodes for block list (list of active blocks)
53110. IF cond THEN block
53111. stat -> breakstat
53112. flush
53113. anchor table of constants and prototype (to avoid being collected)
53114. call statement uses no results
53115. create declared variables
53116. arg list is empty?
53117. `+` `-` `/` `%`
53118. upvalue in this level
53119. skip the 'l'
53120. default is global variable
53121. stat -> retstat
53122. must be last statement
53123. stat -> DO block END
53124. chunk -> { stat [';'] }
53125. # active locals outside the breakable structure
53126. else was LOCAL or UPVAL
53127. optional step
53128. there is no list item
53129. constructor
53130. field
53131. return all 'active' values
53132. ** ===== ** Rules for Constructors **
=====

53133. registers with returned values
53134. true if some variable in the block is an upvalue
53135. limit
53136. return first untreated operator
53137. default step = 1
53138. **comment:** to avoid warnings
 label: code-design
53139. close last argument
53140. fornum -> NAME = exp1,exp1[,exp1] forbody
53141. `then' part
53142. field -> ['. | ':'] NAME
53143. ELSEIF cond THEN block
53144. ** check whether, in an assignment to a local variable, the local variable ** is needed in a previous assignment (to a table). If so, save original ** local value in a safe place and use this safe copy in the previous ** assignment.
53145. ORDER OPR
53146. info points to global name
53147. variable (global, local, upvalue, or indexed)
53148. previous assignment will use safe copy
53149. ls->t.token == 'l'
53150. don't need 'arg'
53151. index -> '[' expr ']'
53152. finish scope
53153. }=====

53154. explist1 -> expr { `,' expr }
53155. funcname -> NAME {field} [': NAME]
53156. skip RETURN
53157. **comment:** remove extra values
 label: code-design
53158. stat -> funcstat
53159. conflict?
53160. stat -> func
53161. stat -> whilestat
53162. skip the dot or colon
53163. expression?
53164. initial value
53165. registers 0/1 are always valid
53166. true if 'block' is a loop
53167. ** ===== ** Expression parsing **
=====

53168. scope for loop and control variables
53169. eventual position to save local variable
53170. prefixexp -> NAME | '(' expr ')'
53171. make copy

53172. if condition then break
53173. param -> NAME
53174. stat -> repeatstat
53175. `else' part
53176. do not count last expression (unknown number of elements)
53177. vararg
53178. power and concat (right associative)
53179. GRAMMAR RULES
53180. repeatstat -> REPEAT block UNTIL cond
53181. **comment**: extra space to call generator
 label: code-design
53182. ** subexpr -> (simpleexp | unop subexpr) { binop subexpr } ** where `binop' is any binary operator with a priority higher than `limit'
53183. avoid default
53184. finish loop
53185. ** ====== ** Rules for Statements **
=====

53186. forbody -> DO block
53187. optional return values
53188. definition 'happens' in the first line
53189. read condition
53190. call remove function and arguments and leaves (unless changed) one result
53191. false conditions finish the loop
53192. ** prototypes for recursive non-terminal functions
53193. skip REPEAT
53194. table descriptor
53195. assignment -> '=' explist1
53196. test_then_block -> [IF | ELSEIF] cond THEN block
53197. stat -> ifstat
53198. right priority
53199. no more items pending
53200. at least one expression
53201. complete semantics when there are upvalues
53202. skip 'for'
53203. read sub-expression with higher priority
53204. whilestat -> WHILE cond DO block END
53205. loop block
53206. list of jumps out of this loop
53207. look up at current level
53208. constructor_item -> recfield
53209. funcargs -> `(` [explist1] `)`
53210. read condition (inside scope block)
53211. total number of 'record' elements
53212. values must go to the 'stack'
53213. order
53214. priority for unary operators
53215. new one
53216. scope for declared variables
53217. not found at current level; try upper one
53218. main func. is always vararg
53219. primaryexp -> prefixexp { `.' NAME | `[' exp `]' | `:' NAME funcargs | funcargs }
53220. close last expression
53221. use 'arg' as default name
53222. base register for call
53223. local will be used as an upval
53224. constructor_part -> listfield
53225. first variable name
53226. stat -> LOCAL NAME { `,' NAME} [= explist1]
53227. funcargs -> constructor
53228. return all values
53229. must use 'seminfo' before 'next'
53230. number of array elements pending to be stored
53231. else...
53232. skip ELSE (after patch, for correct line info)
53233. skip LOCAL
53234. set initial table size
53235. skip FUNCTION
53236. stat -> localstat
53237. default assignment
53238. loop scope ('break' jumps to this point)
53239. funcargs -> STRING
53240. recfield -> (NAME | `[' exp1 `]`) = exp1
53241. skip IF or ELSEIF
53242. total number of array elements
53243. ======

53244. final return
53245. is 'parlist' not empty?
53246. ifstat -> IF cond THEN block {ELSEIF cond THEN block} [ELSE block] END
53247. parlist -> [param { `,' param }]
53248. last token read was anchored in defunct function; must reanchor it
53249. ** \$Id: lparser.h,v 1.57.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Lua Parser ** See Copyright Notice in lua.h
53250. patch list of 'exit when true'
53251. number of elements in 'locvars'
53252. info = result register
53253. chain of current blocks
53254. copy of the Lua state
53255. list of pending jumps to 'pc'
53256. info = index of constant in 'k'
53257. table to find (and reuse) elements in 'k'

53258. current function header
53259. upvalues
53260. no value
53261. info = index of upvalue in `upvalues'
53262. next position to code (equivalent to `ncode')
53263. first free register
53264. lexical state
53265. number of elements in `k'
53266. info = table register; aux = index register (or `k')
53267. enclosing function
53268. declared-variable stack
53269. info = index of table; aux = index of global name in `k'
53270. state needed to generate code for a given function
53271. patch list of `exit when false'
53272. nval = numerical value
53273. info = instruction pc
53274. info = local register
53275. number of active local variables
53276. defined in lparser.c
53277. number of elements in `p'
53278. `pc' of last `jump target'
53279. ** Expression descriptor
53280. memory allocation error: free partial state
53281. collect all objects
53282. mark it as unfinished state
53283. separate udata that have GC metamethods
53284. close all upvalues for this thread
53285. registry
53286. only the main thread can be closed
53287. initialize first ci
53288. initialize CallInfo array
53289. repeat until no more errors
53290. table of globals
53291. ** \$Id: lstate.c,v 2.36.1.2 2008/01/03 15:20:39 roberto Exp \$ ** Global State ** See Copyright Notice in lua.h
53292. init stack
53293. `function' entry for this `ci'
53294. ** Main thread combines a thread state and the global state
53295. no error function during GC metamethods
53296. share table of globals
53297. ** open parts that may cause memory-allocation errors
53298. initial size of string table
53299. initialize stack array
53300. call GC metamethods for all udata
53301. defined in ldo.c
53302. list of open upvalues in this stack
53303. metatables for basic types
53304. nested C calls when resuming coroutine
53305. array with tag-method names
53306. thread
53307. expected number of results from this function
53308. to be called in unprotected errors
53309. current error recover point
53310. last free slot in the stack
53311. number of tail calls lost under this entry
53312. **comment:** temporary place for environments
 label: code-design
53313. ** `global state', shared by all threads of this state
53314. size of pause between successive GCs
53315. base for this function
53316. registry
53317. last element of list of userdata to be GC
53318. GC `granularity'
53319. first free slot in the stack
53320. `savedpc' of current function
53321. stack base
53322. call info for current function
53323. an estimate of number of bytes actually in use
53324. ** Union of all collectable objects
53325. position of sweep in `strt'
53326. table of globals
53327. **comment:** temporary buffer for string concatenation
 label: code-design
53328. ** `per thread' state
53329. base of current function
53330. how much GC is `behind schedule'
53331. list of objects to be traversed atomically
53332. points after end of ci array
53333. macros to convert a GCOBJECT into a specific value
53334. number of elements
53335. number of bytes currently allocated
53336. head of double-linked list of all open upvalues
53337. macro to convert any Lua object into a GCOBJECT
53338. current error handling function (stack index)
53339. hash table for strings
53340. list of all collectable objects
53341. number of nested C calls
53342. function index in the stack
53343. array of CallInfo's

53344. ** \$Id: lstate.h,v 2.24.1.2 2008/01/03 15:20:39 roberto Exp \$ ** Global State ** See Copyright Notice in lua.h
53345. ** informations about a call
53346. extra stack space to handle TM calls and some other extras
53347. state of garbage collector
53348. position of sweep in `rootgc'
53349. list of gray objects
53350. list of weak tables (to be cleared)
53351. top for this function
53352. size of array 'base_ci'
53353. function to reallocate memory
53354. auxiliary data to `frealloc'
53355. rehash
53356. chain new entry
53357. not found
53358. save next
53359. ending 0
53360. seed
53361. chain it on udata list (after main thread)
53362. ** \$Id: lstring.c,v 2.8.1.1 2007/12/27 13:02:25 roberto Exp \$ ** String table (keeps all strings handled by Lua) ** See Copyright Notice in lua.h
53363. is not finalized
53364. compute hash
53365. if string is too long, don't hash all its chars
53366. too crowded
53367. chain it
53368. for each node in the list
53369. new position
53370. **comment:** string may be dead
 label: code-design
53371. cannot resize during GC traverse
53372. ** \$Id: lstring.h,v 1.43.1.1 2007/12/27 13:02:25 roberto Exp \$ ** String table (keep all strings handled by Lua) ** See Copyright Notice in lua.h
53373. `s2' cannot be found after that
53374. explicit request?
53375. else return match(ms, s, ep);
53376. 0 or more repetitions
53377. ** {===== ** PATTERN MATCHING **
=====

53378. not found
53379. add capture to accumulated result
53380. 1st char is already checked
53381. else return match(ms, s, ep+1);
53382. using goto's to optimize tail recursion
53383. start
53384. position capture?
53385. relative string position: negative means back from end
53386. to store the format ('%...')
53387. end capture
53388. pop metatable
53389. create metatable for strings
53390. }=====

53391. end of pattern
53392. macro to `unsign' a character
53393. 0 or more repetitions (minimum)
53394. skip it
53395. format item
53396. avoids a negative `l1'
53397. string library...
53398. add whole match
53399. balanced string?
53400. ...is the __index metamethod
53401. matches any char
53402. points to what is next
53403. maximum size of each formatted item (> len(format("%99.99f", -1e308)))
53404. else return match(ms, s, p+2)
53405. keep original text
53406. pop dummy string
53407. **comment:** no precision and string is too long to be formatted; keep original string
 label: code-design
53408. empty match? go at least one position
53409. empty interval; return no values
53410. match succeeded
53411. string ends out of balance
53412. 1 or more repetitions
53413. ** \$Id: lstrlib.c,v 1.132.1.5 2010/05/14 15:34:19 roberto Exp \$ ** Standard library for string operations and pattern-matching ** See Copyright Notice in lua.h
53414. skip the 'addsize' at the end
53415. else return match(ms, s+1, ep);
53416. skip the 'addsize' at the end
53417. to search for a `*s2' inside `s1'
53418. init of source string
53419. number of strings pushed
53420. do a plain search
53421. empty strings are everywhere
53422. to store the formatted item
53423. skip escapes (e.g. `%'])
53424. non empty match?
53425. optional
53426. ** maximum size of each format specification (such as '%-099.99d') ** (+10 accounts for %99.99x plus margin of error)
53427. %%
53428. skip precision

53429. it is a pattern item
53430. set string metatable
53431. ms->level == 0, too
53432. skip width
53433. end
53434. skip ESC
53435. add result to accumulator
53436. undo capture
53437. total number of captures (finished or unfinished)
53438. look for a `]'
53439. (2 digits at most)
53440. else didn't match; reduce 1 repetition to try again
53441. valid flags in a format specification
53442. start capture
53443. ** Open string library
53444. number of substitutions
53445. skip the `^'
53446. end (`\0') of source string
53447. keeps trying to match with the maximum repetitions
53448. is the `'\$ the last char in pattern?
53449. overflow?
53450. 1st char will be checked by `memchr'
53451. capture results (%0-%9)?
53452. check end of string
53453. try with one more repetition
53454. counts maximum expand for item
53455. frontier?
53456. case default
53457. else return match(ms, s, p+4);
53458. skip flags
53459. match failed?
53460. number of arguments
53461. also treat cases `pnLlh'
53462. close capture
53463. correct `l1' and `s1' to try again
53464. nil or false?
53465. dummy string
53466. or no special characters?
53467. shrink array
53468. cannot find a free place?
53469. i is zero or a present index
53470. **comment:** key may be dead already, but it is ok to use it in `next'
 label: code-design
53471. no more elements to count
53472. compute new size for array part
53473. check whether `key' is somewhere in the chain
53474. grow table
53475. count as such
53476. count extra key
53477. that's it
53478. copy colliding node into free pos. (mp->next also goes)
53479. redo the chain with `n' in place of `mp'
53480. is `key' inside array part?
53481. yes; that's the index (corrected to C)
53482. key index in hash table
53483. is `key' an appropriate array index?
53484. ** hash for lua_Numbers
53485. try first array part
53486. nums[i] = number of keys between $2^{(i-1)}$ and 2^i
53487. ** number of ints inside a lua_Number
53488. value
53489. then hash part
53490. use specialized version
53491. 2^i
53492. now `mp' is free
53493. temporary values (kept only if some malloc fails)
53494. adjust upper limit
53495. optimal size (till now)
53496. all elements smaller than n will go to array part
53497. re-insert key into grown table
53498. index is int?
53499. new node will go into free position
53500. first iteration
53501. ** max size of array part is $2^{MAXBITS}$
53502. count elements in range ($2^{(lg-1)}$, 2^{lg}]
53503. ** main search function
53504. array part must grow?
53505. else must find a boundary in hash part
53506. re-insert elements from hash part
53507. array part must shrink?
53508. table was built with bad purposes: resort to linear search
53509. key not found
53510. **comment:** ** for some types, it is better to avoid modulus by power of 2, as ** they tend to have many 2 factors.
 label: code-design
53511. is colliding node out of its main position?
53512. ** search function for strings
53513. avoid problems with -O
53514. **comment:** to avoid warnings

label: code-design

53515. could not find a free place
53516. find previous
53517. total number of elements
53518. for each slice
53519. 'key' did not match some condition
53520. number of elements to go to array part
53521. else go through
53522. find original element
53523. ** inserts a new key into a hash table; first, check whether key's main ** position is free. If not, check whether colliding node is in its main ** position or not: if it is not, move colliding node to an empty place and ** put new key in its main position; otherwise (colliding node is in its main ** position), new key goes to an empty position.
53524. count to traverse all array keys
53525. reset counts
53526. save old hash ...
53527. optimal size for array part
53528. all elements already counted
53529. resize the table to new computed sizes
53530. ** returns the index of a 'key' for table traversals. First goes all ** elements in the array part, then elements in the hash part. The ** beginning of a traversal is signalled by -1.
53531. ** search function for integers
53532. all those keys are integer keys
53533. colliding node is in its own main position
53534. 2^{\lg}
53535. key
53536. a non-nil value?
53537. counter
53538. count keys in array part
53539. now do a binary search between them
53540. chain new position
53541. use common `dummynode'
53542. summation of 'nums'
53543. overflow?
53544. no elements to hash part?
53545. all positions are free
53546. re-insert elements from vanishing slice
53547. hash elements are numbered after array ones
53548. find 'i' and 'j' such that i is present and j is not
53549. there is a boundary in the array part: (binary) search for it
53550. more than half elements present
53551. ** }=====

53552. that is easy...
53553. ** \$Id: ltable.c,v 2.32.1.2 2007/12/28 15:32:23 roberto Exp \$ ** Lua tables (hash) ** See Copyright Notice in lua.h
53554. hash part is empty?
53555. **comment:** ** Implementation of tables (aka arrays, objects, or hash tables). ** Tables keep its elements in two parts: an array part and a hash part. ** Non-negative integer keys are all candidates to be kept in the array ** part. The actual size of the array is the largest `n` such that at ** least half the slots between 0 and n are in use. ** Hash uses a mix of chained scatter table with Brent's variation. ** A main invariant of these tables is that, if an element is not ** in its main position (i.e. the 'original' position that its hash gives ** to it), then the colliding element is in its own main position. ** Hence even when the load factor reaches 100%, performance remains good.

label: code-design

53556. ** {===== ** Rehash ** ======

53557. (1 <= key && key <= t->sizearray)
53558. create new hash part with appropriate size
53559. free old array
53560. number of elements smaller than 2^i
53561. yes; move colliding node into free position
53562. count keys in hash part
53563. ** returns the index for 'key' if 'key' is an appropriate key to live in ** the array part of the table, -1 otherwise.
53564. ** Try to find a boundary in table 't'. A 'boundary' is an integer index ** such that t[i] is non-nil and t[i+1] is nil (and 0 if t[1] is nil).
53565. no more elements
53566. ** returns the 'main' position of an element in a table (that is, the index ** of its hash value)
53567. get a free place
53568. ** \$Id: ltable.h,v 2.10.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Lua tables (hash) ** See Copyright Notice in lua.h
53569. insert new element at the end
53570. remove a[j]
53571. ** \$Id: ltablib.c,v 1.38.1.3 2008/02/14 16:46:58 roberto Exp \$ ** Library for Table Manipulation ** See Copyright Notice in lua.h
53572. 'grow' array if necessary
53573. -2 to compensate function and 'a'
53574. t[i] = t[i-1]
53575. result = t[pos]
53576. repeat --j until a[j] <= P
53577. call recursively the smaller one
53578. called with only 2 arguments
53579. t[e] = nil
53580. remove value
53581. assume array is smaller than 2^{40}
53582. a < b?
53583. Pivot
53584. function
53585. remove nil result
53586. move up elements
53587. remove value and result
53588. nothing to remove
53589. repeat the routine for the larger one
53590. -1 to compensate function
53591. first key
53592. adjust so that smaller half is in [j..i] and larger one in [l..u]

53593. is there a 2nd argument?
53594. 2nd argument is the position
53595. t[pos] = t[pos+1]
53596. a[i]<a[l]?
53597. ===== ** Quicksort ** (based on `Algorithms in MODULA-3', Robert Sedgewick; ** Addison-Wesley, 1993.)
53598. add last value (if interval was not empty)
53599. remove a[l]
53600. pop pivot, a[i], a[j]
53601. a[l] <= P == a[u-1] <= a[u], only need to sort from l+1 to u-2
53602. t.n = n-1
53603. for tail recursion
53604. only 3 elements
53605. first empty element
53606. key
53607. a[l..i-1] <= a[i] == P <= a[i+1..u]
53608. swap pivot (a[u-1]) with a[i]
53609. invariant: a[l..i] <= P <= a[j..u]
53610. sort elements a[l], a[(l+u)/2] and a[u]
53611. swap a[l] - a[u]
53612. }=====

53613. a[u] < a[l]?
53614. a[u]<a[i]?
53615. 2nd argument
53616. only 2 elements
53617. value
53618. function?
53619. position is outside bounds?
53620. new size
53621. 1st argument
53622. where to insert new element
53623. repeat ++i until a[i] >= P
53624. remove a[i]
53625. make sure there is two arguments
53626. t[pos] = v
53627. never collect these names
53628. **comment:** ** function to be used with macro "fasttm": optimized for absence of ** tag methods
 label: code-design
53629. cache this fact
53630. no tag method?
53631. ** \$Id: ltm.c,v 2.8.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Tag methods ** See Copyright Notice in lua.h
53632. ORDER TM
53633. * WARNING: if you change the order of this enumeration, * grep "ORDER TM"
53634. number of elements in the enum
53635. ** \$Id: ltm.h,v 2.6.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Tag methods ** See Copyright Notice in lua.h
53636. last tag method with `fast' access
53637. stdin
53638. repeat until gets a complete line
53639. open libraries
53640. number of arguments to the script
53641. not an option?
53642. remove line
53643. invalid option
53644. create state
53645. remove it
53646. pass error message
53647. stop collector during initialization
53648. executes stdin as a file
53649. line ends with newline?
53650. skip this function and traceback
53651. count total number of arguments
53652. status OK
53653. function index
53654. push traceback function
53655. stop if file fails
53656. option
53657. any result to print?
53658. if another SIGINT happens before lstop, terminate process (default action)
53659. call debug.traceback
53660. join them
53661. invalid args?
53662. ** \$Id: lua.c,v 1.160.1.2 2007/12/28 15:32:23 roberto Exp \$ ** Lua stand-alone interpreter ** See Copyright Notice in lua.h
53663. no more input?
53664. check that argument has no extra characters at the end
53665. cannot try to add lines?
53666. add a new line...
53667. go through
53668. force a complete garbage collection in case of errors
53669. put it under chunk and args
53670. else...
53671. remove global
53672. change it to `return'
53673. clear stack
53674. 'message' not a string?
53675. keep it intact
53676. first line starts with '=' ?
53677. ...between the two lines
53678. **comment:** unused arg.

label: code-design

53679. remove traceback function

53680. collect arguments

53681. no input

53682. ===== ** some useful macros =====

53683. minimum Lua stack available to a C function

53684. ** functions that read/write blocks when loading/dumping Lua chunks

53685. ** basic types

53686. ** push functions (C -> stack)

53687. ** prototype for memory-allocation functions

53688. ** 'load' and 'call' functions (load and run Lua code)

53689. ** access functions (stack -> C)

53690. ** garbage-collection function and options

53691. (n) 'global', 'local', 'field', 'method'

53692. (S)

53693. ** set functions (stack -> Lua)

53694. thread status; 0 is OK

53695. (l)

53696. ** state manipulation

53697. ** \$Id: lua.h,v 1.218.1.7 2012/01/13 20:36:20 roberto Exp \$ ** Lua - An Extensible Extension Language ** Lua.org, PUC-Rio, Brazil (<http://www.lua.org>) ** See Copyright Notice at the end of this file

53698. option for multiple returns in 'lua_pcall' and 'lua_call'

53699. ***** Copyright (C) 1994-2012 Lua.org, PUC-Rio. All rights

reserved. * * Permission is hereby granted, free of charge, to any person obtaining * a copy of this software and associated documentation files (the * "Software"), to deal in the Software without restriction, including * without limitation the rights to use, copy, modify, merge, publish, * distribute, sublicense, and/or sell copies of the Software, and to * permit persons to whom the Software is furnished to do so, subject to * the following conditions: * * The above copyright notice and this permission notice shall be * included in all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

53700. active function

53701. ** Event codes

53702. private part

53703. **comment:** hack

label: code-design

53704. Functions to be called by the debugger in specific events

53705. type of numbers in Lua

53706. type for integer functions

53707. ** get functions (Lua -> stack)

53708. ** pseudo-indices

53709. (u) number of upvalues

53710. (S) 'Lua', 'C', 'main', 'tail'

53711. ** coroutine functions

53712. **comment:** ** generic extra include file

label: code-design

53713. }=====

53714. ** Event masks

53715. ** miscellaneous functions

53716. ** basic stack manipulation

53717. (n)

53718. activation record

53719. ** {===== ** Debug API ** =====}

=====

53720. mark for precompiled code ('<esc>Lua')

53721. ** compatibility macros and functions

53722. end of options; keep it

53723. list

53724. default program name

53725. default output file

53726. ** \$Id: luac.c,v 1.54 2006/06/02 17:37:11 lhf Exp \$ ** Lua compiler (saves bytecodes to files; also list bytecodes) ** See Copyright Notice in lua.h

53727. actual output file name

53728. output file

53729. show version

53730. unknown option

53731. parse only

53732. strip debug information

53733. end of options; use stdin

53734. list bytecodes?

53735. actual program name

53736. dump bytecodes?

53737. default output file name

53738. strip debug information?

53739. end of options; skip it

53740. @@ LUA_API is a mark for all core API functions. @@ LUALIB_API is a mark for all standard library functions. ** CHANGE them if you need to define those functions in some special way. ** For instance, if you want to create one Windows DLL with the core and ** the libraries, you may want to use the following definition (define ** LUA_BUILD_AS_DLL to get it).

53741. @@ LUA_PATH and LUA_CPATH are the names of the environment variables that @* Lua check to set its paths. @@ LUA_INIT is the name of the environment variable that Lua @* checks for initialization code. ** CHANGE them if you want different names.

53742. @@ luaL_userstate* allow user-specific actions on threads. ** CHANGE them if you defined LUAI_EXTRASPACE and need to do something ** extra when a thread is created/deleted/resumed/yielded.

53743. **comment:** @@ LUAI_MAXCALLS limits the number of nested calls. ** CHANGE it if you need really deep recursive calls. This limit is ** arbitrary; its only purpose is to stop infinite recursion before ** exhausting memory.

label: code-design

53744. }=====

53745. 16 digits, sign, point, and \0

53746. get line
53747. @@ LUA_INTFRMLEN is the length modifier for integer conversions @* in 'string.format'. @@ LUA_INTFRM_T is the integer type corresponding to the previous length @* modifier. ** CHANGE them if your system supports long long or does not support long.
53748. @@ LUA_COMPAT_OPENLIB controls compatibility with old 'luaL_openlib' @* behavior. ** CHANGE it to undefined as soon as you replace to 'luaL_register'
** your uses of 'luaL_openlib'
53749. @@ LUA_ANSI controls the use of non-ansi features. ** CHANGE it (define it) if you want Lua to avoid the use of any ** non-ansi feature or library.
53750. non-empty line?
53751. **comment:** @@ LUAI_USER_ALIGNMENT_T is a type that requires maximum alignment. ** CHANGE it if your system requires alignments larger than double. (For ** instance, if your system supports long doubles and they must be ** aligned in 16-byte boundaries, then you should add long double in the ** union.) Probably you do not need to change this.
label: code-design
53752. =====
53753. ** In Windows, any exclamation mark (!) in the path is replaced by the ** path of the directory of the executable file of the current process.
53754. int has at least 32 bits
53755. @@ LUA_COMPAT_GETN controls compatibility with old getn behavior. ** CHANGE it (define it) if you want exact compatibility with the ** behavior of setn/getn in Lua 5.0.
53756. **comment:** needs an extra library: -ldl
label: code-design
53757. @@ LUA_BUFFER_SIZE is the buffer size used by the lauxlib buffer system.
53758. ** ===== ** Search for "@@" to find all configurable definitions. **
=====
53759. @@ LUA_GCMUL defines the default speed of garbage collection relative to @* memory allocation as a percentage. ** CHANGE it if you want to change the granularity of the garbage ** collection. (Higher values mean coarser collections. 0 represents ** infinity, where each step performs a full collection.) You can also ** change this value dynamically.
53760. @@ LUA_IDSIZE gives the maximum size for the description of the source @* of a function in debug information. ** CHANGE it if you want a different size.
53761. @@ LUAI_EXTRASPACE allows you to add user-specific data in a lua_State @* (the data goes just *before* the lua_State pointer). ** CHANGE (define) this if you really need that. This value must be ** a multiple of the maximum alignment required for your machine.
53762. @@ LUAI_THROW/LUAI_TRY define how Lua does exception handling. ** CHANGE them if you prefer to use longjmp/setjmp even with C++ ** or if want/don't to use _longjmp/_setjmp instead of regular ** longjmp/setjmp. By default, Lua handles errors with exceptions when ** compiling as C++ code, with _longjmp/_setjmp when asked to use them, ** and with longjmp/setjmp otherwise.
53763. **comment:** @@ lua_number2int is a macro to convert lua_Number to int. @@ lua_number2integer is a macro to convert lua_Number to lua_Integer. ** CHANGE them if you know a faster way to convert a lua_Number to ** int (with any rounding method and without throwing errors) in your ** system. In Pentium machines, a naive typecast from double to int ** in C is extremely slow, so any alternative is worth trying.
label: code-design
53764. @@ LUA_MAXVARS is the maximum number of local variables per function @* (must be smaller than 250).
53765. @@ The luaL_num* macros define the primitive operations over numbers.
53766. @@ LUAI_FUNC is a mark for all extern functions that are not to be @* exported to outside modules. @@ LUAI_DATA is a mark for all extern (const) variables that are not to @* be exported to outside modules. ** CHANGE them if you need to mark them in some special way. Elf/gcc ** (versions 3.2 and later) mark them as "hidden" to optimize access ** when Lua is compiled as a shared library.
53767. ** \$Id: luacnf.h,v 1.82.1.7 2008/02/11 16:25:08 roberto Exp \$ ** Configuration file for Lua ** See Copyright Notice in lua.h
53768. avoid overflows in comparison
53769. dummy variable
53770. @@ LUA_USE_POSIX includes all functionality listed as X/Open System @* Interfaces Extension (XSI). ** CHANGE it (define it) if your system is XSI compatible.
53771. 200% (wait memory to double before next GC)
53772. C++ exceptions
53773. more often than not the libs go together with the core
53774. assume stdin is a tty
53775. @@ LUA_MAXINPUT is the maximum length for an input line in the @* stand-alone interpreter. ** CHANGE it if you need longer lines.
53776. @@ lua_readline defines how to show a prompt and then read a line from @* the standard input. @@ lua_saveline defines how to "save" a read line in a "history". @@ lua_freefile defines how to free a line read by lua_readline. ** CHANGE them if you want to improve this functionality (e.g., by using ** GNU readline and history facilities).
53777. @@ LUA_PATHSEP is the character that separates templates in a path. @@ LUA_PATH_MARK is the string that marks the substitution points in a @* template. @@ LUA_EXECDIR in a Windows path is replaced by the executable's @* directory. @@ LUA_IGMARK is a mark to ignore all before it when building the @* luaopen_ function name. ** CHANGE them if for some reason your system cannot use those ** characters. (E.g., if one of those characters is a common character ** in file/directory names.) Probably you do not need to change them.
53778. @@ LUA_MAXCAPTURES is the maximum number of captures that a pattern @* can do during pattern-matching. ** CHANGE it if you need more captures. This limit is arbitrary.
53779. @@ LUA_UINT32 is an unsigned integer with at least 32 bits. @@ LUA_INT32 is a signed integer with at least 32 bits. @@ LUA_UMEM is an unsigned integer big enough to count the total @* memory used by Lua. @@ LUA_MEM is a signed integer big enough to count the total memory @* used by Lua. ** CHANGE here if for some weird reason the default definitions are not ** good enough for your machine. (The definitions in the 'else' ** part always works, but may waste space on machines with 64-bit ** longs.) Probably you do not need to change this.
53780. ** Local configuration. You can use this space to add your redefinitions ** without modifying the main part of the file.
53781. @@ LUA_COMPAT_MOD controls compatibility with old math.mod function. ** CHANGE it to undefined as soon as your programs use 'math.fmod' or ** the new '%' operator instead of 'math.mod'.
53782. show prompt
53783. @@ LUA_DIRSEP is the directory separator (for submodules). ** CHANGE it if your machine does not use "/" as the directory separator ** and is not Windows. (On Windows Lua automatically uses "\").
53784. @@ LUA_PATH_DEFAULT is the default path that Lua uses to look for @* Lua libraries. @@ LUA_CPATH_DEFAULT is the default path that Lua uses to look for @* C libraries. ** CHANGE them if your machine has a non-conventional directory ** hierarchy or if you want to install your libraries in ** non-conventional directories.
53785. @@ LUA_MAXCCALLS is the maximum depth for nested C calls (short) and @* syntactical nested non-terminals in a program.
53786. @@ LUA_COMPAT_VARARG controls compatibility with old vararg feature. ** CHANGE it to undefined as soon as your programs use only '...' to ** access vararg parameters (instead of the old 'arg' table).
53787. ** ===== ** Stand-alone configuration **
=====
53788. On a Pentium, resort to a trick
53789. add it to history
53790. @@ LUA_PROGNAME is the default name for the stand-alone Lua program. ** CHANGE it if your stand-alone interpreter has a different name and ** your system is not able to detect that name automatically.
53791. 16-bit ints
53792. @@ LUA_UACNUMBER is the result of an 'usual argument conversion' @* over a number.
53793. @@ LUA_BITSINT defines the number of bits in an int. ** CHANGE here if Lua cannot automatically detect the number of bits of ** your machine. Probably you do not need to change this.
53794. @@ luaL_stdin_is_tty detects whether the standard input is a 'tty' (that @* is, whether we're running lua interactively). ** CHANGE it if you have a better definition for non-POSIX/non-Windows ** systems.
53795. the next trick should work on any Pentium, but sometimes clashes with a DirectX idiosyncrasy

53796. @@ LUA_INTEGER is the integral type used by lua_pushinteger/lua_tointeger. ** CHANGE that if ptrdiff_t is not adequate on your machine. (On most ** machines, ptrdiff_t gives a good choice between int or long.)

53797. @@ lua_popen spawns a new process connected to the current one through @* the file streams. ** CHANGE it if you have a way to implement it in your system.

53798. @@ LUA_COMPAT_LOADLIB controls compatibility about global loadlib. ** CHANGE it to undefined as soon as you do not need a global 'loadlib' ** function (the function is still available as 'package.loadlib').

53799. ** {===== ** CHANGE (to smaller values) the following definitions if your system ** has a small C stack. (Or you may want to change them to larger ** values if your system has a large C stack and these limits are ** too rigid for you.) Some of these constants control the size of ** stack-allocated arrays used by the compiler or the interpreter, while ** others limit the maximum number of recursive calls that the compiler ** or the interpreter can perform. Values too large may cause a C stack ** overflow for some forms of deep constructs. ** ======

53800. ** {===== ** LUA_NUMBER is the type of numbers in Lua. ** CHANGE the following definitions only if you want to build Lua ** with a number type different from double. You may also need to ** change lua_number2int & lua_number2integer. ** ======

53801. **comment:** in Unix, try _longjmp/_setjmp (more efficient)
label: code-design

53802. **comment:** @@ luai_apicheck is the assert macro used by the Lua-C API. ** CHANGE luai_apicheck if you want Lua to perform some checks in the ** parameters it gets from API calls. This may slow down the interpreter ** a bit, but may be quite useful when debugging C code that interfaces ** with Lua. A useful redefinition is to use assert.h.
label: code-design

53803. @@ LUAI_MAXCSTACK limits the number of Lua stack slots that a C function @* can use. ** CHANGE it if you need lots of (Lua) stack space for your C ** functions. This limit is arbitrary; its only purpose is to stop C ** functions to consume unlimited stack space. (must be smaller than ** -LUA_REGISTRYINDEX)

53804. @@ LUA_QL describes how error messages quote program elements. ** CHANGE it if you want a different appearance.

53805. @@ LUA_COMPAT_LSTR controls compatibility with old long string nesting @* facility. ** CHANGE it to 2 if you want the old behaviour, or undefine it to turn ** off the advisory error when nesting [[...]].

53806. @@ LUAI_GCPAUSE defines the default pause between garbage-collector cycles @* as a percentage. ** CHANGE it if you want the GC to run faster or slower (higher values ** mean larger pauses which mean slower collection.) You can also change ** this value dynamically.

53807. @@ LUAI_MAXUPVALUES is the maximum number of upvalues per function @* (must be smaller than 250).

53808. @@ LUA_NUMBER_SCAN is the format for reading numbers. @@ LUA_NUMBER_FMT is the format for writing numbers. @@ lua_number2str converts a number to a string. @@ LUAI_MAXNUMBER2STR is maximum size of previous conversion. @@ lua_str2number converts a string to a number.

53809. On a Microsoft compiler, use assembler

53810. **comment:** this option always works, but may be slow
label: code-design

53811. default handling with long jumps

53812. GC runs 'twice the speed' of memory allocation

53813. @@ lua_tmpnam is the function that the OS library uses to create a @* temporary name. @@ LUA_TMPNAMBUFSIZE is the maximum size of a name created by lua_tmpnam. ** CHANGE them if you have an alternative to tmpnam (which is considered ** insecure) or if you want the original tmpnam anyway. By default, Lua ** uses tmpnam except when POSIX is available, where it uses mkstemp.

53814. @@ LUA_PROMPT is the default prompt used by stand-alone Lua. @@ LUA_PROMPT2 is the default continuation prompt used by stand-alone Lua. ** CHANGE them if you want different prompts. (You can also change the ** prompts dynamically, assigning to globals _PROMPT/_PROMPT2.)

53815. @@ LUA_COMPAT_GFIND controls compatibility with old 'string.gfind' name. ** CHANGE it to undefined as soon as you rename 'string.gfind' to ** 'string.gmatch'.

53816. **comment:** needs some extra libraries
label: code-design

53817. @@ LUA_DL_* define which dynamic-library system Lua should use. ** CHANGE here if Lua has problems choosing the appropriate ** dynamic-library system for your platform (either Windows' DLL, Mac's ** dyld, or Unix's dlopen). If your system is some kind of Unix, there ** is a good chance that it has dlopen, so LUA_DL_DLOPEN will work for ** it. To use dlopen you also need to adapt the src/Makefile (probably ** adding -ldl to the linker options), so Lua does not select it ** automatically. (When you change the makefile to add -lldl, you must ** also add -DLUA_USE_DLOPEN.) ** If you do not want any kind of dynamic library, undefine all these ** options. ** By default, _WIN32 gets LUA_DL_DLL and MAC OS X gets LUA_DL_DYLD.

53818. empty

53819. **comment:** does not need extra library
label: code-design

53820. ** \$Id: lualib.h,v 1.36.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Lua standard libraries ** See Copyright Notice in lua.h

53821. open all previous libraries

53822. Key to file-handle type

53823. is lua_Number integral?

53824. ** load precompiled chunk

53825. ** \$Id: lundump.c,v 2.7.1.4 2008/04/04 19:51:41 roberto Exp \$ ** load precompiled Lua chunks ** See Copyright Notice in lua.h

53826. * make header

53827. remove trailing '\0'

53828. endianness

53829. for header of binary files -- this is Lua 5.1

53830. make header; from lundump.c

53831. print one chunk; from print.c

53832. dump one chunk; from ldump.c

53833. load one chunk; from lundump.c

53834. for header of binary files -- this is the official format

53835. size of header of binary files

53836. ** \$Id: lundump.h,v 1.37.1.1 2007/12/27 13:02:25 roberto Exp \$ ** load precompiled Lua chunks ** See Copyright Notice in lua.h

53837. first try 'le'

53838. true must be 1 !!

53839. remove new frame

53840. try first operand

53841. yes: continue its execution

53842. restart luaV_execute over new Lua function

53843. it was a C function ('precall' called it)

53844. update internal index...

53845. avoid pointing inside table (may rehash)

53846. previous frame

53847. call TM

53848. tail call: put new frame in place of previous one

53849. will try TM

53850. repeat until only 1 result left

53851. else repeat with 'tm'

53852. same metatables => same metamethods

53853. both strings longer than 'len'; go on comparing (after the '\0')

53854. previous function index

53855. same metamethods?

53856. increment index

53857. collect total length
53858. warning!! several calls may realloc the stack and invalidate `ra'
53859. pre-alloc it at once
53860. or no TM?
53861. ** \$Id: lvm.c,v 2.63.1.5 2011/08/17 20:43:11 roberto Exp \$ ** Lua virtual machine ** See Copyright Notice in lua.h
53862. second op is empty?
53863. 3th argument
53864. l is finished?
53865. no: return
53866. no metamethod
53867. push function
53868. l is smaller than r (because r is not finished)
53869. try second operand
53870. next steps may throw errors
53871. **comment:** needs more space?
 label: code-design
53872. continue loop?
53873. was previous function running `here'?
53874. limit for table tag-method chains (to avoid loops)
53875. entry point
53876. correct top
53877. 't' is a table?
53878. move frame down
53879. result is no nil?
53880. did hook yield?
53881. to be used after possible stack reallocation
53882. func. + 2 args (state and index)
53883. jump back
53884. strings are equal up to a '\0'
53885. r is finished?
53886. concat all strings
53887. at least two string values; get as many as possible
53888. ...and external index
53889. try metamethod
53890. one more call lost
53891. got 'n' strings to create 1 new
53892. ** some macros for common tasks in `luaV_execute'
53893. call base
53894. no TM?
53895. skip next instruction (if C)
53896. yield
53897. result is first op (as string)
53898. save control variable
53899. no metamethod?
53900. else try `lt'
53901. main loop of interpreter
53902. call linehook when enter a new function, when jump back (loop), or when enter a new line
53903. do a primitive get
53904. index of first '\0' in both strings
53905. different metamethods?
53906. else previous instruction set top
53907. it was a C function ('precall' called it); adjust results
53908. else will try the tag method
53909. 2nd argument
53910. do a primitive set
53911. 1st argument
53912. number of elements handled in this pass (at least 2)
53913. previous call may change the stack
53914. next assignment may change this value
53915. ** \$Id: lvm.h,v 2.5.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Lua virtual machine ** See Copyright Notice in lua.h
53916. return number of missing bytes
53917. ** \$Id: lzio.c,v 1.31.1.1 2007/12/27 13:02:25 roberto Exp \$ ** a generic input stream interface ** See Copyright Notice in lua.h
53918. min. between n and z->n
53919. -----
53920. ----- read ---
53921. luaZ_fill removed first byte; put back it
53922. end of stream
53923. current position in buffer
53924. Lua state (for reader)
53925. ** \$Id: lzio.h,v 1.21.1.1 2007/12/27 13:02:25 roberto Exp \$ ** Buffered streams ** See Copyright Notice in lua.h
53926. ----- Private Part -----
53927. additional data
53928. bytes still unread
53929. read next n bytes
53930. cannot happen
53931. ** \$Id: print.c,v 1.55a 2006/05/31 13:30:05 lhf Exp \$ ** print bytecodes ** See Copyright Notice in lua.h
53932. Last change: revised for Lua 5.2.4
53933. (S)
53934. (t)
53935. Last change: revised for Lua 5.2.4
53936. ======
53937. (l)
53938. private part
53939. \$Id: manual.of,v 1.104 2013/06/01 00:13:11 roberto Exp \$
53940. (u) number of parameters
53941. (n)
53942. (u)
53943. (u) number of upvalues

53944. get global 'f'
53945. Last change: revised for Lua 5.2.4
53946. no errors?
53947. it has no upvalues
53948. function to be called
53949. not a closure
53950. save continuation
53951. save context
53952. adjust frame top
53953. signal it
53954. stack large enough?
53955. save information for error recovery
53956. do a 'conventional' protected call
53957. ** get functions (Lua -> stack)
53958. no continuation or no yieldable?
53959. for unary minus, add fake 2nd operand
53960. test for pseudo index
53961. set global table as 1st upvalue of 'f' (may be LUA_ENV)
53962. include current debt
53963. yes; check is OK
53964. push empty string
53965. end of cycle?
53966. GC values are expressed in Kbytes: #bytes/2^10
53967. may call tag method
53968. generational mode?
53969. does it have one upvalue?
53970. C closure
53971. ** access functions (stack -> C)
53972. ** 'load' and 'call' functions (run Lua code)
53973. ** set functions (stack -> Lua)
53974. upvalues
53975. Lua closure
53976. not a C function
53977. 1st operand
53978. mark that function may do error recovery
53979. get global table from registry
53980. no
53981. 2nd operand
53982. light C function?
53983. else n == 1; nothing to do
53984. remove key
53985. use L->top as a temporary
53986. data to 'f_call'
53987. pop value and key
53988. **comment:** to avoid warnings
 label: code-design
53989. just do the call
53990. change collector to incremental mode
53991. ** push functions (C -> stack)
53992. ensure that true is 1
53993. conversion failed?
53994. global table
53995. need to prepare continuation?
53996. previous call may reallocate the stack
53997. corresponding test
53998. pop index and value
53999. 'subtract' index (index is negative)
54000. all other operations expect two operands
54001. can grow without overflow?
54002. ** miscellaneous functions
54003. ** to be called by 'lua_checkstack' in protected mode, to grow stack ** capturing memory errors
54004. value at a non-valid index
54005. test for valid but not pseudo index
54006. code unreachable; will unlock when control actually leaves the kernel
54007. change collector to generational mode
54008. do a single step
54009. ** convert an acceptable stack index into an absolute index
54010. if it is here, there were no errors
54011. get newly created function
54012. invalid option
54013. prepare continuation (call is already protected by 'resume')
54014. negative index
54015. do the call
54016. no continuation or no yieldable
54017. get its upvalue pointer
54018. true if it will do major collection
54019. no; need to grow stack
54020. ** Garbage-collection function
54021. 'luaV_tostring' may create a new string
54022. ** Execute a protected call.
54023. ** \$Id: lapi.c,v 2.171.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua API ** See Copyright Notice in lua.h
54024. try to grow stack
54025. lua closure
54026. no more elements
54027. ** basic stack manipulation
54028. LUA_REGISTRYINDEX does not need gc barrier (collector revisits it before finishing collection)
54029. function upvalue?
54030. ** \$Id: lapi.h,v 2.7.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Auxiliary functions from Lua API ** See Copyright Notice in lua.h

54031. ** use appropriate macros to interpret 'pclose' return status
54032. name already in use?
54033. double buffer size
54034. remove previous table
54035. remove table (but keep name)
54036. no more pre-read characters
54037. get _LOADED[modname]
54038. copy to be left at top
54039. remove pushed values
54040. false, because did not find table there
54041. remove both metatables
54042. get/create library table
54043. not big enough?
54044. not found
54045. **comment:** keep some extra space to run error routines, if needed
 label: code-design
54046. push prefix
54047. get a new reference
54048. table already there
54049. read initial portion
54050. return true/nil,what,code
54051. remove table and value
54052. type of termination
54053. add line to correct line numbers
54054. no metafield?
54055. remove it from list
54056. remove this nil
54057. no stack frame?
54058. argument to open function
54059. {
54060. check conversions number -> integer types
54061. remove only metatable
54062. (t[freelist] = t[ref])
54063. binary file?
54064. }=====

54065. ** reads the first character of file 'f' and skips an optional BOM mark ** in its beginning plus its first line if it starts with '#'. Returns ** true if it skipped the first line. In any case, 'cp' has the ** first "valid" character of the file (after the optional BOM and ** a first-line comment).
54066. push last suffix
54067. re-read initial portion
54068. ** { ===== ** Userdata's metatable manipulation **
=====

54069. start 'next' loop
54070. no metatable?
54071. ** { ===== ** Reference system **
=====

54072. copy upvalues to the top
54073. **comment:** not used
 label: code-design
54074. ** \$Id: lauxlib.c,v 1.248.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Auxiliary functions for building Lua libraries ** See Copyright Notice in lua.h
54075. This file uses only the official API of Lua. ** Any function declared here could be written as an application function.
54076. move name to proper place
54077. remove it from stack
54078. 'fread' can return > 0 *and* set the EOF flag. If next call to 'getF' called 'fread', it might still wait for user input. The next check avoids this problem.
54079. find an upper bound
54080. read a block from file
54081. **comment:** ** check whether buffer is using a userdata on the stack as a temporary ** buffer
 label: code-design
54082. move library table to below upvalues
54083. get first free element
54084. remove from stack
54085. to be read by the parser
54086. return problematic part of the name
54087. ** returns a pointer to a free area with at least 'sz' bytes
54088. reopen in binary mode
54089. remove value (but keep name)
54090. put value below buffer
54091. push function
54092. do not count `self'
54093. ** set functions from list 'l' into table at top - 'nup'; each ** function gets the 'nup' elements at the top as upvalues. ** Returns with only the table at the stack.
54094. successful termination?
54095. get info about it
54096. leave previous value on top, but return 0
54097. no free elements
54098. create metatable
54099. for each pair in table
54100. no comment
54101. index of filename on the stack
54102. skip end-of-line, if present
54103. ref = t[freelist]
54104. size of the second part of the stack
54105. value is a userdata?
54106. Utf8 BOM mark
54107. there was a comment
54108. index of free-list header
54109. continue after 'p'
54110. file being read
54111. field has a non-table value?
54112. copy of 'mod'

54113. number of pre-read characters
54114. ** search for 'objidx' in table at index -1. ** return 1 + string at top if find a good name.
54115. are there pre-read characters to be read?
54116. t[ref] = t[freelist]
54117. main?
54118. return next character
54119. 'c' is the first character of the stream
54120. set new table into field
54121. _LOADED[modname] = module
54122. return to Lua to abort
54123. remove _LOADED table
54124. assign new table to field
54125. check function at level
54126. calls to Lua API may change this value
54127. create larger buffer
54128. prefix matched; discard it
54129. try recursively
54130. add a '...'
54131. new table for field
54132. {freelist} = ref
54133. push replacement in place of pattern
54134. try to get metatable
54135. _G[modname] = module
54136. ** {===== ** Argument check functions **
=====**
54137. ignore results from 'lua_load'
54138. any free element?
54139. ** {===== ** Traceback **
=====**
54140. does it have a metatable?
54141. remove function and global table
54142. try global variable (and create one if it does not exist)
54143. place '.' between the two names
54144. do a binary search
54145. ** {===== ** Generic Buffer manipulation **
=====**
54146. ** stripped-down 'require'. Calls 'openf' to open a module, ** registers the result in 'package.loaded' table and, if 'glb' ** is true, also registers the result in the global table. ** Leaves resulting module on the top.
54147. remove object
54148. first line is a comment (Unix exec. file)?
54149. registry.name = metatable
54150. ** {===== ** Load functions **
=====**
54151. read block
54152. **comment**: too many levels?
 label: code-design
54153. interpret result
54154. value is a userdata with wrong metatable
54155. skip first line
54156. remove upvalues
54157. _LOADED[modname] = new table
54158. found object?
54159. value is not a userdata with a metatable
54160. remove metatable and metafield
54161. not enough space?
54162. ignore non-string keys
54163. and skip to last ones
54164. remove previous result
54165. ** Count number of elements in a luaL_Reg list.
54166. close file (even in case of errors)
54167. get correct metatable
54168. ** ensure that stack[idx][fname] has a table and push that table ** into the stack
54169. return them (chars already in buffer)
54170. ** {===== ** Error-report functions **
=====**
54171. }
54172. ** {===== ** Compatibility with 5.1 module functions **
=====**
54173. remove value
54174. open module
54175. else, no information available...
54176. make copy of module (call result)
54177. size of the first part of the stack
54178. area for reading file
54179. get _LOADED table
54180. error is in the self argument itself?
54181. not the same?
54182. closure with those upvalues
54183. error?
54184. is there a name?
54185. 'nil' has a unique fixed reference
54186. not found?
54187. remove old buffer
54188. no such field?
54189. ** Find or create a module table with a given name. The function ** first looks at the _LOADED table and, if that fails, try a ** global variable with that name. In any case, leaves on the stack ** the module table.
54190. is there info?
54191. remove name

54192. move content to new buffer
54193. fill the table with given functions
54194. no op
54195. number of characters in buffer
54196. extra error code for 'luaL_load'
54197. compatibility with old module system
54198. buffer size
54199. ** \$Id: lauxlib.h,v 1.120.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Auxiliary functions for building Lua libraries ** See Copyright Notice in lua.h
54200. }=====** {===== ** File handles for IO library **
54201. ===== ** {===== ** File handles for IO library **
54202. ===== A file handle is a userdata with metatable 'LUA_FILEHANDLE' and ** initial structure 'luaL_Stream' (it may contain other fields ** after that initial structure).
54203. stream (NULL for incompletely created streams)
54204. pre-defined references
54205. initial buffer
54206. to close stream (NULL for closed streams)
54207. ** {===== ** Generic Buffer manipulation **
54208. buffer address
54209. ** ===== ** some useful macros **
54210. create reserved slot
54211. function to be called
54212. 'env' parameter?
54213. ** Reader for generic `load` function: 'lua_load' uses the ** stack for internal stuff, so the reader cannot change the ** stack top. Instead, it keeps its resulting string in a ** reserved slot inside the stack.
54214. set global _VERSION
54215. handle signal
54216. get 3 values from metamethod
54217. and initial value
54218. invalid numeral; force a fail
54219. argument must be a table
54220. ** \$Id: lbaselib.c,v 1.276.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Basic library ** See Copyright Notice in lua.h
54221. no invalid trailing characters?
54222. standard conversion
54223. no metamethod?
54224. set it as 1st upvalue
54225. loading from a reader function
54226. end point for 's'
54227. no space for extra boolean?
54228. skip initial spaces
54229. exchange function...
54230. ** reserved slot, above all arguments, to hold a copy of the returned ** string to avoid it being collected while parsed. 'load' has four ** optional arguments (chunk, source name, mode, and environment).
54231. skip trailing spaces
54232. loading a string?
54233. no metatable
54234. 'env' index or 0 if no 'env'
54235. return nil plus error message
54236. return false, msg
54237. not a number
54238. error (message is on top of the stack)
54239. number of arguments
54240. **comment:** remove 'env' if not used by previous call
 label: code-design
54241. open lib into global table
54242. state,
54243. next value
54244. create space for status result
54245. argument 'self' to metamethod
54246. environment for loaded function
54247. save string in reserved slot
54248. returns either __metatable field (if present) or metatable
54249. }=====** {===== ** Generic Read function **
54250. set global _G
54251. value to print
54252. ...and error handler
54253. put first result in first slot
54254. else not a number; must be something
54255. create space for return values
54256. first result (status)
54257. add extra information?
54258. create a 2nd argument if there isn't one
54259. else not a number
54260. get result
54261. put before error message
54262. ** {===== ** Generic Read function **
54263. **comment:** not used
 label: code-design
54264. get function
54265. call it
54266. pop result
54267. will return generator,
54268. number of bits to consider in a number
54269. ** \$Id: lbitlib.c,v 1.18.1.2 2013/07/09 18:01:41 roberto Exp \$ ** Standard library for bitwise operations ** See Copyright Notice in lua.h
54270. builds a number with 'n' ones (1 <= n <= LUA_NBITS)
54271. shift right?

54272. arithmetic shift for 'negative' number
54273. **comment:** ** get field and width arguments for field-manipulation functions, ** checking whether they are valid. ** ('luaL_error' called without 'return' to avoid later warnings about ** 'width' being used uninitialized.)
 label: code-design
54274. add signal bit
54275. macro to trim extra bits
54276. erase bits outside given width
54277. avoid undefined shift of LUA_NBITS when i == 0
54278. shift left
54279. i = i % NBITS
54280. position after whole expression
54281. operand must be on the 'stack'
54282. operand must be on the 'stack'
54283. insert last jump in `f list
54284. find last element
54285. free registers with list values
54286. invalid var kind to store
54287. jump to default target
54288. argument is +1 to reserve 0 as non-op
54289. put this jump in `t list
54290. else may be a collision (e.g., between 0.0 and "\0\0\0\0\0\0\0\0"); go through and create a new entry for this value
54291. base register for op_self
54292. insert last jump in `t list
54293. else go through
54294. exp is already in a register
54295. end of list
54296. l = max(l, pl)
54297. turn offset into absolute position
54298. always false; do nothing
54299. cannot happen
54300. fold it
54301. not found
54302. remove previous OP_NOT
54303. put new instruction in code array
54304. 't' is in a register?
54305. constant fits in RK operand?
54306. interchange true and false lists
54307. regular case
54308. constant not found; create a new entry
54309. nothing to do...
54310. from = min(from, pfrom)
54311. cannot patch other instructions
54312. those instructions may be jump targets
54313. handle -0 and NaN
54314. position of an eventual LOAD true
54315. o1 <==> o2
54316. there is one value available (somewhere)
54317. cannot use nil as key; instead use table itself to represent nil
54318. go through
54319. point to itself represents end of list
54320. do not attempt to divide by 0
54321. last register to set nil
54322. reg. is not a local?
54323. default
54324. no jumps to current position?
54325. ** returns current `pc` and marks it as a jump target (to avoid wrong ** optimizations with consecutive instructions not in the same basic block).
54326. save corresponding line information
54327. numerical value does not need GC barrier; table has no metatable, so it does not need to invalidate cache
54328. constant fits in argC?
54329. exchange args to replace by `<` or `<=`
54330. expression is an open function call?
54331. assume 't' is in an upvalue
54332. else no optimization
54333. ** check whether list has any jump that do not produce a value ** (or produce an inverted value)
54334. position of an eventual LOAD false
54335. keep them on hold
54336. not a constant in the right range: put it in a register
54337. save list of jumps to here
54338. always true; do nothing
54339. ** \$Id: lcode.c,v 2.62.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Code generator for Lua ** See Copyright Notice in lua.h
54340. `pc` will change
54341. register where 'e' was placed
54342. use raw representation as key to avoid numeric problems
54343. function and 'self' produced by op_self
54344. can connect both?
54345. no register to put value or register already has the value
54346. put value on it
54347. can relocate its simple result
54348. pc of last jump
54349. minus constant?
54350. list must be closed
54351. cannot operate on constants
54352. ** grep "ORDER OPR" if you change these enums (ORDER OP)
54353. ** Marks the end of a patch list. It is an invalid value both as an absolute ** address, and as a list link (would link an element to itself).
54354. ** \$Id: lcode.h,v 1.58.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Code generator for Lua ** See Copyright Notice in lua.h
54355. does it have frames?
54356. error flag
54357. return true + `resume` returns

54358. move function from L to NL
54359. initial state
54360. ** \$Id: lcorolib.c,v 1.5.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Coroutine Library ** See Copyright Notice in lua.h
54361. it is running
54362. propagate error
54363. some error occurred
54364. error object is a string?
54365. move error message
54366. remove results anyway
54367. move function to top
54368. return false + error message
54369. add extra info
54370. move yielded values
54371. EOZ
54372. 7.
54373. 1.
54374. 6.
54375. }
54376. 3.
54377. d.
54378. 0.
54379. a.
54380. b.
54381. 9.
54382. e.
54383. {
54384. ** \$Id: lctype.c,v 1.11.1.1 2013/04/12 18:48:47 roberto Exp \$ ** 'ctype' functions for Lua ** See Copyright Notice in lua.h
54385. 5.
54386. 2.
54387. c.
54388. f.
54389. 4.
54390. 8.
54391. **comment:** ASCII case: can use its own tables; faster and fixed
 label: code-design
54392. ** use standard C ctypes
54393. must use standard C ctype
54394. {
54395. two more entries for 0 and -1 (EOZ)
54396. ** \$Id: lctype.h,v 1.12.1.1 2013/04/12 18:48:47 roberto Exp \$ ** 'ctype' functions for Lua ** See Copyright Notice in lua.h
54397. ** WARNING: the functions defined here do not necessarily correspond ** to the similar functions in the standard C ctype.h. They are ** optimized for the specific needs of Lua
54398. }{
54399. }
54400. ** add 1 to char to allow index -1 (EOZ)
54401. ** this 'tolower' only works for alphabetic characters
54402. ** 'lalpha' (Lua alphabetic) and 'alnum' (Lua alphanumeric) both include '_'
54403. ** \$Id: ldblib.c,v 1.132.1.2 2015/02/19 17:16:55 roberto Exp \$ ** Interface from Lua to its debug API ** See Copyright Notice in lua.h
54404. setmetatable(hooktable) = hooktable
54405. non-string 'msg'?
54406. set hooks
54407. push name
54408. external hook?
54409. set new hook
54410. out of range?
54411. return 1st argument
54412. return it untouched
54413. creating hook table?
54414. push local name
54415. push function
54416. level out of range
54417. turn off hooks
54418. no metatable
54419. remove eventual returns
54420. function argument?
54421. return table
54422. re-order
54423. remove hook table
54424. no name (nor value)
54425. push local value
54426. stack-level argument
54427. local-variable index
54428. * hooktable.__mode = "k"
54429. get hook
54430. cannot know who sets that register
54431. active function; get information through 'ar'
54432. calling instruction index
54433. table index
54434. move from 'b' to 'a'
54435. pop value
54436. not found
54437. no source available; use "?" instead
54438. update 'jmptarget'
54439. for iterator
54440. 'name' already filled
54441. is a local?
54442. ** \$Id: ldebug.c,v 2.90.1.4 2015/02/19 17:05:13 roberto Exp \$ ** Debug Interface ** See Copyright Notice in lua.h
54443. get name for 'b'

54444. jumped code can change 'a'
54445. =====
54446. no reasonable name found
54447. first operand is wrong
54448. generic name for any valid slot
54449. ** find a "name" for the RK value 'c'
54450. affect all registers above base
54451. check whether 'o' is an upvalue
54452. add file:line information
54453. no 'standard' name?
54454. ===== ** Symbolic Execution **
=====

54455. current position sets that register
54456. information about non-active function?
54457. push function
54458. 'c' is a register
54459. name of indexed variable
54460. turn off hooks?
54461. key index
54462. **comment:** to avoid warnings
 label: code-design
54463. affect all regs above its base
54464. no such level
54465. calling instruction
54466. generic name for any vararg
54467. not a Lua function?
54468. level found?
54469. no name
54470. **comment:** ** only ANSI way to check whether a pointer points to an array ** (used only for error messages, so efficiency is not a big concern)
 label: code-design
54471. for all lines with code
54472. is Lua code?
54473. else no useful name can be found
54474. skip the '>'
54475. calling function
54476. literal constant?
54477. all other instructions can call only through metamethods
54478. any code before this address is conditional
54479. search for 'c'
54480. get function name
54481. pop function
54482. boolean 'true' to be the value of all indices
54483. else no reasonable name found
54484. move argument
54485. set registers from 'a' to 'a+b'
54486. no? try a register
54487. access to vararg values?
54488. go through to return NULL
54489. new table to store active lines
54490. handled by lua_getinfo
54491. ** try to find last instruction before 'lastpc' that modified register 'reg'
54492. invalid (negative) level
54493. call it
54494. could find instruction?
54495. no such vararg
54496. jump is forward and do not skip 'lastpc'?
54497. consider live variables at function start (parameters)
54498. is there an error handling function?
54499. invalid option
54500. found a constant name?
54501. exchange its 'func' and 'extra' values
54502. calling function is a known Lua function?
54503. any instruction that set A
54504. could not find reasonable name
54505. table[line] = true
54506. is 'n' inside 'ci' stack?
54507. else try symbolic execution
54508. is code conditional (inside a jump)?
54509. get function that yielded
54510. ** this function can be called asynchronous (e.g. during a signal)
54511. is 'c' a constant?
54512. push it on stack
54513. it is its own name
54514. keep last instruction that changed 'reg'
54515. ** \$Id: ldebug.h,v 2.7.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Auxiliary functions from Debug Interface module ** See Copyright Notice in lua.h
54516. Active Lua function (given call info)
54517. call it (last chance to jump out)
54518. continue running the coroutine
54519. push error message
54520. finish interrupted instruction
54521. correct 'pc'
54522. stack grow uses memory
54523. run continuation
54524. save context
54525. an error occurred?
54526. back to caller
54527. execute down to higher C 'boundary'
54528. do the actual call

54529. Lua function: prepare its call
54530. no pending pcall
54531. allow yields
54532. dynamic structures used by the parser
54533. jump to it
54534. close possible pending closures
54535. is there a continuation?
54536. no recovery point
54537. restore old error handler
54538. res == final position of 1st result
54539. error message on current top
54540. move fixed parameters to final position
54541. ensure minimum stack size
54542. is a Lua function?
54543. not in base level?
54544. }=====

54545. **comment:** reuse preregistered msg.
 label: code-design
54546. erase new segment
54547. number of real arguments
54548. re-throw in main thread
54549. finish 'lua_callk'/'lua_pcall'
54550. 'oldpc' for caller function
54551. initialize upvalues
54552. does it have a continuation?
54553. no handler at all; abort
54554. C closure
54555. move results to correct place
54556. stack overflow?
54557. 'default' status
54558. hook may change stack
54559. dummy variable
54560. ** \$Id: ldo.c,v 2.108.1.3 2013/11/08 18:22:50 roberto Exp \$ ** Stack and Call structure of Lua ** See Copyright Notice in lua.h
54561. C++ exceptions
54562. ** do the work for 'lua_resume' in protected mode
54563. finish 'luaD_precall'
54564. light C function
54565. "finish" luaD_pcall
54566. was inside a pcall?
54567. ** LUAI_THROW/LUAI_TRY define how Lua does exception handling. By ** default, Lua handles errors with exceptions when compiling as ** C++ code, with _longjmp/_setjmp when asked to use them, and with ** longjmp/setjmp otherwise.
54568. Lua function
54569. number of arguments (Lua) or returns (C)
54570. restore 'nny'
54571. dynamic structure used by the scanner
54572. save 'nny'
54573. retry with 'function' tag method
54574. must have a continuation
54575. stack is empty?
54576. ** returns true if function has been executed (C function)
54577. cannot call hooks inside a hook
54578. yielded inside a hook?
54579. memory error?
54580. some space for error handling
54581. unrecoverable error
54582. protect stack below results
54583. cannot yield during parsing
54584. Lua function?
54585. call has error status
54586. shrink it
54587. now 'enter' new function
54588. main thread has a handler?
54589. hooks assume 'pc' is already incremented
54590. return to 'luaD_hook'
54591. jump back to 'lua_resume'
54592. error after extra size?
54593. coroutine is in base level; start running it
54594. panic function?
54595. call continuation
54596. inside a hook?
54597. mark it as dead
54598. 'common' yield
54599. first fixed argument
54600. no call status?
54601. resuming from previous yield
54602. check again for new 'base'
54603. previous call may change stack
54604. data to `f_parser'
54605. finish 'lua_pcall'
54606. chain new error handler
54607. ** check whether thread has a suspended protected call
54608. coroutine finished normally
54609. chain list of long jump buffers
54610. recover point?
54611. call it
54612. yield results come from continuation
54613. may be starting a coroutine
54614. (here it is)

54615. error while handing stack error
54616. tag method is the new function to be called
54617. ** {===== ** Error-recovery functions ** =====}
54618. call continuation function
54619. yield or regular error
54620. 0 iff wanted == LUA_MULTRET
54621. thread has no error handler
54622. complete missing arguments
54623. ** Execute a protected parser.
54624. just continue running Lua code
54625. **comment:** in Unix, try _longjmp/_setjmp (more efficient)
 label: code-design
54626. error calling 'lua_resume'?
54627. remove args from the stack
54628. final position of first argument
54629. not a function
54630. Open a hole inside the stack at `func'
54631. search for a pcall
54632. read first character
54633. set status
54634. copy error obj.
54635. default handling with long jumps
54636. ** Call a function (C or Lua). The function to be called is at *func. ** The arguments are on the stack, right after the function. ** When returns, all the results are on the stack, starting at the original ** function position.
54637. should be zero to be yieldable
54638. C function?
54639. handling stack overflow?
54640. starting point
54641. error?
54642. mark thread as `dead'
54643. must be inside a hook
54644. previous call can change stack
54645. error code
54646. ** signal an error in the call to 'resume', not in the execution of the ** coroutine itself. (Such errors should not be handled by any coroutine ** error handler and should not kill the coroutine.)
54647. would grow instead of shrink?
54648. don't change stack (change only for debugging)
54649. thread has an error handler?
54650. part of stack in use
54651. now it must be a function
54652. save current 'func'
54653. ** \$Id: ldo.h,v 2.20.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Stack and Call structure of Lua ** See Copyright Notice in lua.h
54654. **comment:** type of protected functions, to be ran by 'runprotected'
 label: code-design
54655. ** dump Lua function as precompiled chunk
54656. ** \$Id: ldump.c,v 2.17.1.1 2013/04/12 18:48:47 roberto Exp \$ ** save precompiled Lua chunks ** See Copyright Notice in lua.h
54657. include trailing '\0'
54658. current value lives in the stack
54659. is it open?
54660. remove from `uvhead' list
54661. ** Look for n-th local variable at line `line' in function `func'. ** Returns NULL if not found.
54662. remove from open list
54663. remove upvalue from 'uvhead' list
54664. found a corresponding upvalue?
54665. not found
54666. double link it in `uvhead' list
54667. now current value lives here
54668. move value to upvalue slot
54669. ** \$Id: lfunc.c,v 2.30.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Auxiliary functions to manipulate prototypes and closures ** See Copyright Notice in lua.h
54670. **comment:** is it dead?
 label: code-design
54671. not found: create a new one
54672. free upvalue
54673. resurrect it
54674. is variable active?
54675. link upvalue into 'allgc' list
54676. remove from `open' list
54677. ** \$Id: lfunc.h,v 2.8.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Auxiliary functions to manipulate prototypes and closures ** See Copyright Notice in lua.h
54678. at this point, all resurrected objects are marked.
54679. is 'curr' dead?
54680. true if table has entry "white-key -> white-value"
54681. "sweep" object
54682. no more strings to sweep?
54683. sweep all objects to turn them back to white (as white has not changed, nothing extra will be collected)
54684. mark any finalizing object left from previous cycle
54685. mark nested protos
54686. is there a weak mode?
54687. ** sweep a list until a live object (or end of list)
54688. **comment:** ** macro to adjust 'pause': 'pause' is actually used like ** 'pause / PAUSEADJ' (value chosen by tests)
 label: code-design
54689. remove it
54690. separate all objects with finalizers
54691. push finalizer...
54692. **comment:** may have to clean white keys
 label: code-design
54693. **comment:** free extra CallInfo slots
 label: code-design

54694. open upvalues remain gray
54695. do not run finalizers during emergency GC
54696. run entire collection
54697. clear nothing
54698. make object white
54699. strings are 'values', so are never weak
54700. ===== ** Sweep Functions ** =====

54701. memory traversed in this step
54702. ** barrier that moves collector forward, that is, mark the white object ** being pointed by a black object.
54703. ** set a reasonable "time" to wait before starting a new GC cycle; ** cycle will start when memory use hits threshold
54704. **comment:** unused and unmarked key; remove it
label: code-design
54705. nothing else to mark; make it black
54706. start counting work
54707. free all string lists
54708. at this point, all strongly accessible objects are marked.
54709. registry and global metatables may be changed by API
54710. **comment:** ** barrier for prototypes. When creating first closure (cache is ** NULL), use a forward barrier; this may be the only closure of the ** prototype (if it is a "regular" function, with a single instance) ** and the prototype may be big, so it is better to avoid traversing ** it again. Otherwise, use a backward barrier, to avoid marking all ** possible instances.
label: code-design
54711. return it to 'allgc' list
54712. mark value
54713. mark live elements in the stack
54714. ** enter first sweep phase (strings) and prepare pointers for other ** sweep phases. The calls to 'sweepolive' make pointers point to an ** object inside the list (instead of to the header), so that the real ** sweep do not need to skip objects created between "now" and the start ** of the real sweep. ** Returns how many objects it swept.
54715. it is being visited now
54716. set the old bit of all surviving objects
54717. mark that it is not in 'tobefnz'
54718. start new collection
54719. ** clear entries with unmarked keys from all weaktables in list 'l' up ** to element 'f'
54720. avoid removing current sweep object
54721. traverse marked some value?
54722. perform a full regular collection
54723. =====

54724. must keep invariant?
54725. update marks
54726. use a backward barrier
54727. strong keys?
54728. save what was counted
54729. first time?
54730. ** check color (and invariants) for an upvalue that was closed, ** i.e., moved into the 'allgc' list
54731. ** clear entries with unmarked values from all weaktables in list 'l' up ** to element 'f'
54732. is there an error object?
54733. traverse array part
54734. move 'o' to 'finobj' list
54735. stop counting (objects being finalized)
54736. adjust 'estimate'
54737. may there be some black objects?
54738. has to be cleared later
54739. remove from 'gray' list
54740. **comment:** ** retraverse all gray lists. Because tables may be reinserted in other ** lists when traversed, traverse the original lists to avoid traversing ** twice the same table (which is not wrong, but inefficient)
label: code-design
54741. remove 'cur' from list
54742. stop counting (do not (re)count grays)
54743. strong values?
54744. is there a white value?
54745. ** barrier that moves collector backward, that is, mark the black object ** pointing to a white object as gray again. (Current implementation ** only works for tables; access to 'gclist' is not uniform across ** different types.)
54746. traverse objects caught by write barrier and by 'remarkupvals'
54747. **comment:** remark occasional upvalues of (maybe) dead threads
label: code-design
54748. ** link table 'h' into list pointed by 'p'
54749. start to count memory traversed
54750. is there a finalizer?
54751. ** mark root set and reset all gray lists, to start a new ** incremental (or full) collection
54752. entry is empty?
54753. don't bother with it
54754. ** 'makewhite' erases all color bits plus the old bit and then ** sets only the current white bit
54755. update control
54756. generational mode?
54757. count call infos to compute size
54758. half the size of the string table
54759. clear keys from all ephemeron tables
54760. no need to clean
54761. mark running thread
54762. ... or is finalized...
54763. estimate of total memory traversed
54764. sweep open upvalues
54765. ** traverse one gray object, turning it to black (except for threads, ** which are always gray).
54766. stop sweeping this list
54767. complete counting
54768. go to next element
54769. signal for a major collection
54770. standard list for collectable objects

54771. ** {===== ** Finalization **
=====

54772. ** {===== ** Generic functions **
=====

54773. traverse main gray list
54774. ** mark an object. Userdata, strings, and closed upvalues are visited ** and turned black here. Other objects are marked gray and added ** to appropriate list to be visited (and turned black) later. (Open ** upvalues are already linked in 'headuv' list.)
54775. ** cost of sweeping one element (the size of a small object divided ** by some adjust for the sweep speed)
54776. ** create a new collectable object (with given type and size) and link ** it to '*list'. 'offset' tells how many bytes to allocate before the ** object itself (used only by states).
54777. or has no finalizer?
54778. ** advances the garbage collector until it reaches a state allowed ** by 'statemask'
54779. if there is array part, assume it may have white values (do not traverse it just to check)
54780. real end of stack
54781. remove it from 'tobefnz' list
54782. ** mark all values stored in marked open upvalues. (See comment in ** 'lstate.h'.)
54783. sweep main thread
54784. traverse hash part
54785. halve its size
54786. is really weak?
54787. change to incremental mode
54788. no more `gray' objects
54789. maximum number of finalizers to call in each GC step
54790. make sure gray lists are consistent
54791. ... and its argument
54792. mark its prototype
54793. ** mark metamethods for basic types
54794. value was collected?
54795. **comment:** ** macro to adjust 'stepmul': 'stepmul' is actually used like ** 'stepmul / STEPMULADJ' (value chosen by tests)
 label: code-design
54796. free concatenation buffer
54797. finish collection
54798. ** performs a basic GC step only if collector is running
54799. find last 'next' field in 'tobefnz' list (to add elements in its end)
54800. open upvalues are never black
54801. remove value ...
54802. ** {===== ** Traverse functions **
=====

54803. mark its upvalues
54804. not keeping invariant?
54805. do not sweep old generation
54806. overflow; truncate to maximum
54807. go through
54808. ** if object 'o' has a finalizer, remove it from 'allgc' list (must ** search the list to find it) and link it in 'finobj' list.
54809. prepare to sweep strings, finalizable objects, and regular objects
54810. ** \$Id: lgc.c,v 2.140.1.3 2014/09/01 16:55:08 roberto Exp \$ ** Garbage Collector ** See Copyright Notice in lua.h
54811. table must be cleared
54812. won't be finalized again
54813. re-throw error
54814. **comment:** this "white" makes all objects look dead
 label: code-design
54815. clear values from weak tables, before checking finalizers
54816. skip restart
54817. mark objects that will be finalized
54818. not being collected?
54819. error in __gc metamethod
54820. value not marked yet?
54821. ** change GC mode
54822. call one finalizer
54823. mark it as such
54824. finish mark phase
54825. restore state
54826. change to generational mode
54827. run a few finalizers (or all of them at the end of a collect cycle)
54828. no white keys
54829. avoid ridiculous low values (and 0)
54830. ** if key is not marked, mark its entry as dead (therefore removing it ** from the table)
54831. **comment:** remove dead objects from weak tables
 label: code-design
54832. true if an object is marked in this traversal
54833. ** {===== ** Mark functions **
=====

54834. bits to clear and to set in all live objects
54835. traverse all finalizable objects
54836. make prototype gray (again)
54837. remove 'o' from root list
54838. flip current white
54839. mark upvalue names
54840. clear values from resurrected weak tables
54841. add what was traversed by 'atomic'
54842. avoid being called too often
54843. mark main obj. as white to avoid other barriers
54844. make object gray (again)
54845. separate objects to be finalized
54846. must sweep all objects to turn them back to white (as white has not changed, nothing will be collected)
54847. stop sweep when this is true
54848. do not stop
54849. see MOVE OLD rule

54850. restore hooks
54851. no white values
54852. allow cache to be collected
54853. clear not-marked stack slice
54854. save original lists
54855. ** sweep at most 'count' elements from a list of GCObjects erasing dead ** objects, where a dead (not alive) object is one marked with the "old" ** (non current) white and not fixed. ** In non-generational mode, change all non-dead objects back to white, ** preparing for next collection cycle. ** In generational mode, keep black objects black, and also mark them as ** old; stop when hitting an old object, as all objects after that ** one will be old too. ** When object is a thread, sweep its list of open upvalues too.
54856. get ephemeron list
54857. ** move all unreachable objects (or 'all' objects) that need ** finalization from list 'finobj' to list 'tobefnz' (to be finalized)
54858. tables will return to this list when traversed
54859. always perform at least one single step
54860. insert into 'grayagain' list
54861. ** mark all objects in list of being-finalized
54862. mark it now
54863. ** tells whether a key or value can be cleared from a weak ** table. Non-collectable objects are never removed from weak ** tables. Strings behave as `values', so are never removed too. for ** other objects: if really collected, cannot keep them; for objects ** being finalized, keep them in keys, but not in values
54864. obj. is already separated...
54865. link at the end of 'tobefnz' list
54866. clear keys from all allweak tables
54867. final traversal?
54868. search for pointer pointing to 'o'
54869. should not change the stack during an emergency gc cycle
54870. get first element
54871. finish any pending sweep phase to start a new cycle
54872. sweep phase
54873. normal mode
54874. keep table gray
54875. restart counting
54876. sweep thread's upvalues
54877. true if table has white keys
54878. overflow?
54879. does table have white keys?
54880. nothing to traverse now
54881. table will have to be cleared
54882. do some work
54883. not weak
54884. remove 'curr' from 'finobj' list
54885. remove value
54886. all weak
54887. ** {===== ** GC control **
=====**
54888. ** sweep the (open) upvalues of a thread and resize its stack and ** list of call-info structures.
54889. traverse all elements from 'I'
54890. ** performs a full GC cycle; if "isemergency", does not call ** finalizers (which could change stack positions)
54891. generational mode must be kept in propagate phase
54892. no overflow
54893. propagate changes
54894. key is not marked (yet)?
54895. do not change sizes in emergency
54896. stack not completely built yet
54897. error while running __gc?
54898. avoid GC steps
54899. estimate of memory marked by 'atomic'
54900. using less than that half?
54901. mark basic metatables
54902. mark literals
54903. and remove entry from table
54904. no grays left
54905. signal for another major collection?
54906. convert debt from Kb to 'work units' (avoid zero debt and overflows)
54907. stop debug hooks during GC metamethod
54908. clear all color bits + old bit
54909. mark local-variable names
54910. nothing to change
54911. run complete (minor) cycle
54912. will have to revisit all ephemeron tables
54913. pause until next cycle
54914. restore invariant
54915. mark key
54916. open?
54917. have to propagate again
54918. empty
54919. convert 'work units' to Kb
54920. finalizers can create objs. in 'finobj'
54921. maximum number of elements to sweep in each single step
54922. keep estimate from last major coll.
54923. erase 'curr'
54924. ** one after last element in a hash array
54925. link it in list 'finobj'
54926. ** performs a basic GC step
54927. ** call all pending finalizers
54928. nothing to be done
54929. traverse array part (numeric keys are 'strong')
54930. and (next line) call the finalizer
54931. must propagate again
54932. remark, to propagate 'preserveness'

54933. **comment:** Layout for bit use in `marked' field:

label: code-design

54934. object has been separated for finalization

54935. object is old (only in generational mode)

54936. object is fixed (should not be collected)

54937. how much to allocate before next GC step

54938. **comment:** ** some useful bit tricks

label: code-design

54939. ** Collectable objects may have one of three colors: white, which ** means the object is not marked; gray, which means the ** object is marked, but its references may be not marked; and ** black, which means that the object and all its references are marked. ** The main invariant of the garbage collector, while marking objects, ** is that a black object can never point to a white one. Moreover, ** any gray object must be in a "gray list" (gray, grayagain, weak, ** allweak, ephemeron) so that it can be visited again before finishing ** the collection cycle. These lists have no meaning when the invariant ** is not being enforced (e.g., sweep phase).

54940. object is white (type 1)

54941. ** macros to tell when main invariant (white objects cannot point to black ** ones) must be kept. During a non-generational collection, the sweep ** phase may break the invariant, as objects turned white may point to ** still-black objects. The invariant is restored when sweep ends and ** all objects are white again. During a generational collection, the ** invariant must be kept all times.

54942. ** \$Id: lgc.h,v 2.58.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Garbage Collector ** See Copyright Notice in lua.h

54943. object is white (type 0)

54944. bit 7 is currently used by tests (luaL_checkmemory)

54945. ** Possible states of the Garbage Collector

54946. ~100 small strings

54947. MOVE OLD rule: whenever an object is moved to the beginning of a GC list, its old bit must be cleared

54948. neither white nor black

54949. object is black

54950. object is in 'finobj' list or in 'tobefnz'

54951. ** Outside the collector, the state in generational mode is kept in ** 'propagate', so 'keepinvariant' is always true.

54952. ** these libs are loaded by lua.c and are readily available to any Lua ** program

54953. remove lib

54954. remove _PRELOAD table

54955. ** \$Id: linit.c,v 1.32.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Initialization of libraries for lua.c and other clients ** See Copyright Notice in lua.h

54956. ** If you embed Lua in your program and need to open the standard ** libraries, call luaL_openlibs in your program. If you need a ** different set of libraries, copy this file to your project and edit ** it to suit your needs.

54957. ** these libs are preloaded and must be required before used

54958. call open functions from 'loadedlibs' and set results to global table

54959. add open functions from 'preloadedlibs' into 'package.preload' table

54960. create (and set) default files

54961. use standard output

54962. make sure argument is an open stream

54963. ** function to (not) close the standard files stdin, stdout, and stderr

54964. skip if char is '+'

54965. true iff read something

54966. ensure stack space for all results and for auxlib's buffer

54967. copy arguments

54968. close/not close file when finished

54969. at least one argument

54970. pop new metatable

54971. chop 'eol' if needed

54972. double buffer size at each iteration

54973. {

54974. mark file handle as 'closed'

54975. mark stream as closed

54976. number

54977. push metatable

54978. number of arguments to read

54979. eof?

54980. push arguments to 'g_read'

54981. }=====

54982. ** functions for 'io' library

54983. try to read 'n' chars

54984. metatable.__index = metatable

54985. file

54986. optimization: could be done exactly as for strings

54987. read entire file

54988. add file to module

54989. ** \$Id: liolib.c,v 2.112.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Standard I/O (and system) library ** See Copyright Notice in lua.h

54990. ** function to close 'popen' files

54991. avoid buffers too large

54992. check whether read something

54993. file handle

54994. to traverse/check mode

54995. ** methods for file handles

54996. 'n' is number of results

54997. }{

54998. file is already closed?

54999. line

55000. no argument?

55001. ** ===== ** lua_fseek: configuration for longer offsets **

55002. ignore closed and incompletely open files

55003. is there error information?

55004. prepare buffer to read whole block

55005. ** ===== ** lua_popen spawns a new process connected to the current ** one through the file streams. ** =====

55006. ** When creating file handles, always creates a 'closed' file handle ** before opening the actual file; so, if there is a memory error, the ** file is not left opened.

55007. create metatable for file handles

55008. always success

55009. ** function to close regular files

55010. Windows (but not DDK) and Visual C++ 2005 or higher
55011. 2nd result is error message
55012. skip if char is 'b'
55013. how much to read in each cycle
55014. put file at index 1
55015. close buffer
55016. add file methods to new metatable
55017. ** This definition must come before the inclusion of 'stdio.h'; it ** should not affect non-POSIX systems
55018. ** {===== ** READ **
=====

55019. add file to registry
55020. close it
55021. close it after iteration
55022. "result" to be removed
55023. keep file opened
55024. file handle already on stack top
55025. ** Check whether 'mode' matches '[rwa]%^+?b?'. ** Change this macro to accept other modes for 'fopen' besides ** the standard ones.
55026. open a new file
55027. new module
55028. line with end-of-line
55029. number of chars actually read
55030. read at least an `eol'
55031. read fails
55032. default definitions
55033. first result is nil: EOF or error
55034. return them
55035. check that it's a valid file handle
55036. error
55037. push nil instead
55038. not a file
55039. ensure that arguments will fit here and into 'io_readline' stack
55040. should return at least a nil
55041. do not close it after iteration
55042. }
55043. to return 1 result
55044. return current value
55045. no arguments?
55046. generator created file?
55047. push file at the stack top (to be returned)
55048. put it at index 1
55049. remove last result
55050. get default input
55051. no file name?
55052. read at least one value?
55053. LUA_NUMBER
55054. skip 2nd `l'
55055. short literal strings
55056. ** increment line number and skips newline sequence (any of ** \n, \r, \n\r, or \r\n)
55057. boolean value does not need GC barrier; table has no metatable, so it does not need to invalidate cache
55058. identifier or reserved word?
55059. '...'
55060. create env name
55061. save 'c'
55062. exponent part?
55063. digital escape \ddd
55064. try to update decimal point separator
55065. else short comment
55066. fixed format (symbols and reserved words)?
55067. initialize buffer
55068. string starts with a newline?
55069. ** skip a sequence [=*[`l or `]=*] and return its number of '='s or ** -1 if sequence is malformed
55070. read up to 3 digits
55071. temporarily anchor it in stack
55072. skip it
55073. use this one
55074. final character to be saved
55075. and discharge it
55076. read next token
55077. ** this function is quite liberal in what it accepts, as 'luaO_str2d' ** will reject ill-formed numerals.
55078. spaces
55079. long comment?
55080. reserved words are never collected
55081. string already present
55082. optional exponent sign
55083. long string or simply T'
55084. skip the 'z'
55085. skip `\\n\\r' or `\\r\\n'
55086. escape sequences
55087. skip until end of line (or end of file)
55088. **comment:** to avoid warnings
 label: code-design
55089. entry for `str'
55090. for error message
55091. names, strings, and numerals
55092. skip `\\n' or `\\r'
55093. result accumulator
55094. read next character
55095. ORDER RESERVED

55096. not in use yet? (see 'addK')
55097. else go through
55098. reserved word
55099. ** change all characters 'from' in buffer to 'to'
55100. read two hexadecimal digits
55101. hexadecimal?
55102. no look-ahead token
55103. will raise an error next loop
55104. previous call may dirty the buff.
55105. ** in case of format error, try to change decimal point separator to ** the one defined in the current locale and check again
55106. keep input for error message
55107. format error with correct decimal point: no more options
55108. do not save the `\'
55109. avoid wasting space
55110. skip long comment
55111. prepare error message
55112. ** creates a new string and anchors it in function's table so that ** it will not be collected until the end of the function's compilation ** (by that time it should be anchored in function's prototype)
55113. skip 2nd `]'
55114. create new string
55115. reserved word?
55116. skip delimiter
55117. follow locale for decimal point
55118. line breaks
55119. '..'
55120. re-use value previously stored
55121. '-' or '--' (comment)
55122. remove string from stack
55123. try new decimal separator
55124. ',', '.', '...', or number
55125. [t]string] = true
55126. else is a comment
55127. undo change (for error message)
55128. zap following span of spaces
55129. ** \$Id: llex.c,v 2.63.1.3 2015/02/09 17:56:34 roberto Exp \$ ** Lexical Analyzer ** See Copyright Notice in lua.h
55130. single-byte symbols?
55131. single-char tokens (+ - / ...)
55132. never collect this name
55133. is there a look-ahead token?
55134. keep delimiter (for error messages)
55135. 'skip_sep' may dirty the buffer
55136. ** ===== ** LEXICAL ANALYZER **
=====

55137. format error?
55138. environment variable name
55139. ** \$Id: llex.h,v 1.72.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lexical Analyzer ** See Copyright Notice in lua.h
55140. current character (charint)
55141. terminal symbols denoted by reserved words
55142. input stream
55143. semantics information
55144. dynamic structures used by the parser
55145. input line counter
55146. state of the lexer plus state of the parser when shared by all functions
55147. look ahead token
55148. number of reserved words
55149. current source name
55150. current token
55151. * WARNING: if you change the order of this enumeration, * grep "ORDER RESERVED"
55152. other terminal symbols
55153. buffer for tokens
55154. current function (parser)
55155. locale decimal point
55156. line of last token `consumed'
55157. type to ensure maximum alignment
55158. ** lua_number2int is a macro to convert lua_Number to int. ** lua_number2integer is a macro to convert lua_Number to lua_Integer. ** lua_number2unsigned is a macro to convert a lua_Number to a lua_Unsigned. ** lua_unsigned2number is a macro to convert a lua_Unsigned to a lua_Number. ** luai_hashnum is a macro to hash a lua_Number value into an integer. ** The hash must be deterministic and give reasonable values for ** both small and large values (outside the range of integers).
55159. ** conversion of pointer to integer ** this is for hashing only; there is no problem if the integer ** cannot hold the whole pointer value
55160. **comment:** ** macro to control inclusion of some hard tests on stack reallocation
 label: test
55161. ** non-return type
55162. ** assertion for checking API calls
55163. {}
55164. maximum value of an int (-2 for safety)
55165. realloc stack keeping its size
55166. avoid -0
55167. {}
55168. ** maximum depth for nested C calls and syntactical nested non-terminals ** in a program. (Value must fit in an unsigned short int.)
55169. **comment:** the following definitions always work, but may be slow
 label: code-design
55170. add double bits for his hash
55171. minimum size for string buffer
55172. ** maximum number of upvalues in a closure (both C and Lua). (Value ** must fit in an unsigned char.)
55173. ** these macros allow user-specific actions on threads when you defined ** LUAI_EXTRASPACE and need to do something extra when a thread is ** created/deleted/resumed/yielded.
55174. {}
55175. the next trick should work on any machine using IEEE754 with a 32-bit int type

55176. minimum size for the string table (must be power of 2)
55177. chars used as small naturals (so that 'char' is reserved for characters)
55178. trick with Microsoft assembler for X86
55179. **comment:** to avoid warnings
 label: code-design
55180. **comment:** on several machines, coercion from unsigned to double is slow, so it may be worth to avoid
 label: code-design
55181. internal assertions for in-house debugging
55182. to avoid problems with conditions too long
55183. empty
55184. ** type for virtual-machine instructions ** must be an unsigned with (at least) 4 bytes (see details in lopcodes.h)
55185. maximum stack for a Lua function
55186. the trick can be expanded to lua_Integer when it is a 32-bit value
55187. result of a `usual argument conversion' over lua_Number
55188. ** \$Id: llimits.h,v 1.103.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Limits, basic types, and some other `installation-dependent' definitions ** See Copyright Notice in lua.h
55189. the following definition assures proper modulo behavior
55190. only upper limit
55191. [l, u]
55192. number of arguments
55193. Number between 0 and 1
55194. ** \$Id: lmathlib.c,v 1.83.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Standard mathematical library ** See Copyright Notice in lua.h
55195. no arguments
55196. discard first value to avoid undesirable correlations
55197. ** Open math library
55198. lower and upper limits
55199. check number of arguments
55200. [1, u]
55201. the '%' avoids the (rare) case of r==1, and is needed also because on some systems (SunOS!) `rand()' may return a value larger than RAND_MAX
55202. ** \$Id: lmem.c,v 1.84.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Interface to Memory Manager ** See Copyright Notice in lua.h
55203. cannot double it?
55204. cannot grow even a little?
55205. update only when everything else is OK
55206. minimum size
55207. ** generic allocation routine.
55208. force a GC whenever possible
55209. try to free some memory...
55210. still have at least one free place
55211. try again
55212. ** About the realloc function: ** void * frealloc (void *ud, void *ptr, size_t osize, size_t nsize); ** ('osize' is the old size, 'nsize' is the new size) *** *** *
 frealloc(ud, NULL, x, s) creates a new block of size 's' (no ** matter 'x'). *** *** * frealloc(p, x, 0) frees the block 'p' *** (in this specific case, frealloc must
 return NULL); ** particularly, frealloc(ud, NULL, 0, 0) does nothing ** (which is equivalent to free(NULL) in ANSI C) *** *** frealloc returns NULL if it cannot
 create or reallocate the area ** (any reallocation to an equal or smaller size cannot fail!)
55213. not to be called directly
55214. ** \$Id: lmem.h,v 1.40.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Interface to Memory Manager ** See Copyright Notice in lua.h
55215. **comment:** ** This macro avoids the runtime division MAX_SISET/(e), as 'e' is ** always constant. ** The macro is somewhat complex to avoid warnings: ** +1
 avoids warnings of "comparison has constant result"; ** cast to 'void' avoids warnings of "value unused".
 label: code-design
55216. no errors?
55217. remove file name
55218. pop CLIBS table
55219. concatenate error message
55220. _LOADED[name] = true
55221. root not found
55222. unable to load library
55223. **comment:** concatenate error msg. entry
 label: code-design
55224. return nil + error message
55225. fill it with pre-defined searchers
55226. CLIBS[#CLIBS + 1] = plib
55227. create new metatable
55228. not found
55229. ** ===== ** This is an implementation of loadlib based on the
 dlfcn interface. ** The dlfcn interface is available in Linux, SunOS, Solaris, IRIX, FreeBSD, ** NetBSD, AIX 4.2, HPUX 11, and probably most other Unix
 flavors, at least ** as an emulation layer on top of native functions. **
=====
55230. run loader to load module
55231. is table an initialized module?
55232. non-empty separator?
55233. _LOADED[name]
55234. set field 'cpath'
55235. to build error message
55236. name is 1st argument (before search data)
55237. package is already loaded
55238. CLIBS[path] = plib
55239. ** LUA_PATH and LUA_CPATH are the names of the environment ** variables that Lua check to set its paths.
55240. **comment:** remove extra return
 label: code-design
55241. ** LUA_PATH_SEP is the character that separates templates in a path. ** LUA_PATH_MARK is the string that marks the substitution points in a ** template. **
 LUA_EXEC_DIR in a Windows path is replaced by the executable's ** directory. ** LUA_IGMARK is a mark to ignore all before it when building the **
 luaopen_function name.
55242. put it in field 'searchers'
55243. unable to find function
55244. use default
55245. iterate over available searchers to find a loader
55246. copy new environment table to top
55247. open failed
55248. set finalizer for CLIBS table

55249. try alternative name
55250. =====
55251. remove 'getfield' result
55252. module loader found
55253. module did not set a value?
55254. remove function
55255. **comment:** not used
 label: code-design
55256. does file exist and is readable?
55257. did it find a loader?
55258. get a searcher
55259. no errors
55260. ** return registry.LUA_NOENV as a boolean
55261. look for last dot in module name
55262. pop CLIBS table and 'plib'
55263. module not found in this path
55264. non-nil return?
55265. searcher returned error message?
55266. return nil, error message, and where
55267. check whether table already has a _NAME field
55268. else must load package
55269. try to open file
55270. return the loaded function
55271. set 'package' as upvalue for next lib
55272. error codes for ll_loadfunc
55273. set field 'preload'
55274. ** system-dependent functions
55275. open lib into global table
55276. ** {===== ** Fallback for other systems **
=====

55277. table (in the registry) that keeps handles for all loaded C libraries
55278. return that file name
55279. _LOADED table will be at index 2
55280. mt._index = _G
55281. put it in field 'loaders'
55282. module._M = module
55283. ** if needed, includes windows header before everything else
55284. no more searchers?
55285. auxiliary mark (for internal use)
55286. skip separators
55287. template
55288. is it there?
55289. return 'true'
55290. _LOADED[name] = returned value
55291. pop handle
55292. get calling function
55293. pop global table
55294. find next separator
55295. create error message
55296. module
55297. check loaded C libraries
55298. else create new function
55299. module loaded successfully?
55300. ** __gc tag method for CLIBS table: calls 'll_unloadlib' for all lib ** handles in list CLIBS
55301. set field 'path'
55302. return open function and file name
55303. no; initialize it
55304. for each handle, in reverse order
55305. open function not found
55306. ** optional flags for LoadLibraryEx
55307. real error
55308. no more templates
55309. create 'package' table
55310. remove nil
55311. ** {===== ** 'require' function **
=====

55312. **comment:** will be 2nd argument to module
 label: code-design
55313. metatable for CLIBS
55314. remove path template
55315. no environment variable?
55316. error message is on top of the stack
55317. must load library?
55318. pass name as argument to module loader
55319. error; error message is on stack top
55320. loading only library (no function)?
55321. use true as result
55322. store config information
55323. set field 'loaded'
55324. call it
55325. separator for open functions in C libraries
55326. **comment:** not used: symbols are 'global' by default
 label: code-design
55327. ** LUA_CSUBSEP is the character that replaces dots in submodule names ** when searching for a C loader. ** LUA_LSUBSEP is the character that replaces dots in submodule names ** when searching for a Lua loader.
55328. get option (a function)
55329. will be at index 3
55330. remove original string

55331. get handle CLIBS[n]
55332. avoid 'calling' extra info.
55333. remove value
55334. replace it by 'dirsep'
55335. is root
55336. else go ahead and try old-style name
55337. make a copy of 'searchers' table
55338. ** \$Id: loadlib.c,v 1.111.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Dynamic library loader for Lua ** See Copyright Notice in lua.h ** ** This module contains an implementation of loadlib for Unix systems ** that have dlcfn, an implementation for Windows, and a stub for other ** systems.
55339. prefix for open functions in C libraries
55340. extra copy to be returned
55341. set _PACKAGE as package name (full module name minus last part)
55342. create table CLIBS to keep track of loaded C libraries
55343. ** changes the environment variable of calling function
55344. not found?
55345. ** {===== This is an implementation of loadlib for Windows using native functions. ** ======

55346. plib = CLIBS[path]
55347. return 'package' table
55348. get/create module table
55349. ** {===== ** 'module' function ** ======

55350. create 'searchers' table
55351. last parameter
55352. remove both returns
55353. replace ";" by ";AUXMARK;" and then AUXMARK by default path
55354. set 'package' as upvalue for all searchers
55355. each fractional digit divides value by 2^4
55356. signal
55357. ** convert an hexadecimal numeric string to a number, following ** C99 specification for 'strtod'
55358. nothing recognized
55359. read integer part
55360. skip '0x'
55361. 1 if number is negative
55362. ** \$Id: lobject.c,v 2.58.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Some generic functions over Lua objects ** See Copyright Notice in lua.h
55363. should be enough space for a `%'
55364. small enough?
55365. valid up to here
55366. read exponent
55367. string; format as [string "source"]
55368. ** converts an integer to a "floating point byte", represented as ** (eeeeexxx), where the real value is (1xxx) * 2^(eeee - 1) if ** eeee != 0 and (xxx) otherwise.
55369. check '0x'
55370. invalid format (no '0x')
55371. converts back
55372. add '...' before rest of name
55373. fmt + item
55374. skip initial spaces
55375. exponent part?
55376. check signal
55377. 'literal' source
55378. must have at least one digit
55379. truncate it
55380. reject 'inf' and 'nan'
55381. this function handles only `%d', `%c', %f, %p, and `%s' formats
55382. invalid format (no digit)
55383. nothing is valid yet
55384. add prefix
55385. hexa?
55386. OK if no trailing characters
55387. skip dot
55388. skip 'p'
55389. save space for prefix+suffix+'\0'
55390. read fractional part
55391. exponent
55392. file name
55393. keep it
55394. find first new line (if any)
55395. number of chars of a literal string without the ending \0
55396. small one-line source?
55397. stop at first newline
55398. number of characters in string
55399. collectable objects
55400. Macros to access values
55401. constants used by the function
55402. to new object
55403. long strings
55404. reserved words for short strings; "has hash" for longs
55405. double linked list (when open)
55406. ** Union of all collectable objects
55407. ** numbers are represented in the 'd_' field. All other values have the ** value (NNMARK | tag) in 'tt__'. A number with such pattern would be ** a "signaled NaN", which is never generated by regular operations by ** the CPU (nor by 'strtod')
55408. log2 of size of `node' array
55409. {
55410. Macros for internal tests
55411. index of upvalue (in stack or in outer function's list)
55412. ** Tables
55413. numbers
55414. }=====

55415. Variant tags for strings
55416. macro defining a nil value
55417. ** Extra tags for non-values
55418. map from opcodes to source lines (debug information)
55419. light userdata
55420. allows for external implementation for part of the trick
55421. ** Closures
55422. mark a tag as collectable
55423. ensures maximum alignment for strings
55424. maximum stack used by this function
55425. ** \$Id: lobject.h,v 2.71.1.2 2014/05/07 14:14:58 roberto Exp \$ ** Type definitions for Lua objects ** See Copyright Notice in lua.h
55426. used for debug information
55427. C closure
55428. }{
55429. ensures maximum alignment for 'local' udata
55430. functions defined inside the function
55431. Lua closure
55432. ** Common header in struct form
55433. size of 'upvalues'
55434. light C function
55435. correspondence with standard representation
55436. ** different types of assignments, according to destination
55437. check whether a number is valid (useful only for NaN trick)
55438. whether it is in stack
55439. for chaining
55440. array part
55441. get the actual string (array of bytes) from a TString
55442. ** Tagged Values. This is the basic representation of values in Lua, ** an actual value plus a tag with its type.
55443. basic check to distinguish numbers from non-numbers
55444. to stack (not from same stack)
55445. **comment:** ** these redefinitions are not mandatory, but these forms are more efficient
 label: code-design
55446. ** (address of) a fixed nil value
55447. the value (when closed)
55448. ** Description of an upvalue for function prototypes
55449. type tag of a TValue with no variants (bits 0-3)
55450. Macros to test type
55451. light C functions
55452. ** Lua Upvalues
55453. **comment:** first point where variable is dead
 label: code-design
55454. ** tags for Tagged Values have the following use of bits: ** bits 0-3: actual tag (a LUA_T* value) ** bits 4-5: variant bits ** bit 6: whether value is collectable
55455. a dead value may get the 'gc' field, but cannot access its contents
55456. no such field; numbers are the entire struct
55457. size of 'p'
55458. ** LUA_TFUNCTION variants: ** 0 - Lua function ** 1 - light C function ** 2 - regular C function (closure)
55459. ** Header for userdata; memory area follows the end of this structure
55460. from table to same table
55461. size of 'array' array
55462. short strings
55463. size of 'k'
55464. ** 'module' operation for hashing (size is always a power of 2)
55465. ** Description of a local variable for function prototypes ** (used for debug information)
55466. first point where variable is active
55467. any free position is before this position
55468. tag with no variants (bits 0-3)
55469. upvalue name (for debug information)
55470. Variant tags for functions
55471. last created closure with this prototype
55472. points to stack or to its own value
55473. ** {===== ** types and prototypes **
=====**
55474. type tag of a TValue (bits 0-3 for tags + variant bits 4-5)
55475. Macros to set values
55476. booleans
55477. upvalue information
55478. ** Header for string value; string bytes follow the end of this structure
55479. little endian
55480. to table
55481. ** number of all possible tags (including LUA_TNONE but excluding DEADKEY)
55482. }
55483. Bit mark for collectable types
55484. ** {===== ** NaN Trick **
=====**
55485. index to stack elements
55486. number of fixed parameters
55487. big endian
55488. get the actual string (array of bytes) from a Lua value
55489. raw type tag of a TValue
55490. from stack to (same) stack
55491. 1<<p means tagmethod(p) is not present
55492. number of bytes
55493. list of upvalues
55494. ** Common Header for all collectable objects (in macro form, to be ** included in other objects)
55495. information about local variables (debug information)
55496. ** Union of all Lua values
55497. empty
55498. field-access macros

55499. ** Function Prototypes
 55500. OP_LOADNIL
 55501. OP_UNM
 55502. OP_MOVE
 55503. OP_SELF
 55504. OP_VARARG
 55505. OP_RETURN
 55506. OP JMP
 55507. OP_FORPREP
 55508. OP_CLOSURE
 55509. OP_LOADKX
 55510. OP_DIV
 55511. OP_NOT
 55512. OP_LE
 55513. OP_TFORCALL
 55514. OP_LEN
 55515. OP_EQ
 55516. OP_SETUPVAL
 55517. OP_POW
 55518. OP_LT
 55519. OP_SETLIST
 55520. OP_TFORLOOP
 55521. T A B C mode opcode
 55522. ORDER OP
 55523. OP_EXTRAARG
 55524. OP_CONCAT
 55525. OP_SETTABLE
 55526. OP_MUL
 55527. OP_LOADK
 55528. OP_GETTABLE
 55529. OP_LOADBOOL
 55530. OP_GETTABUP
 55531. OP_FORLOOP
 55532. OP_MOD
 55533. OP_NEWTABLE
 55534. OP_SETTABUP
 55535. OP_TEST
 55536. OP_TAILCALL
 55537. OP_ADD
 55538. ** \$Id: lopcodes.c,v 1.49.1.1 2013/04/12 18:48:47 roberto Exp \$ *** Opcodes for Lua virtual machine ** See Copyright Notice in lua.h
 55539. OP_CALL
 55540. OP_SUB
 55541. OP_GETUPVAL
 55542. OP_TESTSET
 55543. A B R(A), R(A+1), ..., R(A+B) := nil
 55544. **comment:** argument is not used
 label: code-design
 55545. ** invalid register that fits in 8 bits
 55546. number of list items to accumulate before a SETLIST instruction
 55547. A B C R(A) := R(B)[RK(C)]
 55548. A B C if ((RK(B) == RK(C)) ~= A) then pc++
 55549. A C if not (R(A) <= C) then pc++
 55550. A B C if (R(B) <= C) then R(A) := R(B) else pc++
 55551. test whether value is a constant
 55552. A B C if ((RK(B) < RK(C)) ~= A) then pc++
 55553. ** R(x) - register ** Kst(x) - constant (in constant table) ** RK(x) == if ISK(x) then Kst(INDEXK(x)) else R(x)
 55554. ===== We assume that instructions are unsigned numbers. All
 instructions have an opcode in the first 6 bits. Instructions can have the following fields: 'A' : 8 bits 'B' : 9 bits 'C' : 9 bits 'Ax' : 26 bits ('A', 'B', and 'C' together)
 'Bx' : 18 bits ('B' and 'C' together) `sBx' : signed Bx A signed argument is represented in excess K; that is, the number value is the unsigned value minus K. K is
 exactly the maximum value for that argument (so that -max is represented by 0, and +max is represented by 2*max), which is half the maximum for the
 corresponding unsigned argument. =====
 55555. A B C R(A+1) := R(B); R(A) := R(B)[RK(C)]
 55556. A B C return R(A)(R(A+1), ... ,R(A+B-1))
 55557. this bit 1 means constant (0 means register)
 55558. A Bx R(A) := Kst(Bx)
 55559. ** Macros to operate RK indices
 55560. basic instruction format
 55561. A B C R(A) := UpValue[B][RK(C)]
 55562. A C R(A+3), ... ,R(A+2+C) := R(A)(R(A+1), R(A+2));
 55563. A B C R(A) := RK(B) ^ RK(C)
 55564. ** the following macros help to manipulate instructions
 55565. A B C R(A)[RK(B)] := RK(C)
 55566. A sBx if R(A+1) ~= nil then { R(A)=R(A+1); pc += sBx }
 55567. A B C R(A) := RK(B) - RK(C)
 55568. A sBx R(A)+=R(A+2); if R(A) <= R(A+1) then { pc+=sBx; R(A+3)=R(A) }
 55569. A B C R(A)[(C-1)*FPF+i] := R(A+i), 1 <= i <= B
 55570. `sBx' is signed
 55571. A B C R(A) := {} (size = B,C)
 55572. A B UpValue[B] := R(A)
 55573. A B C if ((RK(B) <= RK(C)) ~= A) then pc++
 55574. A sBx R(A)=R(A+2); pc+=sBx
 55575. ----- name args description -----
 55576. ===== Notes: (*) In OP_CALL, if (B == 0) then B = top. If (C == 0), then 'top' is set to last_result+1, so next open instruction (OP_CALL, OP_RETURN, OP_SETLIST) may use 'top'. (*) In OP_VARARG, if (B == 0) then
 use actual number of varargs and set top (like in OP_CALL with C == 0). (*) In OP_RETURN, if (B == 0) then return up to 'top'. (*) In OP_SETLIST, if (B == 0)
 then B = 'top'; if (C == 0) then next 'instruction' is EXTRAARG(real C). (*) In OP_LOADKX, the next 'instruction' is always EXTRAARG. (*) For comparisons,
 A specifies what condition the test should accept (true or false). (*) All 'skips' (pc++) assume that next instruction is a jump.
 =====

55577. A B C R(A) := RK(B) / RK(C)
55578. ** size and position of opcode arguments.
55579. A B R(A) := not R(B)
55580. A B C R(A) := RK(B) * RK(C)
55581. ** limits for opcode arguments. ** we use (signed) int to manipulate most arguments, ** so they must fit in LUAI_BITSINT-1 bits (-1 for sign)
55582. argument is used
55583. A B R(A) := length of R(B)
55584. A B R(A) := R(B)
55585. opcode names
55586. ** grep "ORDER OP" if you change these enums
55587. argument is a register or a jump offset
55588. A B C R(A) := (Bool)B; if (C) pc++
55589. gets the index of the constant
55590. ** \$Id: lopcodes.h,v 1.142.1.2 2014/10/20 18:32:09 roberto Exp \$ ** Opcodes for Lua virtual machine ** See Copyright Notice in lua.h
55591. **comment:** Ax extra (larger) argument for previous opcode
 label: code-design
55592. argument is a constant or register/constant
55593. A B return R(A), ... ,R(A+B-2) (see note)
55594. A B C R(A) := RK(B) % RK(C)
55595. A B R(A) := -R(B)
55596. A Bx R(A) := closure(KPROTO[Bx])
55597. A B R(A) := UpValue[B]
55598. A B C R(A) := R(B)..R(C)
55599. A R(A) := Kst(extra arg)
55600. A B C UpValue[A][RK(B)] := RK(C)
55601. A B R(A), R(A+1), ..., R(A+B-2) = vararg
55602. A B C R(A), ... ,R(A+C-2) := R(A)(R(A+1), ... ,R(A+B-1))
55603. code a constant index as a RK value
55604. ** masks for instruction properties. The format is: ** bits 0-1: op mode ** bits 2-3: C arg mode ** bits 4-5: B arg mode ** bit 6: instruction set register A ** bit 7: operator is a test (next instruction must be a jump)
55605. A sBx pc+=sBx; if (A) close all upvalues >= R(A - 1)
55606. creates a mask with `n' 1 bits at position `p'
55607. creates a mask with `n' 0 bits at position `p'
55608. A B C R(A) := RK(B) + RK(C)
55609. valid two-char conversion specifier
55610. ** list of valid conversion specifiers for the 'strftime' function
55611. invalid date?
55612. UTC?
55613. ** By default, Lua uses tmpnam except when POSIX is available, where it ** uses mkstemp.
55614. ** By default, Lua uses gmtime/localtime, except when POSIX is available, ** where it uses gmtime_r/localtime_r
55615. should be big enough for any conversion result
55616. if NULL push nil
55617. get current time
55618. called without args?
55619. one-char conversion specifier?
55620. does not set field
55621. **comment:** to avoid warnings
 label: code-design
55622. ======
55623. end buffer
55624. ** ====== ** Time/Date operations ** { year=%Y, month=%m, day=%d, hour=%H,
 min=%M, sec=%S, ** wday=%w+1, yday=%j, isdst=? } ** ======
55625. 9 = number of fields
55626. skip '!'
55627. undefined?
55628. **comment:** 'if' to avoid warnings for unreachable 'return'
 label: code-design
55629. no conversion specifier?
55630. make sure table is at the top
55631. ** \$Id: loslib.c,v 1.40.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Standard Operating System library ** See Copyright Notice in lua.h
55632. true if there is a shell
55633. fix it at the last register
55634. not found as local at current level; try upvalues
55635. index is the local being assigned? (index cannot be upvalue)
55636. end of scope for declared variables
55637. function created in next register
55638. read first token
55639. must skip over 'then' part if condition is false
55640. assume that locals are already out of scope
55641. global name?
55642. param -> `...'
55643. create and...
55644. `: NAME funcargs
55645. funcname -> NAME {fieldsel} [`: NAME]
55646. not found
55647. ** check whether, in an assignment to an upvalue/local variable, the ** upvalue/local variable is begin used in a previous assignment to a ** table. If so, save original upvalue/local value in a safe place and ** use this safe copy in the previous assignment.
55648. '[' exp1 ']'
55649. expand while operators have priorities higher than 'limit'
55650. must enter block before 'goto'
55651. env[varname]
55652. stat -> forstat
55653. ** compiles the main function, which is a regular vararg function with an ** upvalue named LUA_ENV
55654. no value (yet)
55655. skip WHILE
55656. free registers
55657. ======
55658. stat -> func | assignment

55659. no more levels?
55660. reserve register for parameters
55661. ** structure to chain all variables in the left-hand side of an ** assignment
55662. `falses' are all equal here
55663. control variables
55664. left priority for each binary operator
55665. skip over block if condition is false
55666. skip DO
55667. local function?
55668. ~=, >, >=
55669. ...set environment upvalue
55670. skip other no-op statements
55671. last token from outer function must be EOS
55672. forstat -> FOR (fornum | forlist) END
55673. ^, .. (right associative)
55674. cond -> exp
55675. create control variables
55676. chain
55677. last list item read
55678. skip double colon
55679. instruction to skip 'then' code (if condition is false)
55680. remove goto from pending list
55681. patch escape list to 'if' end
55682. only one single value?
55683. funcargs
55684. upvalues?
55685. scope block
55686. main function is always vararg
55687. generic for
55688. check all previous assignments
55689. correct pending gotos to current block and try to close it with visible labels
55690. debug information will only see the variable after this point!
55691. variable is local
55692. stat -> ';' (empty statement)
55693. stat -> 'goto' NAME
55694. close the loop
55695. return no values
55696. includes call itself
55697. open call
55698. funcstat -> FUNCTION funcname body
55699. stat -> assignment ?
55700. remove 'near to' from final message
55701. numeric for?
55702. stat -> LOCAL NAME {, NAME} [= explist]
55703. set initial array size
55704. key is variable name
55705. tail call?
55706. may be needed for error messages
55707. last exp. provides the difference
55708. ** "export" pending gotos to outer level, to check them against ** outer labels; if the block being exited has upvalues, and ** the goto exits the scope of any variable (which can be the ** upvalue), close those variables being exited.
55709. ** \$Id: lp parser.c,v 2.130.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua Parser ** See Copyright Notice in lua.h
55710. create and anchor TString
55711. check for repeated labels
55712. linked list of funcstates
55713. ** nodes for block list (list of active blocks)
55714. close it if label already defined
55715. IF cond THEN block
55716. stat -> breakstat
55717. update pending gotos to outer block
55718. anchor closure (to avoid being collected)
55719. block -> statlist
55720. flush
55721. call statement uses no results
55722. correct label?
55723. label not found; cannot close goto
55724. prototype of current function
55725. create declared variables
55726. arg list is empty?
55727. skip ','
55728. it's on the stack too
55729. skip the '['
55730. skip no-op statements
55731. stat -> retstat
55732. stat -> label
55733. stat -> DO block END
55734. else was LOCAL or UPVAL
55735. ** check whether new label 'lb' matches any pending gotos in current ** block; solves forward jumps
55736. inner block?
55737. optional step
55738. default is global
55739. there is no list item
55740. constructor
55741. return all `active' values
55742. ===== ** Rules for Constructors **
55743. registers with returned values
55744. true if some variable in the block is an upvalue

55745. limit
55746. return first untreated operator
55747. **comment:** maximum number of local variables per function (must be smaller than 250, due to the bytecode format)
 label: code-design
55748. default step = 1
55749. close last argument
55750. forum -> NAME = exp1,exp1[,exp1] forbody
55751. simpleexp -> NUMBER | STRING | NIL | TRUE | FALSE | ... | constructor | FUNCTION body | suffixedexp
55752. 'then' part
55753. anchor table of constants (to avoid being collected)
55754. look up locals at current level
55755. ELSEIF cond THEN block
55756. ORDER OPR
55757. may be 'listfield' or 'recfield'
55758. stat -> RETURN [explist] ';'
55759. constructor -> '{' [field { sep field } [sep]] '}' sep -> ',' | ';' | '
55760. variable (global, local, upvalue, or indexed)
55761. previous assignment will use safe copy
55762. followed by 'else'/elseif?
55763. check labels in current block for a match
55764. ls->t.token == 'l'
55765. finish scope
55766. index -> '[' expr ']'
55767. Find variable with given name 'n'. If it is an upvalue, add this upvalue into all intermediate functions.
55768. ======
55769. skip RETURN
55770. close it
55771. **comment:** remove extra values
 label: code-design
55772. ** generates an error for an undefined 'goto'; choose appropriate ** message when label name is a reserved word (which can only be 'break')
55773. stat -> funcstat
55774. all scopes should be correctly finished
55775. remove local labels
55776. stat -> func
55777. stat -> whilstat
55778. pending gotos in outer block?
55779. new local variable
55780. ** try to close a goto with existing labels; this solves backward jumps
55781. skip the dot or colon
55782. expression?
55783. initial value
55784. must jump over it
55785. label is last no-op statement in the block?
55786. registers 0/1 are always valid
55787. true if 'block' is a loop
55788. ====== ** Expression parsing **
=====

55789. suffixedexp -> primaryexp { '.' NAME | '[' exp ']' | ':' NAME funcargs | funcargs }
55790. scope for loop and control variables
55791. label -> '::' NAME '::'
55792. try upper levels
55793. eventual position to save local variable
55794. optional return values
55795. enter its scope
55796. stat -> repeatstat
55797. param -> NAME
55798. 'return' must be last statement
55799. move to next one
55800. create a 'jump to here' to close upvalues
55801. 'else' part
55802. do not count last expression (unknown number of elements)
55803. field -> listfield | recfield
55804. vararg
55805. regular case (not goto/break)
55806. assignment -> ',' suffixedexp assignment
55807. index of first label in this block
55808. GRAMMAR RULES
55809. repeatstat -> REPEAT block UNTIL cond
55810. and that is it
55811. skip optional semicolon
55812. check for repeated labels on the same block
55813. **comment:** extra space to call generator
 label: code-design
55814. ** subexpr -> (simpleexp | unop subexpr) { binop subexpr } ** where 'binop' is any binary operator with a priority higher than 'limit'
55815. avoid default
55816. finish loop
55817. fieldsel -> '[' | ':'] NAME
55818. ====== ** Rules for Statements **
=====

55819. forbody -> DO block
55820. will jump to label if condition is true
55821. read condition
55822. create new entry for this label
55823. call remove function and arguments and leaves (unless changed) one result
55824. false conditions finish the loop
55825. 'goto' is the entire block?
55826. ** prototypes for recursive non-terminal functions
55827. index of first pending goto in this block

55828. fieldsel
55829. definition 'happens' in the first line
55830. assignment -> '=' explist
55831. ** check whether current token is in the follow set of a block. ** 'until' closes syntactical blocks, but do not close scope, ** so it handled in separate.
55832. error
55833. skip REPEAT
55834. table descriptor
55835. test_then_block -> [IF | ELSEIF] cond THEN block
55836. stat -> ifstat
55837. gen, state, control, plus at least one declared var
55838. right priority
55839. ** adds a new prototype into list of prototypes
55840. no more items pending
55841. ** codes instruction to create new closure in parent function. ** The OP_CLOSURE instruction must use the last available register, ** so that, if it invokes the GC, the GC knows which registers ** are in use at that time.
55842. at least one expression
55843. funcargs -> `(` [explist] `)`
55844. read sub-expression with higher priority
55845. whilestat -> WHILE cond DO block END
55846. loop block
55847. skip 'for'
55848. read condition (inside scope block)
55849. try existing upvalues
55850. primaryexp -> NAME | `(` expr `)`
55851. `+` `.` `*` `/` `%`
55852. total number of 'record' elements
55853. values must go to the 'stack'
55854. priority for unary operators
55855. create 'self' parameter
55856. table is the upvalue/local being assigned now?
55857. scope for declared variables
55858. pop table of constants
55859. # active locals outside the block
55860. close last expression
55861. handle goto/break
55862. listfield -> exp
55863. forlist -> NAME {,NAME} IN explist forbody
55864. statlist -> { stat [;] }
55865. base register for call
55866. skip break
55867. Mark block where variable at given level was defined (to emit close instructions later).
55868. local will be used as an upval
55869. index of new label being created
55870. first variable name
55871. semantic error
55872. copy upvalue/local value to a temporary (in position 'extra')
55873. exit list for finished parts
55874. funcargs -> constructor
55875. return all values
55876. must use 'seminfo' before 'next'
55877. number of array elements pending to be stored
55878. not found; is a global
55879. skip LOCAL
55880. not found?
55881. set initial table size
55882. and, or
55883. ** create a label named "break" to resolve break statements
55884. found?
55885. ==, <, <=
55886. default assignment
55887. funcargs -> STRING
55888. loop scope ('break' jumps to this point)
55889. will be a new upvalue
55890. recfield -> (NAME | `['exp1`]`) = exp1
55891. skip FUNCTION
55892. stat -> localstat
55893. skip IF or ELSEIF
55894. close pending breaks
55895. total number of array elements
55896. }=====

55897. final return
55898. fix it at stack top
55899. assigning to a table?
55900. is 'parlist' not empty?
55901. body -> (' parlist ') block END
55902. parlist -> [param { `,' param }]
55903. explist -> expr { `,' expr }
55904. ifstat -> IF cond THEN block {ELSEIF cond THEN block} [ELSE block] END
55905. parse main body
55906. create main closure
55907. last token read was anchored in defunct function; must re-anchor it
55908. get environment variable
55909. patch list of 'exit when true'
55910. whether 't' is register (VLOCAL) or upvalue (VUPVAL)
55911. info = result register
55912. chain of current blocks
55913. t = table register/upvalue; idx = index R/K
55914. list of labels or gotos

55915. for indexed variables (VINDEXED)
55916. label identifier
55917. list of pending jumps to 'pc'
55918. array
55919. info = index of constant in 'k'
55920. ** \$Id: lp parser.h,v 1.70.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua Parser ** See Copyright Notice in lua.h
55921. table to find (and reuse) elements in 'k'
55922. number of elements in 'f->locvars'
55923. current function header
55924. no value
55925. for generic use
55926. dynamic structures used by the parser
55927. description of active local variable
55928. variable index in stack
55929. number of entries in use
55930. lexical state
55931. next position to code (equivalent to 'ncode')
55932. number of elements in 'k'
55933. first free register
55934. enclosing function
55935. description of pending goto statements and label statements
55936. local level where it appears in current block
55937. 'label' of last 'jump label'
55938. table (register or upvalue)
55939. list of active local variables
55940. state needed to generate code for a given function
55941. patch list of 'exit when false'
55942. nval = numerical value
55943. for VKNUM
55944. info = instruction pc
55945. info = local register
55946. number of upvalues
55947. array size
55948. number of active local variables
55949. list of pending gotos
55950. position in code
55951. defined in lp parser.c
55952. number of elements in 'p'
55953. control of blocks
55954. list of active labels
55955. index of first local var (in Dyndata array)
55956. ** Expression descriptor
55957. info = index of upvalue in 'upvalues'
55958. index (R/K)
55959. line where it appeared
55960. ** open parts of the state that may cause memory-allocation errors
55961. no GC while building state
55962. erase new stack
55963. public function
55964. allow gc
55965. memory allocation error: free partial state
55966. ** Compute an initial seed as random as possible. In ANSI, rely on ** Address Space Layout Randomization (if present) to increase ** randomness..
55967. collect all objects
55968. registry[LUA_RIDX_GLOBALS] = table of globals
55969. ** \$Id: lstate.c,v 2.99.1.2 2013/11/08 17:45:31 roberto Exp \$ ** Global State ** See Copyright Notice in lua.h
55970. heap variable
55971. create registry
55972. close all upvalues for this thread
55973. only the main thread can be closed
55974. 200%
55975. initialize first ci
55976. ** thread state + extra space
55977. ** Create registry table and its predefined values
55978. stack not completely built yet
55979. registry[LUA_RIDX_MAINTHREAD] = L
55980. closing a fully built state?
55981. global variable
55982. local variable
55983. 'function' entry for this 'ci'
55984. free stack array
55985. it should never be collected
55986. GC runs 'twice the speed' of memory allocation
55987. init stack
55988. ** preinitialize a state with consistent values without allocating ** any memory (to avoid errors)
55989. ** a macro to help the creation of a unique random seed when a state is ** created; the seed is used to randomize hashes.
55990. ** set GCdebt to a new value keeping the value (totalbytes + GCdebt) ** invariant
55991. ** Main thread combines a thread state and the global state
55992. free the entire 'ci' list
55993. free main block
55994. initial size of string table
55995. pre-create memory-error message
55996. initialize stack array
55997. metatables for basic types
55998. thread
55999. to be called in unprotected errors
56000. current error recover point
56001. current position of sweep in list 'finobj'
56002. ** Union of all collectable objects

56003. randomized seed for hashes
56004. call is running on same invocation of luaV_execute of previous call
56005. call has an error status (pcall)
56006. call was tail called
56007. number of elements
56008. head of double-linked list of all open upvalues
56009. function index in the stack
56010. **comment:** call is a yieldable protected call
 label: code-design
56011. list of gray objects
56012. an estimate of the non-garbage memory in use
56013. number of bytes currently allocated - GCdebt
56014. list of tables with weak values
56015. list of open upvalues in this stack
56016. list of collectable objects with finalizers
56017. list of all-weak tables
56018. last free slot in the stack
56019. call is running a Lua function
56020. stack base
56021. true if GC is running
56022. ** Bits in CallInfo status
56023. macro to convert any Lua object into a GCOBJECT
56024. current error handling function (stack index)
56025. number of non-yieldable calls in stack
56026. list of all collectable objects
56027. extra stack space to handle TM calls and some other extras
56028. state of garbage collector
56029. call reentered after suspension
56030. actual number of total bytes allocated
56031. bytes allocated not yet compensated by the collector
56032. kinds of Garbage Collection
56033. defined in ldo.c
56034. array with tag-method names
56035. ** information about a call
56036. gc was forced by an allocation failure
56037. last hook called yielded
56038. only for Lua functions
56039. ** 'global state', shared by all threads of this state
56040. size of pause between successive GCs
56041. ** Some notes about garbage-collected objects: All objects in Lua must ** be kept somehow accessible until being freed. ** ** Lua keeps most objects linked in list g->allgc. The link uses field ** 'next' of the CommonHeader. ** ** Strings are kept in several lists headed by the array g->strt.hash. ** ** Open upvalues are not subject to independent garbage collection. They ** are collected together with their respective threads. Lua keeps a ** double-linked list with all open upvalues (g->uvhead) so that it can ** mark objects referred by them. (They are always gray, so they must ** be remarked in the atomic step. Usually their contents would be marked ** when traversing the respective threads, but the thread may already be ** dead, while the upvalue is still accessible through closures.) ** ** Objects with finalizers are kept in the list g->finobj. ** ** The list g->tobefnz links all objects being finalized.
56042. base for this function
56043. first free slot in the stack
56044. generational collection
56045. kind of GC running
56046. memory-error message
56047. current position of sweep in list 'allgc'
56048. only for C functions
56049. CallInfo for first level (C calling Lua)
56050. ** 'per thread' state
56051. list of userdata to be GC
56052. list of objects to be traversed atomically
56053. macros to convert a GCOBJECT into a specific value
56054. hash table for strings
56055. ** \$Id: lstate.h,v 2.82.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Global State ** See Copyright Notice in lua.h
56056. last pc traced
56057. top for this function
56058. auxiliary data to 'frealloc'
56059. expected number of results from this function
56060. call is running a debug hook
56061. continuation in case of yields
56062. dynamic call link
56063. list of ephemeron tables (weak keys)
56064. common header
56065. GC 'granularity'
56066. call info for current function
56067. pointer to version number
56068. position of sweep in 'strt'
56069. context info. in case of yields
56070. number of nested C calls
56071. **comment:** temporary buffer for string concatenation
 label: code-design
56072. pause between major collections (only in gen. mode)
56073. function to reallocate memory
56074. memory traversed by the GC
56075. ** equality for long strings
56076. same instance or...
56077. cannot resize while GC is traversing strings
56078. short string?
56079. chain it
56080. shrinking slice must be empty
56081. total size of TString object
56082. rehash
56083. equal length and ...

56084. ** new zero-terminated string
56085. ** \$Id: lstring.c,v 2.26.1.1 2013/04/12 18:48:47 roberto Exp \$ ** String table (keeps all strings handled by Lua) ** See Copyright Notice in lua.h
56086. ** Lua will use at most ~(2^LUAI_HASHLIMIT) bytes from a string to ** compute its hash
56087. ** checks whether short string exists and reuses it or creates a new one
56088. (pointer to) list where it will be inserted
56089. ** creates a new short string, inserting it into string table
56090. ** new string (with explicit length)
56091. ending 0
56092. ** resizes the string table
56093. string is dead (but was not collected yet)?
56094. not found; create a new string
56095. resurrect it
56096. ** equality for strings
56097. see MOVE OLD rule
56098. too crowded
56099. new position
56100. ** creates a new string object
56101. save next
56102. for each node in the list
56103. equal contents
56104. ** \$Id: lstring.h,v 1.49.1.1 2013/04/12 18:48:47 roberto Exp \$ ** String table (keep all strings handled by Lua) ** See Copyright Notice in lua.h
56105. ** test whether a string is a reserved word
56106. ** equality for short strings, which are always interned
56107. `s2' cannot be found after that
56108. 0 or more repetitions
56109. end ('\0') of source string
56110. return match(ms, s, ep + 1);
56111. accept empty?
56112. get string library
56113. set table as metatable for strings
56114. ** \$Id: lstrlib.c,v 1.178.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Standard library for string operations and pattern-matching ** See Copyright Notice in lua.h
56115. may overflow?
56116. ===== ** PATTERN MATCHING **
=====

56117. not found
56118. end of pattern?
56119. add capture to accumulated result
56120. remove result from 'luaL_tolstring'
56121. 1st char is already checked
56122. using goto's to optimize tail recursion
56123. start
56124. position capture?
56125. {
56126. to store the format ('%...')
56127. end capture
56128. pop metatable
56129. no suffix
56130. return match(ms, s + 1, ep);
56131. }=====

56132. macro to `unsign' a character
56133. no; go to default
56134. ** LUA_INTFRMLEN is the length modifier for integer conversions in ** 'string.format'; LUA_INTFRM_T is the integer type corresponding to ** the previous length
56135. 0 or more repetitions (minimum)
56136. ** LUA_FLTFRMLEN is the length modifier for float conversions in ** 'string.format'; LUA_FLTFRM_T is the float type corresponding to ** the previous length
56137. skip it
56138. format item
56139. points to optional suffix
56140. avoids a negative 'l'
56141. add whole match
56142. ** add length modifier into formats
56143. balanced string?
56144. may have more after \0
56145. maximum recursion depth for 'match'
56146. recursive function
56147. matches any char
56148. deprecated option
56149. points to what is next
56150. maximum size of each formatted item (> len(format("%99.99f", -1e308)))
56151. keep original text
56152. pop dummy string
56153. **comment:** no precision and string is too long to be formatted; keep original string
 label: code-design
56154. to put formatted item
56155. empty match? go at least one position
56156. empty interval; return no values
56157. return match(ms, s, p + 4);
56158. string ends out of balance
56159. 1 or more repetitions
56160. else fail (s == NULL)
56161. check whether pattern has no special characters
56162. to search for a '*s2' inside 's1'
56163. init of source string
56164. number of strings pushed
56165. do a plain search
56166. return match(ms, s, p + 2)
56167. else return match(ms, s, ep + 1);

56168. pattern class plus optional suffix
56169. reasonable limit to avoid arithmetic overflow
56170. matched once
56171. empty strings are everywhere
56172. skip escapes (e.g. `%'`)
56173. return match(ms, s, ep);
56174. go through
56175. non empty match?
56176. optional
56177. ** maximum size of each format specification (such as '%-099.99d') ** (+10 accounts for %99.99x plus margin of error)
56178. %%
56179. skip precision
56180. start after string's end?
56181. pattern has a special character
56182. end (`\0') of pattern
56183. copy table
56184. ms->level == 0, too
56185. no special chars found
56186. skip width
56187. ** maximum number of captures that a pattern can do during ** pattern-matching. This limit is arbitrary.
56188. **comment:** avoid empty 'memcpy' (may be expensive)
 label: code-design
56189. does not match at least once?
56190. translate a relative string position: negative means back from end
56191. first n-1 copies (followed by separator)
56192. last copy (not followed by separator)
56193. 1 match already done
56194. end
56195. skip ESC
56196. fail
56197. match failed
56198. add result to accumulator
56199. table to be metatable for strings
56200. undo capture
56201. total number of captures (finished or unfinished)
56202. look for a `]'
56203. handle optional suffix
56204. else didn't match; reduce 1 repetition to try again
56205. ====== ** STRING FORMAT **
=====
56206. (2 digits at most)
56207. valid flags in a format specification
56208. start capture
56209. ** Open string library
56210. (size_t -> int) overflow?
56211. escaped sequences not in the format class[*+?-]?
56212. skip the `^'
56213. number of substitutions
56214. keeps trying to match with the maximum repetitions
56215. is the `'\$ the last char in pattern?
56216. explicit request or no special characters?
56217. 1st char will be checked by `memchr'
56218. check end of string
56219. capture results (%0-%9)?
56220. number of bytes in added item
56221. try with one more repetition
56222. counts maximum expand for item
56223. }
56224. frontier?
56225. skip flags
56226. match failed?
56227. metatable.__index = string
56228. cannot find anything
56229. number of arguments
56230. also treat cases 'pnLlh'
56231. skip anchor character
56232. '+' or no suffix
56233. LUA_TNUMBER or LUA_TSTRING
56234. close capture
56235. control for recursive depth (to avoid C stack overflow)
56236. correct 'l1' and 's1' to try again
56237. nil or false?
56238. dummy string
56239. shrink array
56240. cannot find a free place?
56241. i is zero or a present index
56242. ** \$Id: ltable.c,v 2.72.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua tables (hash) ** See Copyright Notice in lua.h
56243. **comment:** key may be dead already, but it is ok to use it in 'next'
 label: code-design
56244. no more elements to count
56245. compute new size for array part
56246. check whether 'key' is somewhere in the chain
56247. grow table
56248. no hash?
56249. count as such
56250. count extra key
56251. that's it
56252. main position is taken?

56253. copy colliding node into free pos. (mp->next also goes)
56254. redo the chain with `n' in place of `mp'
56255. is `key' inside array part?
56256. yes; that's the index (corrected to C)
56257. key index in hash table
56258. is `key' an appropriate array index?
56259. ** hash for lua_Numbers
56260. try first array part
56261. value
56262. then hash part
56263. use specialized version
56264. doesn't need barrier/invalidate cache, as entry was already present in the table
56265. 2^i
56266. now `mp' is free
56267. adjust upper limit
56268. optimal size (till now)
56269. all elements smaller than n will go to array part
56270. index is int?
56271. new node will go into free position
56272. first iteration
56273. ** max size of array part is 2^{MAXBITS}
56274. count elements in range ($2^{(\lg-1)}$, 2^{\lg})
56275. ** main search function
56276. array part must grow?
56277. else must find a boundary in hash part
56278. re-insert elements from hash part
56279. array part must shrink?
56280. use unsigned to avoid overflows
56281. ** beware: when using this function you probably need to check a GC ** barrier and invalidate the TM cache.
56282. table was built with bad purposes: resort to linear search
56283. key not found
56284. **comment:** ** for some types, it is better to avoid modulus by power of 2, as ** they tend to have many 2 factors.
label: code-design
56285. is colliding node out of its main position?
56286. now it has its hash
56287. could not find a free place
56288. find previous
56289. ** search function for short strings
56290. total number of elements
56291. for each slice
56292. 'key' did not match some condition
56293. number of elements to go to array part
56294. else go through
56295. find original element
56296. count to traverse all array keys
56297. reset counts
56298. save old hash ...
56299. optimal size for array part
56300. all elements already counted
56301. resize the table to new computed sizes
56302. ** search function for integers
56303. all those keys are integer keys
56304. colliding node is in its own main position
56305. 2^{\lg}
56306. key
56307. a non-nil value?
56308. counter
56309. count keys in array part
56310. how do a binary search between them
56311. chain new position
56312. use common `dummynode'
56313. summation of `nums'
56314. ** inserts a new key into a hash table; first, check whether key's main ** position is free. If not, check whether colliding node is in its main ** position or not: if it is not, move colliding node to an empty place and ** put new key in its main position; otherwise (colliding node is in its main ** position), new key goes to an empty position.
56315. overflow?
56316. ** returns the index of a `key' for table traversals. First goes all ** elements in the array part, then elements in the hash part. The ** beginning of a traversal is signaled by -1.
56317. no elements to hash part?
56318. must be a positive value
56319. all positions are free
56320. re-insert elements from vanishing slice
56321. hash elements are numbered after array ones
56322. find `i' and `j' such that i is present and j is not
56323. there is a boundary in the array part: (binary) search for it
56324. more than half elements present?
56325. ** ======
56326. that is easy...
56327. hash part is empty?
56328. **comment:** ** Implementation of tables (aka arrays, objects, or hash tables). ** Tables keep its elements in two parts: an array part and a hash part. ** Non-negative integer keys are all candidates to be kept in the array ** part. The actual size of the array is the largest `n' such that at ** least half the slots between 0 and n are in use. ** Hash uses a mix of chained scatter table with Brent's variation. ** A main invariant of these tables is that, if an element is not ** in its main position (i.e. the `original' position that its hash gives ** to it), then the colliding element is in its own main position. ** Hence even when the load factor reaches 100%, performance remains good.
label: code-design
56329. insert key into grown table
56330. ** {===== ** Rehash **
=====

56331. `nums[i]` = number of keys with $2^{i-1} < k \leq 2^i$
56332. whatever called 'newkey' take care of TM cache and GC barrier
56333. $(1 \leq \text{key} \&\& \text{key} \leq \text{t}-\text{sizearray})$
56334. create new hash part with appropriate size
56335. handle INT_MIN
56336. free old array
56337. number of elements smaller than 2^i
56338. yes; move colliding node into free position
56339. count keys in hash part
56340. ** returns the index for 'key' if 'key' is an appropriate key to live in ** the array part of the table, -1 otherwise.
56341. ** Try to find a boundary in table 't'. A 'boundary' is an integer index ** such that $t[i]$ is non-nil and $t[i+1]$ is nil (and 0 if $t[1]$ is nil).
56342. no more elements
56343. ** returns the 'main' position of an element in a table (that is, the index ** of its hash value)
56344. get a free place
56345. ** \$Id: ltable.h,v 2.16.1.2 2013/08/30 15:49:41 roberto Exp \$ ** Lua tables (hash) ** See Copyright Notice in lua.h
56346. returns the key, given the value of a table entry
56347. insert new element at the end
56348. remove $a[j]$
56349. ** {===== ** Pack/unpack **
=====**
56350. -2 to compensate function and 'a'
56351. at least one element?
56352. ** {===== ** Quicksort ** (based on 'Algorithms in MODULA-3', Robert Sedgewick; **
Addison-Wesley, 1993.) ** =====**
56353. $t[i] = t[i-1]$
56354. result = $t[\text{pos}]$
56355. repeat --j until $a[j] \leq P$
56356. call recursively the smaller one
56357. called with only 2 arguments
56358. empty range
56359. $t.n$ = number of elements
56360. move table into index 1
56361. remove value
56362. number of elements to pack
56363. $a < b?$
56364. Pivot
56365. assume array is smaller than 2^{40}
56366. $t[\text{pos}] = \text{nil}$
56367. move up elements
56368. assign other elements
56369. repeat the routine for the larger one
56370. -1 to compensate function
56371. push arg[i + 1...e]
56372. first key
56373. adjust so that smaller half is in [j..i] and larger one in [l..u]
56374. is there a 2nd argument?
56375. 2nd argument is the position
56376. $t[\text{pos}] = t[\text{pos}+1]$
56377. $a[i] < a[l]?$
56378. add last value (if interval was not empty)
56379. remove $a[l]$
56380. pop pivot, $a[i], a[j]$
56381. $a[l] \leq P == a[u-1] \leq a[u]$, only need to sort from $l+1$ to $u-2$
56382. return table
56383. only 3 elements
56384. for tail recursion
56385. $a[l..i-1] \leq a[i] == P \leq a[i+1..u]$
56386. first empty element
56387. ** \$Id: ltlib.c,v 1.65.1.2 2014/05/07 16:32:55 roberto Exp \$ ** Library for Table Manipulation ** See Copyright Notice in lua.h
56388. swap pivot ($a[u-1]$) with $a[i]$
56389. invariant: $a[l..i] \leq P \leq a[j..u]$
56390. sort elements $a[l], a[(l+u)/2]$ and $a[u]$
56391. insert first element
56392. }=====**
56393. swap $a[l] - a[u]$
56394. $a[u] < a[l]?$
56395. $a[u] < a[i]?$
56396. number of elements minus 1
56397. only 2 elements
56398. function?
56399. create result table
56400. where to insert new element
56401. validate 'pos' if given
56402. push arg[i] (avoiding overflow problems)
56403. $_G.\text{unpack} = \text{table.unpack}$
56404. repeat ++i until $a[i] \geq P$
56405. remove $a[i]$
56406. make sure there is two arguments
56407. $t[\text{pos}] = v$
56408. these last two cases are used for tests only
56409. never collect these names
56410. **comment:** ** function to be used with macro "fasttm": optimized for absence of ** tag methods
 label: code-design
56411. cache this fact
56412. no tag method?
56413. ** \$Id: ltm.c,v 2.14.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Tag methods ** See Copyright Notice in lua.h
56414. ORDER TM
56415. ** \$Id: ltm.h,v 2.11.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Tag methods ** See Copyright Notice in lua.h

56416. * WARNING: if you change the order of this enumeration, * grep "ORDER TM"
56417. number of elements in the enum
56418. last tag method with 'fast' access
56419. open libraries
56420. ** lua_stdin_is_tty detects whether the standard input is a 'tty' (that ** is, whether we're running lua interactively).
56421. number of arguments to the script
56422. remove line
56423. to be available to 'laction'
56424. remove it
56425. execute arguments -e and -l
56426. count total number of arguments
56427. get line
56428. no concatenated argument?
56429. invalid option; return its index...
56430. any result to print?
56431. -v
56432. add a new line...
56433. mark in error messages for incomplete statements
56434. non-empty line?
56435. try alternative name
56436. is there an error object?
56437. -i option?
56438. create state
56439. line ends with newline?
56440. -E
56441. stop if file fails
56442. -e
56443. assume stdin is a tty
56444. go through
56445. call 'emain' in protected mode
56446. show prompt
56447. not an option?
56448. no next argument or it is another option
56449. remove result from 'get_prompt'
56450. stop collector during initialization
56451. executes stdin as a file
56452. ** \$Id: lua.c,v 1.206.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua stand-alone interpreter ** See Copyright Notice in lua.h
56453. function index
56454. number of 'has_**'
56455. indices of various argument indicators in array args
56456. error running LUA_INIT
56457. option
56458. if another SIGINT happens before lstop, terminate process (default action)
56459. join them
56460. no more input?
56461. try its 'tostring' metamethod
56462. ** lua_readline defines how to show a prompt and then read a line from ** the standard input. ** lua_saveline defines how to "save" a read line in a "history". **
lua_freeline defines how to free a line read by lua_readline.
56463. execute main script (if there is one)
56464. add it to history
56465. signal no errors
56466. first line starts with '=' ?
56467. option '-E'?
56468. ...between the two lines
56469. **comment:** unused arg.
 label: code-design
56470. signal for libraries to ignore env. vars.
56471. stdin
56472. repeat until gets a complete line
56473. both options need an argument
56474. push traceback function
56475. -i
56476. call 'require(name)'
56477. try next 'arg'
56478. check that argument has no extra characters at the end
56479. the next function is called unprotected, so it must avoid errors
56480. cannot try to add lines?
56481. no arguments?
56482. force a complete garbage collection in case of errors
56483. put it under chunk and args
56484. else...
56485. invalid arg?
56486. change it to `return'
56487. ...as a negative value
56488. clear stack
56489. 2nd argument
56490. global[name] = require return
56491. get result
56492. 1st argument
56493. remove traceback function
56494. collect arguments
56495. no input
56496. open standard libraries
56497. ***** * Copyright (C) 1994-2015 Lua.org, PUC-Rio. * * Permission is
hereby granted, free of charge, to any person obtaining * a copy of this software and associated documentation files (the * "Software"), to deal in the Software
without restriction, including * without limitation the rights to use, copy, modify, merge, publish, * distribute, sublicense, and/or sell copies of the Software, and to
* permit persons to whom the Software is furnished to do so, subject to * the following conditions: * * The above copyright notice and this permission notice shall
be * included in all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

56498. (t)
56499. minimum Lua stack available to a C function
56500. ** functions that read/write blocks when loading/dumping Lua chunks
56501. ** basic types
56502. ** push functions (C -> stack)
56503. (u) number of parameters
56504. Functions to be called by the debugger in specific events
56505. mark for precompiled code ('<esc>Lua')
56506. unsigned integer type
56507. ** prototype for memory-allocation functions
56508. ** access functions (stack -> C)
56509. ** garbage-collection function and options
56510. (S)
56511. ** set functions (stack -> Lua)
56512. (I)
56513. ** state manipulation
56514. predefined values in the registry
56515. active function
56516. (n) 'global', 'local', 'field', 'method'
56517. ** Event codes
56518. ** \$Id: lua.h,v 1.285.1.4 2015/02/21 14:04:50 roberto Exp \$ ** Lua - A Scripting Language ** Lua.org, PUC-Rio, Brazil (<http://www.lua.org>) ** See Copyright Notice at the end of this file
56519. private part
56520. ** ===== ** Debug API **
=====
56521. type of numbers in Lua
56522. type for integer functions
56523. (u)
56524. ** get functions (Lua -> stack)
56525. ** pseudo-indices
56526. (u) number of upvalues
56527. ** 'load' and 'call' functions (load and run Lua code)
56528. option for multiple returns in 'lua_pcall' and 'lua_call'
56529. ** coroutine functions
56530. **comment:** ** generic extra include file
 label: code-design
56531. ** Comparison and arithmetic functions
56532. ** Event masks
56533. =====
56534. ** miscellaneous functions
56535. ** basic stack manipulation
56536. (n)
56537. activation record
56538. (S) 'Lua', 'C', 'main', 'tail'
56539. thread status
56540. ** RCS ident string
56541. ORDER TM
56542. ** ===== ** some useful macros **
=====
56543. lua.hpp Lua header files for C++ <<extern "C">> not supplied automatically because Lua also compiles as C++
56544. show version
56545. end of options; use stdin
56546. list bytecodes?
56547. cannot happen
56548. default program name
56549. end of options; skip it
56550. default output file
56551. unknown option
56552. ** \$Id: luac.c,v 1.69 2011/11/29 17:46:33 lhf Exp \$ ** Lua compiler (saves bytecodes to files; also list bytecodes) ** See Copyright Notice in lua.h
56553. actual program name
56554. default output file name
56555. strip debug information?
56556. end of options; keep it
56557. list
56558. actual output file name
56559. output file
56560. parse only
56561. strip debug information
56562. dump bytecodes?
56563. ** \$Id: print.c,v 1.69 2013/07/04 01:03:46 lhf Exp \$ ** print bytecodes ** See Copyright Notice in lua.h
56564. ** ===== ** Search for "@@" to find all configurable definitions. **
=====
56565. **comment:** @@ The following macros supply trivial compatibility for some ** changes in the API. The macros themselves document how to ** change your code to avoid using them.
 label: documentation
56566. @@ LUA_IDSIZE gives the maximum size for the description of the source @* of a function in debug information. ** CHANGE it if you want a different size.
56567. reserve some space for error handling
56568. pentium 64 bits?
56569. @@ LUA_INTEGER is the integral type used by lua_pushinteger/lua_tointeger. ** CHANGE that if ptrdiff_t is not adequate on your machine. (On most ** machines, ptrdiff_t gives a good choice between int or long.)
56570. ** The next definitions activate some tricks to speed up the ** conversion from doubles to integer types, mainly to LUA_UNSIGNED. ** @@ LUA_MSASMTRICK uses Microsoft assembler to avoid clashes with a ** DirectX idiosyncrasy. ** @@ LUA_IEEE754TRICK uses a trick that should work on any machine ** using IEEE754 with a 32-bit integer type. ** @@ LUA_IEEELL extends the trick to LUA_INTEGER; should only be ** defined when

LUA_INTEGER is a 32-bit integer. ** @@ LUA_IIEEEENDIAN is the endianness of doubles in your machine ** (0 for little endian, 1 for big endian); if not defined, Lua will ** check it dynamically for LUA_IIEEE754TRICK (but not for LUA_NANTRICK). ** @@ LUA_NANTRICK controls the use of a trick to pack all types into ** a single double value, using NaN values to represent non-number ** values. The trick only works on 32-bit machines (ints and pointers ** are 32-bit values) with numbers represented as IEEE 754-2008 doubles ** with conventional endianness (12345678 or 87654321), in CPUs that do ** not produce signaling NaN values (all NaNs are quiet).

56571. @@ LUA_API is a mark for all core API functions. @@ LUALIB_API is a mark for all auxiliary library functions. @@ LUAMOD_API is a mark for all standard library opening functions. ** CHANGE them if you need to define those functions in some special way. ** For instance, if you want to create one Windows DLL with the core and ** the libraries, you may want to use the following definition (define ** LUA_BUILD_AS_DLL to get it).

56572. assume 'strtod' handles hex formats

56573. @@ lua_str2number converts a decimal numeric string to a number. @@ lua_strx2number converts an hexadecimal numeric string to a number. ** In C99, 'strtod' does both conversions. C89, however, has no function ** to convert floating hexadecimal strings to numbers. For these ** systems, you can leave 'lua_strx2number' undefined and Lua will ** provide its own implementation.

56574. @@ LUA_ENV is the name of the variable that holds the current @@ environment, used to access global names. ** CHANGE it if you do not like this name.

56575. does not need -ldl

56576. {

56577. @@ LUAI_MAXSHORTLEN is the maximum length for short strings, that is, ** strings that are internalized. (Cannot be smaller than reserved words ** or tags for metamethods, as these strings must be internalized; ** #("function") = 8, #("__newindex") = 10.)

56578. @@ LUAI_MAXSTACK limits the size of the Lua stack. ** CHANGE it if you need a different limit. This limit is arbitrary; ** its only purpose is to stop Lua from consuming unlimited stack ** space (and to reserve some numbers for pseudo-indices).

56579. }=====

56580. ** {===== @@ LUA_NUMBER is the type of numbers in Lua. ** CHANGE the following definitions only if you want to build Lua ** with a number type different from double. You may also need to ** change lua_number2int & lua_number2integer. ** =====

56581. @@ LUA_NUMBER_SCAN is the format for reading numbers. @@ LUA_NUMBER_FMT is the format for writing numbers. @@ lua_number2str converts a number to a string. @@ LUAI_MAXNUMBER2STR is maximum size of previous conversion.

56582. 16 digits, sign, point, and \0

56583. }

56584. @@ LUA_DIRSEP is the directory separator (for submodules). ** CHANGE it if your machine does not use "/" as the directory separator ** and is not Windows. (On Windows Lua automatically uses "\").

56585. @@ The luai_num* macros define the primitive operations over numbers.

56586. pentium 32 bits?

56587. @@ LUA_COMPAT_MAXN defines the function 'maxn' in the table library.

56588. @@ LUA_PATH_DEFAULT is the default path that Lua uses to look for @* Lua libraries. @@ LUA_CPATH_DEFAULT is the default path that Lua uses to look for @* C libraries. ** CHANGE them if your machine has a non-conventional directory ** hierarchy or if you want to install your libraries in ** non-conventional directories.

56589. @@ luai_writestring/luai_writeline define how 'print' prints its results. ** They are only used in libraries and the stand-alone program. (The #if ** avoids including 'stdio.h' everywhere.)

56590. avoid overflows in comparison

56591. @@ LUA_QL describes how error messages quote program elements. ** CHANGE it if you want a different appearance.

56592. @@ LUAI_FUNC is a mark for all extern functions that are not to be @* exported to outside modules. @@ LUAI_DDEF and LUAI_DDEC are marks for all extern (const) variables @* that are not to be exported to outside modules (LUAI_DDEF for @* definitions and LUAI_DDEC for declarations). ** CHANGE them if you need to mark them in some special way. Elf/gcc ** (versions 3.2 and later) mark them as "hidden" to optimize access ** when Lua is compiled as a shared library. Not all elf targets support ** this attribute. Unfortunately, gcc does not offer a way to check ** whether the target offers that support, and those without support ** give a warning about it. To avoid these warnings, change to the ** default definition.

56593. @@ l_mathop allows the addition of an 'l' or 'f' to all math operations

56594. @@ LUA_USE_POSIX includes all functionality listed as X/Open System @* Interfaces Extension (XSI). ** CHANGE it (define it) if your system is XSI compatible.

56595. @@ LUA_COMPAT_LOG10 defines the function 'log10' in the math library. ** You can rewrite 'log10(x)' as 'log(x, 10)'.

56596. @@ LUA_COMPAT_MODULE controls compatibility with previous ** module functions 'module' (Lua) and 'lual_register' (C).

56597. the following operations need the math library

56598. these are quite standard operations

56599. @@ LUA_UNSIGNED is the integral type used by lua_pushunsigned/lua_tounsigned. ** It must have at least 32 bits.

56600. more often than not the libs go together with the core

56601. Microsoft compiler on a Pentium (32 bit) ?

56602. @@ LUA_ANSI controls the use of non-ansi features. ** CHANGE it (define it) if you want Lua to avoid the use of any ** non-ansi feature or library.

56603. ** Some tricks with doubles

56604. @@ LUA_COMPAT_UNPACK controls the presence of global 'unpack'. ** You can replace it with 'table.unpack'.

56605. ** \$Id: luacnf.h,v 1.176.1.2 2013/11/21 17:26:16 roberto Exp \$ ** Configuration file for Lua ** See Copyright Notice in lua.h

56606. assume 'printf' handles 'A' specifiers

56607. {

56608. assume support for long long

56609. enable goodies for regular Windows platforms

56610. =====

56611. 16-bit ints

56612. **comment:** needs some extra libraries

label: code-design

56613. @@ LUA_COMPAT_LOADSTRING defines the function 'loadstring' in the base ** library. You can rewrite 'loadstring(s)' as 'load(s)'.

56614. @@ LUA_INT32 is a signed integer with exactly 32 bits. @@ LUAI_UMEM is an unsigned integer big enough to count the total @* memory used by Lua. @@ LUAI_MEM is a signed integer big enough to count the total memory @* used by Lua. ** CHANGE here if for some weird reason the default definitions are not ** good enough for your machine. Probably you do not need to change ** this.

56615. @@ LUAI_UACNUMBER is the result of an 'usual argument conversion' @* over a number.

56616. ** In Windows, any exclamation mark (!) in the path is replaced by the ** path of the directory of the executable file of the current process.

56617. **comment:** needs an extra library: -lreadline

label: code-design

56618. empty

56619. assume IEEE754 and a 32-bit integer type

56620. @@ macro 'luac_pcall' emulates deprecated function lua_cpcall. ** You can call your C function directly (with light C functions).

56621. **comment:** @@ luai_writestringerror defines how to print error messages. ** (A format string with one argument is enough for Lua...)
label: code-design

56622. ** {===== ** Compatibility with previous versions ** =====

56623. @@ LUAI_BITSINT defines the number of bits in an int. ** CHANGE here if Lua cannot automatically detect the number of bits of ** your machine. Probably you do not need to change this.

56624. int has at least 32 bits

56625. @@ LUA_COMPAT_LOADERS controls the presence of table 'package.loaders'. ** You can replace it with 'package.searchers'.

56626. **comment:** needs an extra library: -ldl

label: code-design

56627. @@ LUAI_BUFFERSIZE is the buffer size used by the lauxlib buffer system. ** CHANGE it if it uses too much C-stack space.

56628. @@ LUA_COMPAT_ALL controls all compatibility options. ** You can define it to get all options, or change specific options ** to fit your specific needs.
56629. ** Local configuration. You can use this space to add your redefinitions ** without modifying the main part of the file.
56630. open all previous libraries
56631. ** \$Id: lualib.h,v 1.43.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua standard libraries ** See Copyright Notice in lua.h
56632. is lua_Number integral?
56633. * make header for precompiled chunks * if you change the code below be sure to update LoadHeader and FORMAT above * and LUAC_HEADERSIZE in lundump.h
56634. ** load precompiled chunk
56635. empty
56636. **comment:** the code below must be consistent with the code in luaU_header
 label: code-design
56637. first char already read
56638. ** \$Id: lundump.c,v 2.22.1.1 2013/04/12 18:48:47 roberto Exp \$ ** load precompiled Lua chunks ** See Copyright Notice in lua.h
56639. this is the official format
56640. remove trailing '\0'
56641. endianness
56642. make header; from lundump.c
56643. dump one chunk; from ldump.c
56644. data to catch conversion errors
56645. load one chunk; from lundump.c
56646. size in bytes of header of binary files
56647. ** \$Id: lundump.h,v 1.39.1.1 2013/04/12 18:48:47 roberto Exp \$ ** load precompiled Lua chunks ** See Copyright Notice in lua.h
56648. first try 'le'
56649. true must be 1 !!
56650. remove new frame
56651. try first operand
56652. update internal index...
56653. restart luaV_execute over new Lua function
56654. no previous entry; must create one. (The next test is always true; we only need the assignment.)
56655. 'ci' still the called one
56656. call TM
56657. ** create a new Lua closure, push it in the stack, and initialize ** its upvalues. Note that the call to 'luaC_barrierproto' must come ** before the assignment to 'p->cache', as the function needs the ** original value of that field.
56658. undo decrement to zero
56659. ** finish execution of an opcode interrupted by an yield
56660. metamethod may yield only when called from Lua code
56661. will try TM
56662. repeat until only 1 result left
56663. last stack slot filled by 'precall'
56664. there is a metamethod
56665. same metatables => same metamethods
56666. anchor new closure in stack
56667. enter a new line
56668. both strings longer than 'len'; go on comparing (after the '\0')
56669. else primitive len
56670. same metamethods?
56671. increment index
56672. when jump back (loop), or when
56673. collect total length
56674. metamethod? break switch to call it
56675. WARNING: several calls may realloc the stack and invalidate 'ra'
56676. or no TM?
56677. first element to concatenate
56678. reset count
56679. no metamethod; is there a previous entry in the table?
56680. undo increment (resume will increment it again)
56681. **comment:** ** check whether cached closure in prototype 'p' may be reused, that is, ** whether there is a cached closure with the same upvalues needed by ** new closure to be created.
 label: code-design
56682. restore top
56683. invocation via reentry: continue execution
56684. else will try the metamethod
56685. l is finished?
56686. cached closure
56687. B == 0?
56688. no metamethod
56689. is there a cached closure?
56690. second operand is empty?
56691. fill in its upvalues
56692. correct top (in case of previous open call)
56693. push function
56694. close all upvalues from previous call
56695. reentry point when frame changes (call/return)
56696. ** \$Id: lvm.c,v 2.155.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua virtual machine ** See Copyright Notice in lua.h
56697. top is one after last element (at top-2)
56698. l is smaller than r (because r is not finished)
56699. called frame
56700. try second operand
56701. result is not nil?
56702. result is first operand
56703. adjust results
56704. **comment:** needs more space?
 label: code-design
56705. continue loop?
56706. do not call hook again (VM yielded, so it did not move)
56707. else repeat with 'tm'
56708. are there elements to concat?
56709. Lua function

56710. metamethod should not be called when operand is K
56711. limit for table tag-method chains (to avoid loops)
56712. popped 'n' strings and pushed one
56713. no match?
56714. upvalue refers to local variable?
56715. invert result
56716. top when 'call_binTM' was called
56717. correct top
56718. mark the end of concat operands
56719. 't' is a table?
56720. if has result, move it to its place
56721. push cashed closure
56722. **comment:** wrong upvalue; cannot reuse closure
 label: code-design
56723. 3rd argument
56724. call linehook when enter a new function,
56725. no metamethod and (now) there is an entry with given key
56726. move new frame into old one
56727. nresults >= 0?
56728. did hook yield?
56729. function was tail called
56730. to be used after possible stack reallocation
56731. func. + 2 args (state and index)
56732. create a new one
56733. protect stack below results
56734. strings are equal up to a '\0'
56735. jump back
56736. caller frame
56737. r is finished?
56738. concat all strings
56739. for test instructions, execute the jump instruction that follows it
56740. ...and external index
56741. try metamethod
56742. 'luav_concat' may invoke TMs and move the stack
56743. pre-allocate it at once
56744. ** some macros for common tasks in 'luaV_execute'
56745. put TM result in proper position
56746. call base
56747. called hook last time?
56748. no TM?
56749. assign new value to that entry
56750. go to next instruction
56751. finish its execution
56752. erase mark
56753. result is second op.
56754. get all var. arguments
56755. at least two non-empty string values; get as many as possible
56756. tail call: put called frame (n) in place of caller one (o)
56757. concat them (may yield again)
56758. mark that it yielded
56759. skip next instruction (if C)
56760. previous value is nil; must check the metamethod
56761. called function
56762. not a table; check metamethod
56763. execute a jump instruction
56764. yet to concatenate
56765. call count hook
56766. call line hook
56767. "<=" using "<" instead?
56768. save control variable
56769. no metamethod?
56770. else try `lt'
56771. got 'n' strings to create 1 new
56772. nb = no break
56773. main loop of interpreter
56774. if previous value is not nil, there must be a previous entry in the table; moreover, a metamethod has no relevance
56775. caller function
56776. check whether it has right upvalues
56777. interrupted instruction
56778. C function?
56779. correct base
56780. get upvalue from enclosing function
56781. do a primitive get
56782. index of first '\0' in both strings
56783. return cached closure (or NULL if no cached closure)
56784. save it on cache for reuse
56785. skip jump instruction
56786. limit of live values
56787. ** equality of Lua values. L == NULL means raw equality (no metamethods)
56788. else previous instruction set top
56789. else will try the tag method
56790. 2nd argument
56791. condition failed?
56792. void
56793. no result? 'p3' is third argument
56794. external invocation: return
56795. 1st argument
56796. number of elements handled in this pass (at least 2)

56797. previous call may change the stack
56798. next assignment may change this value
56799. move final result to final position
56800. not to called directly
56801. ** \$Id: lvm.h,v 2.18.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Lua virtual machine ** See Copyright Notice in lua.h
56802. ** \$Id: lzio.c,v 1.35.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Buffered streams ** See Copyright Notice in lua.h
56803. try to read more
56804. no bytes in buffer?
56805. discount char being returned
56806. luaZ_fill consumed first byte; put it back
56807. no more input; return number of missing bytes
56808. min. between n and z->n
56809. -----
56810. ----- read ---
56811. end of stream
56812. current position in buffer
56813. Lua state (for reader)
56814. ----- Private Part -----
56815. reader function
56816. additional data
56817. bytes still unread
56818. read next n bytes
56819. ** \$Id: lzio.h,v 1.26.1.1 2013/04/12 18:48:47 roberto Exp \$ ** Buffered streams ** See Copyright Notice in lua.h
56820. Last change: revised for Lua 5.3.3
56821. function to be called
56822. calls Lua
56823. number of results
56824. </code>: floor division
56825. %
56826. table to be indexed
56827. second result
56828. uses 'key' (at index -2) and 'value' (at index -1)
56829. call 'f' with 3 arguments and 1 result
56830. =====
56831. set global 'a'
56832. first key
56833. Last change: revised for Lua 5.3.3
56834. first result
56835. number of arguments
56836. ’ | ‘^’ | ‘%’ |
56837. code 1
56838. </code>) is a division
56839. table is in the stack at index 't'
56840. \$Id: manual.of,v 1.162 2016/05/30 15:57:03 roberto Exp \$
56841. </code>
56842. push result of t.x (2nd arg)
56843. **comment:** not used
 label: code-design
56844. 1st argument
56845. remove 't' from the stack
56846. code 2
56847. removes 'value'; keeps 'key' for next iteration
56848. 3rd argument
56849. Last change: revised for Lua 5.3.3
56850. no errors?
56851. it has no upvalues
56852. call to 'tointeger' may change 'n' even if it fails
56853. function to be called
56854. not a closure
56855. save continuation
56856. function can do error recovery
56857. call to 'tonumber' may change 'n' even if it fails
56858. save context
56859. adjust frame top
56860. signal it
56861. stack large enough?
56862. pop value
56863. save information for error recovery
56864. 'luaO_tostring' may create a new string
56865. do a 'conventional' protected call
56866. ** get functions (Lua -> stack)
56867. no continuation or no yieldable?
56868. data to 'f_call'
56869. remove second operand
56870. ** Reverse the stack segment from 'from' to 'to' ** (auxiliary to 'lua_rotate')
56871. test for pseudo index
56872. set global table as 1st upvalue of 'f' (may be LUA_ENV)
56873. =1 to signal that it did an actual step
56874. end of prefix
56875. yes; check is OK
56876. push empty string
56877. end of cycle?
56878. GC values are expressed in Kbytes: #bytes/2^10
56879. stack already checked by previous 'api_check'
56880. may call tag method
56881. 'subtract' index (index is negative)
56882. does it have an upvalue?
56883. C closure

56884. ** access functions (stack -> C)
56885. internal copy's address
56886. ** set functions (stack -> Lua)
56887. upvalues
56888. add 'data' to total debt
56889. Lua closure
56890. not a C function
56891. get global table from registry
56892. reverse the prefix with length 'n'
56893. no
56894. light C function?
56895. allow GC to run
56896. else n == 1; nothing to do
56897. ** \$Id: lapi.c,v 2.259 2016/02/29 14:27:14 roberto Exp \$ ** Lua API ** See Copyright Notice in lua.h
56898. remove key
56899. lock done by caller
56900. pop value and key
56901. save value of 'allowhook'
56902. **comment:** to avoid warnings
 label: code-design
56903. just do the call
56904. end of stack segment being rotated
56905. push 'str' (to make it a TValue)
56906. does not need barrier because closure is white
56907. ** push functions (C -> stack)
56908. ensure that true is 1
56909. need to prepare continuation?
56910. previous call may reallocate the stack
56911. avoid ridiculous low values (and 0)
56912. corresponding test
56913. reverse the suffix
56914. ** Pushes on the stack a string with given length. Avoid using 's' when ** 'len' == 0 (as 's' can be NULL in that case), due to later use of ** 'memcmp' and 'memcpy'.
56915. pop index and value
56916. not convertible?
56917. unlock done in 'auxsetstr'
56918. for unary operations, add fake 2nd operand
56919. all other operations expect two operands
56920. can grow without overflow?
56921. ** 'load' and 'call' functions (run Lua code)
56922. ** miscellaneous functions
56923. ** to be called by 'lua_checkstack' in protected mode, to grow stack ** capturing memory errors
56924. value at a non-valid index
56925. test for valid but not pseudo index
56926. code unreachable; will unlock when control actually leaves the kernel
56927. start of segment
56928. ** convert an acceptable stack index into an absolute index
56929. if it is here, there were no errors
56930. get newly created function
56931. invalid option
56932. to do a "small" step
56933. prepare continuation (call is already protected by 'resume')
56934. negative index
56935. do the call
56936. no continuation or no yieldable
56937. get its upvalue pointer
56938. no; need to grow stack
56939. ** Garbage-collection function
56940. first operand at top - 2, second at top - 1; result go to top - 2
56941. ** Execute a protected call.
56942. reverse the entire segment
56943. restore previous state
56944. test for upvalue
56945. try to grow stack
56946. lua closure
56947. ** t[k] = value at the top of the stack (where 'k' is a string)
56948. no more elements
56949. ** basic stack manipulation
56950. ** Let x = AB, where A is a prefix of length 'n'. Then, ** rotate x n == BA. But BA == (A^r . B^r)^r.
56951. LUA_REGISTRYINDEX does not need gc barrier (collector revisits it before finishing collection)
56952. function upvalue?
56953. ** \$Id: lapi.h,v 2.9 2015/03/06 19:49:50 roberto Exp \$ ** Auxiliary functions from Lua API ** See Copyright Notice in lua.h
56954. ** Again, the use of 'lua_pushvfstring' ensures this function does ** not need reserved stack space when called. (At worst, it generates ** an error with "stack overflow" instead of the given message.)
56955. ** use appropriate macros to interpret 'pclose' return status
56956. name already in use?
56957. double buffer size
56958. remove previous table
56959. remove table (but keep name)
56960. no more pre-read characters
56961. copy to be left at top
56962. remove pushed values
56963. false, because did not find table there
56964. remove both metatables
56965. get/create library table
56966. ** Stripped-down 'require': After checking "loaded" table, calls 'openf' ** to open a module, registers the result in 'package.loaded' table and, ** if 'glb' is true, also registers the result in the global table. ** Leaves resulting module on the top.
56967. userdata to box arbitrary data

56968. special name for messages
56969. not big enough?
56970. not found
56971. creating metatable?
56972. push prefix
56973. get a new reference
56974. table already there
56975. read initial portion
56976. return true/nil,what,code
56977. remove table and value
56978. type of termination
56979. add line to correct line numbers
56980. no metafield?
56981. remove it from list
56982. remove this nil
56983. no stack frame?
56984. remove field
56985. argument to open function
56986. allocation error?
56987. {
56988. no _LOADED[modname]?
56989. _LOADED[modname]
56990. name for the type of the actual argument
56991. remove only metatable
56992. (t[freelist] = t[ref])
56993. binary file?
56994. **comment:** ** The use of 'lua_pushfstring' ensures this function does not ** need reserved stack space when called.
label: code-design
56995. }=====** reads the first character of file 'f' and skips an optional BOM mark ** in its beginning plus its first line if it starts with '#'. Returns ** true if it skipped the first line. In any case, 'cp' has the ** first "valid" character of the file (after the optional BOM and ** a first-line comment).
56997. push last suffix
56998. re-read initial portion
56999. ** {===== ** Userdata's metatable manipulation **
=====** Reference system **
57000. start 'next' loop
57001. no metatable?
57002. ** {=====** Reference system **
=====** Userdata's metatable manipulation **
57003. copy upvalues to the top
57004. **comment:** not used
label: code-design
57005. call 'openf' to open module
57006. move name to proper place
57007. 'fread' can return > 0 *and* set the EOF flag. If next call to 'getF' called 'fread', it might still wait for user input. The next check avoids this problem.
57008. remove it from stack
57009. find an upper bound
57010. read a block from file
57011. **comment:** ** check whether buffer is using a userdata on the stack as a temporary ** buffer
label: code-design
57012. move library table to below upvalues
57013. get first free element
57014. remove from stack
57015. to be read by the parser
57016. return problematic part of the name
57017. ** returns a pointer to a free area with at least 'sz' bytes
57018. reopen in binary mode
57019. remove value (but keep name)
57020. put value below buffer
57021. push function
57022. ** set functions from list 'l' into table at top - 'nup'; each ** function gets the 'nup' elements at the top as upvalues. ** Returns with only the table at the stack.
57023. name start with '_G.'?
57024. successful termination?
57025. get info about it
57026. leave previous value on top, but return 0
57027. no free elements
57028. create metatable
57029. for each pair in table
57030. no comment
57031. index of filename on the stack
57032. use it
57033. skip end-of-line, if present
57034. ref = t[freelist]
57035. size of the second part of the stack
57036. value is a userdata?
57037. delete old buffer
57038. there was a comment
57039. index of free-list header
57040. continue after 'p'
57041. file being read
57042. field has a non-table value?
57043. number of pre-read characters
57044. ** search for 'objidx' in table at index -1. ** return 1 + string at top if find a good name.
57045. use the given type name
57046. are there pre-read characters to be read?
57047. t[ref] = t[freelist]
57048. main?
57049. is there a name from code?

57050. return next character
57051. 'c' is the first character of the stream
57052. set new table into field
57053. _LOADED[modname] = module
57054. return to Lua to abort
57055. remove _LOADED table
57056. assign new table to field
57057. check function at level
57058. calls to Lua API may change this value
57059. return metatable type
57060. create larger buffer
57061. prefix matched; discard it
57062. try recursively
57063. add a '...'
57064. new table for field
57065. {freelist} = ref
57066. for Lua functions, use <file:line>
57067. push replacement in place of pattern
57068. do not count 'self'
57069. UTF-8 BOM mark
57070. free buffer
57071. avoid 'memcpy' when 's' can be NULL
57072. _G[modname] = module
57073. ** {===== ** Argument check functions **
=====}
57074. metatable._gc = boxgc
57075. check numeric types
57076. ** This file uses only the official API of Lua. ** Any function declared here could be written as an application function.
57077. any free element?
57078. ** {===== ** Traceback **
=====}
57079. does it have a metatable?
57080. 'nil' has a unique fixed reference
57081. ** Search for a name for a function in all loaded modules ** (registry._LOADED).
57082. ignore results from 'lua_load'
57083. remove function and global table
57084. no buffer yet
57085. try global variable (and create one if it does not exist)
57086. place '.' between the two names
57087. do a binary search
57088. ** {===== ** Generic Buffer manipulation **
=====}
57089. remove object
57090. first line is a comment (Unix exec. file)?
57091. standard name
57092. registry.name = metatable
57093. ** {===== ** Load functions **
=====}
57094. read block
57095. **comment:** too many levels?
 label: code-design
57096. remove original name
57097. interpret result
57098. value is a userdata with wrong metatable
57099. skip first line
57100. remove upvalues
57101. _LOADED[modname] = new table
57102. found object?
57103. value is not a userdata with a metatable
57104. remove metatable and metatable
57105. not enough space?
57106. ignore non-string keys
57107. and skip to last ones
57108. remove previous result
57109. ** Count number of elements in a luaL_Reg list.
57110. close file (even in case of errors)
57111. get correct metatable
57112. push name without prefix
57113. ** ensure that stack[idx][fname] has a table and push that table ** into the stack
57114. return them (chars already in buffer)
57115. ** {===== ** Error-report functions **
=====}
57116. }
57117. ** {===== ** Compatibility with 5.1 module functions **
=====}
57118. remove value
57119. is metatable nil?
57120. else, no information available...
57121. make copy of module (call result)
57122. size of the first part of the stack
57123. **comment:** ** Ensures the stack has at least 'space' extra slots, raising an error ** if it cannot fulfill the request. (The error handling needs a few ** extra slots to format the error message. In case of an error without ** this extra space, Lua will generate the same 'stack overflow' error, ** but without 'msg'.)
 label: code-design
57124. area for reading file
57125. get _LOADED table
57126. copy of module
57127. error is in the self argument itself?
57128. try first a global name

57129. not the same?
57130. copy original content
57131. nothing left..
57132. closure with those upvalues
57133. metatable.__name = tname
57134. error?
57135. package not already loaded?
57136. ** \$Id: lauxlib.c,v 1.286 2016/01/08 15:33:09 roberto Exp \$ ** Auxiliary functions for building Lua libraries ** See Copyright Notice in lua.h
57137. **comment:** remove its header from the stack
 label: code-design
57138. no such field?
57139. ** Find or create a module table with a given name. The function ** first looks at the _LOADED table and, if that fails, try a ** global variable with that name. In any case, leaves on the stack ** the module table.
57140. is there info?
57141. remove name
57142. fill the table with given functions
57143. no op
57144. extra error code for 'luaL_load'
57145. print a newline and flush the output
57146. print a string
57147. stream (NULL for incompletely created streams)
57148. ** A file handle is a userdata with metatable ' LUA_FILEHANDLE' and ** initial structure 'luaL_Stream' (it may contain other fields ** after that initial structure).
57149. print an error message
57150. }=====** "Abstraction Layer" for basic report of messages and errors
57151. ** {===== ** Generic Buffer manipulation **
57152. predefined references
57153. to close stream (NULL for closed streams)
57154. buffer address
57155. initial buffer
57156. }=====** Compatibility with deprecated conversions **
57157. ** \$Id: lauxlib.h,v 1.129 2015/11/23 11:29:43 roberto Exp \$ ** Auxiliary functions for building Lua libraries ** See Copyright Notice in lua.h
57158. ** {===== ** some useful macros **
57159. }=====** File handles for IO library **
57160. ** {===== ** Continuation function for 'pcall' and 'xpcall'. Both functions ** already pushed a 'true' before doing the call, so in case of success ** 'finishpcall' only has to return everything in the stack minus ** 'extra' values (where 'extra' is exactly the number of items to be ** ignored).
57161. number of characters in buffer
57162. buffer size
57163. ** {=====** File handles for IO library **
57164. compatibility with old module system
57165. ** =====** some useful macros **
57166. function to be called
57167. set global _VERSION
57168. there must be a condition
57169. ** Reader for generic 'load' function: 'lua_load' uses the ** stack for internal stuff, so the reader cannot change the ** stack top. Instead, it keeps its resulting string in a ** reserved slot inside the stack.
57170. and initial value
57171. invalid numeral
57172. argument must be a table
57173. remove it
57174. function
57175. loading from a reader function
57176. no numbers as strings
57177. skip trailing spaces
57178. return nil plus error message
57179. return false, msg
57180. first result
57181. }=====**
57182. return all arguments
57183. leave only message (default if no other one)
57184. set global _G
57185. **comment:** not used
 label: code-design
57186. state
57187. pop result
57188. get 3 values from metamethod
57189. ** Reader for generic 'load' function: 'lua_load' uses the ** stack for internal stuff, so the reader cannot change the ** stack top. Instead, it keeps its resulting string in a ** reserved slot inside the stack.
57190. successful conversion to number
57191. ** 'ipairs' function. Returns 'ipairsaux', given "table", 0. ** (The given "table" may not be a table.)
57192. no metatable
57193. 'env' index or 0 if no 'env'
57194. not a number
57195. condition is true?
57196. return all results
57197. open lib into global table
57198. iteration function
57199. argument 'self' to metamethod
57200. **comment:** to avoid warnings
 label: code-design
57201. ** Traversal function for 'ipairs'
57202. value to print
57203. call 'error'
57204. default message
57205. else not a number

57206. no digit?
57207. create reserved slot
57208. ** Do a protected call with error handling. After 'lua_rotate', the ** stack will have <f, err, true, f, [args...]>; so, the function passes ** 2 to 'finishpcall' to skip the 2 first values when returning results.
57209. handle signal
57210. put it in place
57211. move them below function's arguments
57212. initial value
57213. add extra information
57214. loading a string?
57215. ** \$Id: lbaselib.c,v 1.313 2016/04/11 19:18:40 roberto Exp \$ ** Basic library ** See Copyright Notice in lua.h
57216. placeholders
57217. check error function
57218. state,
57219. returns either __metatable field (if present) or metatable
57220. only to match 'lua_Kfunction' prototype
57221. create a 2nd argument if there isn't one
57222. put before error message
57223. call it
57224. standard conversion?
57225. error
57226. 'env' parameter?
57227. yes; return it
57228. no metamethod?
57229. set it as 1st upvalue
57230. skip initial spaces
57231. ** reserved slot, above all arguments, to hold a copy of the returned ** string to avoid it being collected while parsed. 'load' has four ** optional arguments (chunk, source name, mode, and environment).
57232. error (message is on top of the stack)
57233. number of arguments
57234. **comment:** remove 'env' if not used by previous call
 label: code-design
57235. error message
57236. error?
57237. already a number?
57238. environment for loaded function
57239. save string in reserved slot
57240. get result
57241. ** {===== ** Generic Read function **
=====**
57242. get function
57243. first result (false)
57244. first result if no errors
57245. will return generator,
57246. number of bits to consider in a number
57247. ** a lua_Unsigned with its first LUA_NBITS bits equal to 1. (Shift must ** be made in two parts to avoid problems when LUA_NBITS is equal to the ** number of bits in a lua_Unsigned.)
57248. ** \$Id: lbitlib.c,v 1.30 2015/11/11 19:08:09 roberto Exp \$ ** Standard library for bitwise operations ** See Copyright Notice in lua.h
57249. builds a number with 'n' ones (1 <= n <= LUA_NBITS)
57250. shift right?
57251. arithmetic shift for 'negative' number
57252. **comment:** ** get field and width arguments for field-manipulation functions, ** checking whether they are valid. ** ('luaL_error' called without 'return' to avoid later warnings about ** 'width' being used uninitialized.)
 label: code-design
57253. {
57254. add signal bit
57255. macro to trim extra bits
57256. avoid undefined shift of LUA_NBITS when i == 0
57257. shift left
57258. }{
57259. i = d % NBITS
57260. }
57261. position after whole expression
57262. nothing to concatenate?
57263. ** Traverse a list of tests, patching their destination address and ** registers: tests producing values jump to 'vtarget' (and put their ** values in 'reg'), other tests jump to 'dtarget'.
57264. ** Add constant 'v' to prototype's list of constants (field 'k'). ** Use scanner's table to cache position of constants in constant list ** and try to reuse constants. Because some values should not be used ** as keys (nil cannot be a key, integer keys can collapse with float ** keys), the caller must provide a useful 'key' for indexing the cache.
57265. ** Generate code to store result of expression 'ex' into variable 'var'.
57266. jump when false
57267. ** Fix jump instruction at position 'pc' to jump to 'dest'. ** (Jump addresses are relative in Lua)
57268. everything else is valid
57269. correct value? (warning: must distinguish floats from integers!)
57270. ** Ensures final expression result is either in a register or it is ** a constant.
57271. l = max(l, pl)
57272. put this jump in 't' list
57273. all those operations are relocatable
57274. ** Emit code for binary expressions that "produce values" ** (everything but logical operators 'and'/'or' and comparison ** operators). ** Expression to produce final result will be encoded in 'e1'.
57275. not found
57276. 't' is in an upvalue
57277. instruction will put result in 'reg'
57278. remove previous OP_NOT
57279. false list jumps to here (to go through)
57280. ** Path all jumps in 'list' to jump to 'target'. ** (The assert means that we cannot fix a jump to a forward address ** because we only know addresses once code is generated.)
57281. put new instruction in code array

57282. non-numeric operands or not safe to fold
57283. 'l1' points to 'l2'
57284. no more jumps to here
57285. ** Process 1st operand 'v' of binary operation 'op' before reading ** 2nd operand.
57286. insert new jump in 't' list
57287. from = min(from, pfrom)
57288. R/K index for key
57289. ** returns current 'pc' and marks it as a jump target (to avoid wrong ** optimizations with consecutive instructions not in the same basic block).
57290. ** Emit a "load constant" instruction, using either 'OP_LOADK' ** (if constant index 'k' fits in 18 bits) or an 'OP_LOADKX' ** instruction with "extra argument".
57291. ** Fix an expression to return the number of results 'nresults'. ** Either 'e' is a multi-ret expression (function call or vararg) ** or 'nresults' is LUA_MULTRET (as any expression can satisfy that).
57292. ** Ensures final expression result is in a valid R/K index ** (that is, it is either in a register or in 'k' with an index ** in the range of R/K indices). ** Returns R/K index.
57293. point to itself represents end of list
57294. use boolean itself as key
57295. result is already in a register
57296. already returns 1 value
57297. ** Create a jump instruction and return its position, so its destination ** can be fixed later (with 'fixjump'). If there are jumps to ** this position (kept in 'jpc'), link them all together so that ** 'patchlistaux' will fix all them directly to the final destination.
57298. ** Add an integer to list of constants and return its index. ** Integers use userdata as keys to avoid collision with floats with ** same value; conversion to 'void*' is used only for hashing, so there ** are no "precision" problems.
57299. position of an eventual LOAD false
57300. no original list?
57301. no jumps?
57302. can relocate its simple result
57303. mark "here" as a jump target
57304. ** Ensures final expression result (including results from its jump ** lists) is in register 'reg'. ** If expression has jumps, need to patch these jumps either to ** its final position or to "load" instructions (for those tests ** that do not produce values).
57305. ** Emit a SETLIST instruction. ** 'base' is register that keeps table; ** 'nelems' is #table plus those to be stored now; ** 'tostore' is number of values (in registers 'base + 1',...) to add to ** table (or LUA_MULTRET to add up to stack top).
57306. ** Traverse a list of tests ensuring no one produces a value
57307. ** Add elements in 'list' to list of pending jumps to "here" ** (current position)
57308. ** Emit code to go through if 'e' is true, jump otherwise.
57309. jump to default target
57310. base register for op_self
57311. already jump if true
57312. ** Free register used by expression 'e' (if any)
57313. turn offset into absolute position
57314. ** Return false if folding can raise an error. ** Bitwise operations need operands convertible to integers; division ** operations cannot have 0 as divisor.
57315. else keep numeral, which may be folded with 2nd operand
57316. always false; do nothing
57317. does operation
57318. '(a > b)' ==> '(b < a)'; '(a >= b)' ==> '(b <= a)'
57319. get previous range
57320. index scanner table
57321. ** Gets the destination address of a jump instruction. Used to traverse ** a list of jumps.
57322. use string itself as key
57323. jump when it is false
57324. 'target' is current position?
57325. constant not found; create a new entry
57326. those instructions may be jump targets
57327. Maximum number of registers in a Lua function (must fit in 8 bits)
57328. there is one value available (somewhere)
57329. insert new jump in false list
57330. ** Add a string to list of constants and return its index.
57331. ** check whether list has any jump that do not produce a value ** or produce an inverted value
57332. ** Add a boolean to list of constants and return its index.
57333. reg. is not a local?
57334. pc of new jump
57335. true list jumps to here (to go through)
57336. invert operands
57337. fake 2nd operand
57338. get a register
57339. not a constant in the right range: put it in a register
57340. save list of jumps to here
57341. add list to pending jumps
57342. register where 'e' was placed
57343. function and 'self' produced by op_self
57344. can connect both?
57345. move constants to 'k'
57346. FALLTHROUGH
57347. ** Add nil to list of constants and return its index.
57348. self expression has a fixed register
57349. free registers with list values
57350. ** Code a 'return' instruction
57351. ** Code a "conditional jump", that is, a test or comparison opcode ** followed by a jump. Return jump position.
57352. ** Ensure all pending jumps to current position are fixed (jumping ** to current position with no values) and reset list of pending ** jumps
57353. operand must be on the 'stack'
57354. ** Finalize code for binary operation, after reading 2nd operand. ** For '(a .. b .. c)' (which is '(a .. (b .. c))', because ** concatenation is right associative), merge second CONCAT into first ** one.
57355. ** Ensures expression value is in any register.
57356. ** Returns the position of the instruction "controlling" a given ** jump (that is, its condition), or the jump itself if it is ** unconditional.
57357. argument is +1 to reserve 0 as non-op
57358. ** Negate condition 'e' (where 'e' is a comparison).
57359. else go through
57360. ** Format and emit an 'iABC' instruction. (Assertions check consistency ** of parameters versus opcode.)
57361. is there an index there?
57362. conversion errors

57363. ** Ensures expression value is in register 'reg' (and therefore ** 'e' will become a non-relocatable expression).
57364. ** Ensures final expression result (including results from its jump ** lists) is in some (any) register and return that register.
57365. ** Ensures final expression result (including results from its jump ** lists) is in next available register.
57366. 'pc' will change
57367. use number itself as key
57368. '==', '<', '<=' use their own opcodes
57369. expression already has a register?
57370. ** Create a OP_LOADNIL instruction, but try to optimize: if the previous ** instruction is also OP_LOADNIL and ranges are compatible, adjust ** range of previous instruction instead of emitting a new one. (For ** instance, 'local a; local b' will generate a single opcode.)
57371. list closed by 'luK_infix'
57372. go ahead only if 'v' is true
57373. move value to some (pending) register
57374. position of an eventual LOAD true
57375. otherwise, use next available register
57376. put value there
57377. put final result in it
57378. ** Fix an expression to return one result. ** If expression is not a multi-ret expression (function call or ** vararg), it already returns one result, so nothing needs to be done. ** Function calls become VNONRELOC expressions (as its result comes ** fixed in the base register of the call), while vararg expressions ** become VRELOCABLE (as OP_VARARG puts its results where it wants). ** (Calls are created returning one result, so that does not need ** to be fixed.)
57379. is 't' in a register?
57380. last register to set nil
57381. save jump position
57382. ** Emit code for unary expressions that "produce values" ** (everything but 'not'). ** Expression to produce final result will be encoded in 'e'.
57383. no jumps to current position?
57384. save corresponding line information
57385. '(a ~= b)' ==> 'not (a == b)'
57386. reuse index
57387. jump if true
57388. ** Apply prefix operation 'op' to expression 'e'.
57389. expression is an open function call?
57390. ** \$Id: lcode.c,v 2.109 2016/05/13 19:09:21 roberto Exp \$ ** Code generator for Lua ** See Copyright Notice in lua.h
57391. else no optimization
57392. ** Try to "constant-fold" an operation; return 1 iff successful. ** (In this case, 'e1' has the final result.)
57393. expression itself is a test?
57394. keep them on hold
57395. ** Free registers used by expressions 'e1' and 'e2' (if any) in proper ** order.
57396. ** Create expression 't[k]'. 't' must have its final result already in a ** register or upvalue.
57397. ** Ensures final expression result is either in a register or in an ** upvalue.
57398. register or upvalue index
57399. ** Emit code for comparisons. ** 'e1' was already put in R/K form by 'luaK_infix'.
57400. ** Reserve 'n' registers in register stack
57401. go ahead only if 'v' is false
57402. no fixed register yet?
57403. not a numeral
57404. true == not nil == not false
57405. ** Format and emit an 'iABx' instruction.
57406. ** Emit instruction to jump if 'e' is 'cond' (that is, if 'cond' ** is true, code will jump if 'e' is true.) Return jump position. ** Optimize when 'e' is 'not' something, inverting the condition ** and removing the 'not'.
57407. ** Code 'not e', doing constant folding.
57408. constant fits in 'argC'?
57409. find last element
57410. compute 'ex' into proper place
57411. invalid var kind to store
57412. opcodes operate only on registers
57413. ** Patch destination register for a TESTSET instruction. ** If instruction in position 'node' is not a TESTSET, return 0 ("fails"). ** Otherwise, if 'reg' is not 'NO_REG', set it as the destination ** register. Otherwise, change instruction to a simple 'TEST' (produces ** no register value)
57414. ** Change line information associated with current position.
57415. end of list
57416. ** Emit code to go through if 'e' is false, jump otherwise.
57417. last element links to 'l2'
57418. ** Path all jumps in 'list' to close upvalues up to given 'level' ** (The assertion checks that jumps either were closing nothing ** or were closing higher levels, from inner blocks.)
57419. ** Emit instruction 'i', checking for array sizes and saving also its ** line information. Return 'i' position.
57420. cannot happen
57421. values are useless when negated
57422. generate opcode
57423. already in a register
57424. condition?
57425. interchange true and false lists
57426. ** Free register 'reg', if it is neither a constant index nor ** a local variable.)
57427. ** Add a float to list of constants and return its index.
57428. becomes a non-relocatable value
57429. nothing to do...
57430. ** Check register-stack level, keeping track of its maximum size ** in field 'maxstacksize'
57431. previous is LOADNIL
57432. cannot patch other instructions
57433. ** Ensure that expression 'e' is not a variable.
57434. false == not "x" == not 0.5 == not 1 == not true
57435. cannot use nil as key; instead use table itself to represent nil
57436. numerical value does not need GC barrier; table has no metatable, so it does not need to invalidate cache
57437. **comment:** ** Emit an "extra argument" instruction (format 'iAx')
 label: code-design
57438. no register to put value or register already has the value; change instruction to simple test
57439. folds neither NaN nor 0.0 (to avoid problems with -0.0)
57440. always true; do nothing
57441. both operands are "RK"
57442. division by 0
57443. result has fixed position

57444. ** If expression is a numeric constant, fills 'v' with its value ** and returns 1. Otherwise, returns 0.
57445. ** Concatenate jump-list 'l2' into jump-list 'l1'
57446. ** Emit SELF instruction (convert expression 'e' into 'e:key(e,)').
57447. ** Marks the end of a patch list. It is an invalid value both as an absolute ** address, and as a list link (would link an element to itself).
57448. ** grep "ORDER OPR" if you change these enums (ORDER OP)
57449. ** \$Id: lcode.h,v 1.64 2016/01/05 16:22:37 roberto Exp \$ ** Code generator for Lua ** See Copyright Notice in lua.h
57450. get (pointer to) instruction of given 'expdesc'
57451. does it have frames?
57452. error flag
57453. move function from L to NL
57454. initial state
57455. it is running
57456. propagate error
57457. some error occurred
57458. ** \$Id: lcorolib.c,v 1.10 2016/04/11 19:19:55 roberto Exp \$ ** Coroutine Library ** See Copyright Notice in lua.h
57459. return true + 'resume' returns
57460. move error message
57461. remove results anyway
57462. error object is a string?
57463. move function to top
57464. return false + error message
57465. add extra info
57466. move yielded values
57467. EOZ
57468. 7.
57469. 1.
57470. 6.
57471. }
57472. 3.
57473. ** \$Id: lctype.c,v 1.12 2014/11/02 19:19:04 roberto Exp \$ ** 'ctype' functions for Lua ** See Copyright Notice in lua.h
57474. d.
57475. o.
57476. a.
57477. b.
57478. 9.
57479. e.
57480. {
57481. 5.
57482. 2.
57483. c.
57484. f.
57485. 4.
57486. 8.
57487. **comment:** ASCII case: can use its own tables; faster and fixed
 label: code-design
57488. ** use standard C ctypes
57489. must use standard C ctype
57490. ** this 'tolower' only works for alphabetic characters
57491. {
57492. two more entries for 0 and -1 (EOZ)
57493. ** WARNING: the functions defined here do not necessarily correspond ** to the similar functions in the standard C ctype.h. They are ** optimized for the specific needs of Lua
57494. }{
57495. }
57496. ** add 1 to char to allow index -1 (EOZ)
57497. ** \$Id: lctype.h,v 1.12 2011/07/15 12:50:29 roberto Exp \$ ** 'ctype' functions for Lua ** See Copyright Notice in lua.h
57498. ** 'lalpha' (Lua alphabetic) and 'lalnum' (Lua alphanumeric) both include '_'
57499. ** Variations of 'lua_settable', used by 'db_getinfo' to put results ** from 'lua_getinfo' into result table. Key is always a string; ** value can be a string, an int, or a boolean.
57500. setmetatable(hooktable) = hooktable
57501. non-string 'msg'?
57502. call hook function
57503. stack level
57504. push name
57505. hooktable[L1] = new Lua hook
57506. external hook?
57507. no hook?
57508. 2nd result = mask
57509. create a hook table
57510. out of range?
57511. return 1st argument
57512. value (hook function)
57513. ** If L1 != L, L1 can be in any state, and therefore there are no ** guarantees about its stack space; any push in L1 must be ** checked.
57514. return it untouched
57515. **comment:** ** Auxiliary function used by several library functions: check for ** an optional thread as function's first argument and set 'arg' with ** 1 if this argument is present (so that functions can skip it to ** access their other arguments)
 label: code-design
57516. push local name
57517. push event name
57518. push function
57519. 1st result = hooktable[L1]
57520. level out of range
57521. key (thread)
57522. move local value
57523. info about a function?
57524. exchange object and table
57525. turn off hooks
57526. no metatable

57527. put object into table
57528. ** get (if 'get' is true) or set an upvalue from a closure
57529. upvalue index
57530. is there a hook function?
57531. **comment:** ** Convert a string mask (for 'sethook') into a bit mask
 label: code-design
57532. remove eventual returns
57533. closure
57534. function argument?
57535. return table
57536. re-order
57537. remove hook table
57538. function will operate over current thread
57539. ** In function 'db_getinfo', the call to 'lua_getinfo' may push ** results on the stack; later it creates the result table to put ** these objects. Function 'treatstackoption' puts the result from ** 'lua_getinfo' on top of the result table so that it can call ** 'lua_setfield'.
57540. no name (nor value)
57541. move function to 'L1' stack
57542. hook table must exist
57543. set it in position
57544. 3rd result = count
57545. add '>' to 'options'
57546. ** Convert a bit mask (for 'gethook') into a string mask
57547. stack-level argument
57548. no-op if get is false
57549. ** Call hook function registered at hook table for the current ** thread (if there is one)
57550. push current line
57551. local-variable index
57552. pop value (if not popped by 'lua_setlocal')
57553. ** The hook table at registry[&HOOKKEY] maps threads to their current ** hook function. (We only need the unique address of 'HOOKKEY').
57554. ** \$Id: ldblib.c,v 1.151 2015/11/23 11:29:43 roberto Exp \$ ** Interface from Lua to its debug API ** See Copyright Notice in lua.h
57555. ** Check whether a given upvalue from a given closure exists and ** returns its index
57556. return only name (there is no value)
57557. table to collect results
57558. ** Calls 'lua_getinfo' and collects all results in a new table. ** L1 needs stack space for an optional input (function) plus ** two optional outputs (function and line table) from function ** 'lua_getinfo'.
57559. * hooktable.__mode = "k"
57560. move object to the "main" stack
57561. cannot know who sets that register
57562. other instructions cannot call a function
57563. active function; get information through 'ar'
57564. calling instruction index
57565. table index
57566. move from 'b' to 'a'
57567. pop value
57568. not found
57569. no source available; use "?" instead
57570. ** This function can be called asynchronously (e.g. during a signal). ** Fields 'oldpc', 'basehookcount', and 'hookcount' (set by ** 'resethookcount') are for debug only, and it is no problem if they ** get arbitrary values (causes at most one wrong hook call). 'hookmask' ** is an atomic value. We assume that pointers are atomic too (e.g., gcc ** ensures that for all platforms where it runs). Moreover, 'hook' is ** always checked before being called (see 'luaD_hook').
57571. undo decrement to zero
57572. update 'jmptarget'
57573. ORDER OP
57574. for iterator
57575. 'name' already filled
57576. is a local?
57577. ** Checks whether value 'o' came from an upvalue. (That can only happen ** with instructions OP_GETTABUP/OP_SETTABUP, which operate directly on ** upvalues.)
57578. enter a new line
57579. no line hook and count != 0; nothing to be done
57580. get name for 'b'
57581. when jump back (loop), or when
57582. correct before option 'L', which can raise a mem. error
57583. ======

57584. ** Error when both values are convertible to numbers, but not to integers
57585. no reasonable name found
57586. generic name for any valid slot
57587. ** find a "name" for the RK value 'c'
57588. affect all registers above base
57589. if Lua function, add source:line information
57590. reset count
57591. check whether 'o' is an upvalue
57592. undo increment (resume will increment it again)
57593. was it called inside a hook?
57594. no 'standard' name?
57595. ** ====== ** Symbolic Execution ** ======

57596. current position sets that register
57597. information about non-active function?
57598. push function
57599. 'c' is a register
57600. name of indexed variable
57601. turn off hooks?
57602. do not call hook again (VM yielded, so it did not move)
57603. key index
57604. ** If function yielded, its 'func' can be in the 'extra' field. The ** next function restores 'func' to its correct value for debugging ** purposes. (It exchanges 'func' and 'extra'; so, when called again, ** after debugging, it also "re-restores" ** 'func' to its altered value).
57605. **comment:** to avoid warnings
 label: code-design

57606. affect all regs above its base
57607. no such level
57608. calling instruction
57609. ORDER TM
57610. generic name for any vararg
57611. not a Lua function?
57612. call linehook when enter a new function,
57613. Active Lua function (given call info)
57614. level found?
57615. no name
57616. did hook yield?
57617. protect stack below results
57618. for all lines with code
57619. skip the '>'
57620. calling function
57621. literal constant?
57622. all other instructions can call only through metamethods
57623. any code before this address is conditional
57624. search for 'c'
57625. get function name
57626. pop function
57627. boolean 'true' to be the value of all indices
57628. else no reasonable name found
57629. move argument
57630. set registers from 'a' to 'a+b'
57631. no? try a register
57632. ** \$Id: ldebug.c,v 2.120 2016/03/31 19:01:21 roberto Exp \$ ** Debug Interface ** See Copyright Notice in lua.h
57633. access to vararg values?
57634. assume EXTRA_STACK
57635. go through to return NULL
57636. called hook last time?
57637. new table to store active lines
57638. first operand is wrong?
57639. handled by lua_getinfo
57640. now second is wrong
57641. ** try to find last instruction before 'lastpc' that modified register 'reg'
57642. invalid (negative) level
57643. call it
57644. erase mark
57645. could find instruction?
57646. mark that it yielded
57647. no such vararg
57648. consider live variables at function start (parameters)
57649. jump is forward and do not skip 'lastpc'?
57650. is there an error handling function?
57651. invalid option
57652. found a constant name?
57653. exchange its 'func' and 'extra' values
57654. calling function is a known Lua function?
57655. call count hook
57656. call line hook
57657. any instruction that set A
57658. could not find reasonable name
57659. table[line] = true
57660. is 'n' inside 'ci' stack?
57661. add src:line information to 'msg'
57662. ** The subtraction of two potentially unrelated pointers is ** not ISO C, but it should not crash a program; the subsequent ** checks are ISO C and ensure a correct result.
57663. else try symbolic execution
57664. format message
57665. is code conditional (inside a jump)?
57666. get function that yielded
57667. is 'c' a constant?
57668. push it on stack
57669. it is its own name
57670. keep last instruction that changed 'reg'
57671. ** \$Id: ldebug.h,v 2.14 2015/05/22 17:45:56 roberto Exp \$ ** Auxiliary functions from Debug Interface module ** See Copyright Notice in lua.h
57672. Open a hole inside the stack at 'func'
57673. continue running the coroutine
57674. push error message
57675. error status?
57676. 'pos' part: restore 'p'
57677. finish interrupted instruction
57678. **comment:** ** Do the work for 'lua_resume' in protected mode. Most of the work ** depends on the status of the coroutine: initial state, suspended ** inside a hook, or regularly suspended (optionally with a continuation ** function), plus erroneous cases: non-suspended coroutine or dead ** coroutine.
 label: code-design
57679. correct 'pc'
57680. stack grow uses memory
57681. handle typical cases separately
57682. something in the stack
57683. no results?
57684. run continuation
57685. save context
57686. pushing msg. can break this invariant
57687. }=====

57688. an error occurred?
57689. back to caller
57690. execute down to higher C 'boundary'

57691. do the actual call
57692. Lua function: prepare its call
57693. finish 'lua_callk'/'lua_pcall'; CIST_YPCALL and 'errfunc' already handled
57694. no pending pcall
57695. allow yields
57696. free all CLs (list grew because of an error)
57697. jump to it
57698. save 'p'
57699. close possible pending closures
57700. dynamic structures used by the parser
57701. shrink list
57702. slot ensured by caller
57703. is there a continuation?
57704. no recovery point
57705. restore old error handler
57706. res == final position of 1st result
57707. error message on current top
57708. nothing to move
57709. {
57710. move fixed parameters to final position
57711. ensure minimum stack size
57712. erase original copy (for GC)
57713. is a Lua function?
57714. not in base level?
57715. }=====

57716. ** Recovers from an error in a coroutine. Finds a recover point (if ** there is one) and completes the execution of the interrupted ** 'luaD_pcall'. If there is no recover point, returns zero.
57717. ** Check whether __call metafield of 'func' is a function. If so, put ** it in stack below original 'func' so that 'luaD_precall' can call ** it. Raise an error if __call metafield is not a function.
57718. **comment:** reuse preregistered msg.
 label: code-design
57719. erase new segment
57720. try to get '__call' metamethod
57721. ** \$Id: ldo.c,v 2.151 2015/12/16 16:40:07 roberto Exp \$ ** Stack and Call structure of Lua ** See Copyright Notice in lua.h
57722. complete wanted number of results
57723. number of real arguments
57724. re-throw in main thread
57725. 'oldpc' for caller function
57726. no handler at all; abort
57727. C closure
57728. }{
57729. **comment:** ** Finishes a function call: calls hook if necessary, removes CallInfo, ** moves current number of results to proper place; returns 0 iff call ** wanted multiple (variable number of) results.
 label: code-design
57730. stack overflow?
57731. restore original 'allowhook'
57732. data to 'f_parser'
57733. ** Prepares a function call: checks the stack, creates a new CallInfo ** entry, fills in the relevant information, calls hook if needed. ** If function is a C function, does the call, too. (Otherwise, leave ** the execution ('luaV_execute') to the caller, to allow stackless ** calls.) Returns true iff function has been executed (C function).
57734. hook may change stack
57735. dummy variable
57736. top points after the last result
57737. do not use vararg?
57738. C++ exceptions
57739. does it have a continuation function?
57740. finish 'luaD_precall'
57741. light C function
57742. "finish" luaD_pcall
57743. save "number of non-yieldable" calls
57744. move it to proper place
57745. move all results to correct place
57746. ensure space for metamethod
57747. was inside a pcall?
57748. ** LUAI_THROW/LUAI_TRY define how Lua does exception handling. By ** default, Lua handles errors with exceptions when compiling as ** C++ code, with _longjmp/_setjmp when asked to use them, and with ** longjmp/setjmp otherwise.
57749. Lua function
57750. normal end or yield
57751. ** Call a hook for the given event. Make sure there is a hook to be ** called. (Both 'L->hook' and 'L->hookmask', which triggers this ** function, can be changed asynchronously by signals.)
57752. first argument
57753. restore 'ny'
57754. trying to shrink?
57755. unrecoverable error?
57756. dynamic structure used by the scanner
57757. ** Similar to 'luaD_call', but does not allow yields during the call
57758. ** Executes "full continuation" (everything in the stack) of a ** previously interrupted coroutine until the stack is empty (or another ** interruption long-jumps out of the loop). If the coroutine is ** recovering from an error, 'ud' points to the error status, which must ** be passed to the first continuation function (otherwise the default ** status is LUA_YIELD).
57759. error status can only happen in a protected call
57760. cannot call hooks inside a hook
57761. yielded inside a hook?
57762. memory error?
57763. ** Try to find a suspended protected call (a "recover point") for the ** given thread.
57764. some space for error handling
57765. not handling stack overflow?
57766. protect stack below results
57767. cannot yield during parsing

57768. enough results?
57769. Lua function?
57770. shrink it
57771. unroll continuation
57772. now 'enter' new function
57773. main thread has a handler?
57774. hooks assume 'pc' is already incremented
57775. return to 'luaD_hook'
57776. jump back to 'lua_resume'
57777. error after extra size?
57778. coroutine is in base level; start running it
57779. adjust with nil
57780. panic function?
57781. call continuation
57782. inside a hook?
57783. mark it as dead
57784. number of returns
57785. 'common' yield
57786. assume EXTRA_STACK
57787. first fixed argument
57788. make sure there is a hook
57789. was handling stack overflow?
57790. resuming from previous yield
57791. ISO C handling with long jumps
57792. ** Given 'nres' results at 'firstResult', move 'wanted' of them to 'res'. ** Handle most typical cases (zero results for commands, one result for ** expressions, multiple results for tail calls/single parameters) ** separated.
57793. not enough results; use all of them plus nils
57794. finish 'lua_pcall'
57795. **comment:** in POSIX, try _longjmp/_setjmp (more efficient)
 label: code-design
57796. chain new error handler
57797. complete its execution
57798. chain list of long jump buffers
57799. call it
57800. yield results come from continuation
57801. may be starting a coroutine
57802. continue running after recoverable errors
57803. error while handing stack error
57804. tag method is the new function to be called
57805. ** {===== ** Error-recovery functions **
=====**
57806. macro to check stack size, preserving 'p'
57807. call continuation function
57808. thread has no error handler
57809. complete missing arguments
57810. ** Execute a protected parser.
57811. wanted == LUA_MULTRET
57812. mark thread as 'dead'
57813. move results to proper place
57814. just continue running Lua code
57815. move wanted results to correct place
57816. }
57817. error calling 'lua_resume'?
57818. remove args from the stack
57819. final position of first argument
57820. not a function
57821. search for a pcall
57822. read first character
57823. set status
57824. call panic function (last chance to jump out)
57825. mark that it is running (again)
57826. copy error obj.
57827. ** Call a function (C or Lua). The function to be called is at *func. ** The arguments are on the stack, right after the function. ** When returns, all the results are on the stack, starting at the original ** function position.
57828. must have a continuation and must be able to call it
57829. should be zero to be yieldable
57830. C function?
57831. number of arguments
57832. frame size
57833. starting point
57834. ** Check appropriate error for stack overflow ("regular" overflow or ** overflow while handling stack overflow). If 'nCalls' is larger than ** LUAI_MAXCCALLS (which means it is handling a "regular" overflow) but ** smaller than 9/8 of LUAI_MAXCCALLS, does not report an error (to ** allow overflow handling to work)
57835. must be inside a hook
57836. error code
57837. ** signal an error in the call to 'resume', not in the execution of the ** coroutine itself. (Such errors should not be handled by any coroutine ** error handler and should not kill the coroutine.)
57838. don't change stack (change only for debugging)
57839. ** {===== ** Stack reallocation **
=====**
57840. one result needed
57841. finish 'lua_pcallk' callee
57842. thread has an error handler?
57843. part of stack in use
57844. ** Completes the execution of an interrupted C function, calling its ** continuation function.
57845. now it must be a function
57846. save current 'func'
57847. ** \$Id: ldo.h,v 2.29 2015/12/21 13:02:14 roberto Exp \$ ** Stack and Call structure of Lua ** See Copyright Notice in lua.h

57848. In general, 'pre'/'pos' are empty (nothing to save)
57849. ** Macro to check stack size and grow stack if needed. Parameters ** 'pre'/'pos' allow the macro to preserve a pointer into the ** stack across reallocations, doing the work only when needed. ** 'condmovestack' is used in heavy tests to force a stack reallocation ** at every check.
57850. **comment:** type of protected functions, to be ran by 'runprotected'
label: code-design
57851. no need to save '\0'
57852. include trailing '\0'
57853. ** dump Lua function as precompiled chunk
57854. ** All high-level dumps go through DumpVector; you can change it to ** change the endianness of the result
57855. no debug info or same source as its parent
57856. ** \$Id: ldump.c,v 2.37 2015/10/08 15:53:49 roberto Exp \$ ** save precompiled Lua chunks ** See Copyright Notice in lua.h
57857. link it to the list
57858. current value lives in the stack
57859. now current value lives here
57860. found a corresponding upvalue?
57861. remove from 'open' list
57862. is variable active?
57863. free upvalue
57864. make it closed
57865. ** \$Id: lfunc.c,v 2.45 2014/11/02 19:19:04 roberto Exp \$ ** Auxiliary functions to manipulate prototypes and closures ** See Copyright Notice in lua.h
57866. not found
57867. return it
57868. ** Look for n-th local variable at line 'line' in function 'func'. ** Returns NULL if not found.
57869. ** fill a closure with new closed upvalues
57870. no references?
57871. not found: create a new upvalue
57872. move value to upvalue slot
57873. link it to list of open upvalues
57874. thread not in list of threads with upvalues?
57875. (when open)
57876. linked list
57877. **comment:** mark to avoid cycles with dead threads
label: code-design
57878. the value (when closed)
57879. ** \$Id: lfunc.h,v 2.15 2015/01/13 15:49:11 roberto Exp \$ ** Auxiliary functions to manipulate prototypes and closures ** See Copyright Notice in lua.h
57880. reference counter
57881. ** maximum number of upvalues in a closure (both C and Lua). (Value ** must fit in a VM register.)
57882. ** Upvalues for Lua closures
57883. test whether thread is in 'twups' list
57884. points to stack or to its own value
57885. at this point, all resurrected objects are marked.
57886. is 'curr' dead?
57887. true if table has entry "white-key -> white-value"
57888. "sweep" object
57889. ** barrier that moves collector backward, that is, mark the black object ** pointing to a white object as gray again.
57890. mark any finalizing object left from previous cycle
57891. mark nested protos
57892. is there a weak mode?
57893. ** sweep a list until a live object (or end of list)
57894. **comment:** ** macro to adjust 'pause': 'pause' is actually used like ** 'pause / PAUSEADJ' (value chosen by tests)
label: code-design
57895. remove it
57896. separate all objects with finalizers
57897. minimal debt
57898. push finalizer...
57899. **comment:** may have to clean white keys
label: code-design
57900. keep marked thread with upvalues in the list
57901. make sure gray list is empty
57902. remove 'o' from 'allgc' list
57903. sweep objects to be finalized
57904. ** {===== ** Sweep Functions ** ======**
57905. ** mark an object that can be NULL (either because it is really optional, ** or it was stripped as debug info, or inside an uncompleted structure)
57906. memory traversed in this step
57907. link it to 'fixedge' list
57908. traverse 'grayagain' list
57909. **comment:** unused and unmarked key; remove it
label: code-design
57910. at this point, all strongly accessible objects are marked.
57911. start counting work
57912. should not remove 'sweepgc' object
57913. registry and global metatables may be changed by API
57914. first estimate
57915. return it to 'allgc' list
57916. mark value
57917. mark live elements in the stack
57918. finish sweeps
57919. ** If key is not marked, mark its entry as dead. This allows key to be ** collected, but keeps its entry in the table. A dead node is needed ** when Lua looks up for a key (it may be part of a chain) and when ** traversing a weak table (key might be removed from the table during ** traversal). Other places never manipulate dead keys, because its ** associated nil value is enough to signal that the entry is logically ** empty.
57920. start new collection
57921. ** clear entries with unmarked keys from all weaktables in list 'l' up ** to element 'f'
57922. traverse marked some value?
57923. ** barrier that moves collector forward, that is, mark the white object ** being pointed by a black object. (If in sweep phase, clear the black ** object to white [sweep it] to avoid other barrier calls for this ** same object.)
57924. they will be gray forever
57925. }=====**

57926. must keep invariant?
57927. ** create a new collectable object (with given type and size) and link ** it to 'allgc' list.
57928. strong keys?
57929. link it back to the list
57930. change mark to 'white'
57931. ** If possible, shrink string table
57932. run up to finalizers
57933. remove object from 'allgc' list
57934. current white
57935. ** clear entries with unmarked values from all weaktables in list 'l' up ** to element 'f'
57936. is there an error object?
57937. mark global metatables
57938. nothing more to finalize?
57939. traverse array part
57940. move 'o' to 'finobj' list
57941. stop counting (objects being finalized)
57942. adjust 'estimate'
57943. has to be cleared later
57944. remove from 'gray' list
57945. white-white entry
57946. remove 'cur' from list
57947. strong values?
57948. is there a white value?
57949. markvalue(g, &uvalue);
57950. remove it from hash table
57951. **comment:** remark occasional upvalues of (maybe) dead threads
 label: code-design
57952. remark, to propagate 'resurrection'
57953. GC deficit (be paid now)
57954. is there a finalizer?
57955. entry is empty?
57956. don't bother with it
57957. Clear values from weak tables, before checking finalizers
57958. clear keys from all ephemeron tables
57959. mark that it is out of list
57960. mark running thread
57961. ** Traverse an ephemeron table and link it to proper list. Returns true ** iff any object was marked during this traversal (which implies that ** convergence has to continue). During propagation phase, keep table ** in 'grayagain' list, to be visited again in the atomic phase. In ** the atomic phase, if table has any white->white entry, it has to ** be revisited during ephemeron convergence (as that key may turn ** black). Otherwise, if it has any white key, table has to be cleared ** (in the atomic phase).
57962. ** traverse one gray object, turning it to black (except for threads, ** which are always gray).
57963. ** Traverse a table with weak values and link it to proper list. During ** propagate phase, keep it in 'grayagain' list, to be revisited in the ** atomic phase. In the atomic phase, if table has any white value, ** put it in 'weak' list, to be cleared.
57964. complete counting
57965. ** tells whether a key or value can be cleared from a weak ** table. Non-collectable objects are never removed from weak ** tables. Strings behave as 'values', so are never removed too. for ** other objects: if really collected, cannot keep them; for objects ** being finalized, keep them in keys, but not in values
57966. must retraverse it in atomic phase
57967. go to next element
57968. ** 'makewhite' erases all color bits then sets only the current white ** bit
57969. ** {===== ** Finalization **
=====
57970. ensured by macro luaC_upvalbarrier
57971. ** {===== ** Generic functions **
=====
57972. call remaining finalizers
57973. ** Traverse a prototype. (While a prototype is being build, its ** arrays can be larger than needed; the extra slots are filled with ** NULL, so the use of 'markobjectN')
57974. link table into proper list
57975. do not change stack in emergency cycle
57976. ** mark an object. Userdata, strings, and closed upvalues are visited ** and turned black here. Other objects are marked gray and added ** to appropriate list to be visited (and turned black) later. (Open ** upvalues are already linked in 'headuv' list.)
57977. if there is array part, assume it may have white values (it is not worth traversing it now just to check)
57978. link it in 'finobj' list
57979. ** cost of sweeping one element (the size of a small object divided ** by some adjust for the sweep speed)
57980. estimate must be correct after a full GC cycle
57981. or has no finalizer?
57982. ** advances the garbage collector until it reaches a state allowed ** by 'statemask'
57983. real end of stack
57984. remove it from 'tobefnz' list
57985. sweep main thread
57986. ** \$Id: lgc.c,v 2.212 2016/03/31 19:02:03 roberto Exp \$ ** Garbage Collector ** See Copyright Notice in lua.h
57987. ** Mark all values stored in marked open upvalues from non-marked threads. ** (Values from marked threads were already marked when traversing the ** thread.) Remove from the list threads that no longer have upvalues and ** not-marked threads.
57988. traverse hash part
57989. is really weak?
57990. collect fixed objects
57991. there may be objects to be finalized
57992. ... and its argument
57993. mark its prototype
57994. ** mark metamethods for basic types
57995. value was collected?
57996. **comment:** ** macro to adjust 'stepmul': 'stepmul' is actually used like ** 'stepmul / STEPMULADJ' (value chosen by tests)
 label: code-design
57997. emergency mode or no more finalizers
57998. finish collection
57999. strings are 'values', so are never weak
58000. ** {===== ** Traverse functions **
=====

58001. mark its upvalues
58002. remove value ...
58003. overflow; truncate to maximum
58004. tables may return to this list when traversed
58005. ** if object 'o' has a finalizer, remove it from 'allgc' list (must ** search the list to find it) and link it in 'finobj' list.
58006. repeat until pause or enough "credit" (negative debt)
58007. string table too big?
58008. table has white keys?
58009. table must be cleared
58010. ** sweep at most 'count' elements from a list of GCOBJECTS erasing dead ** objects, where a dead object is one marked with the old (non current) ** white; change all non-dead objects back to white, preparing for next ** collection cycle. Return where to continue the traversal or NULL if ** list is finished.
58011. re-throw error
58012. **comment:** this "white" makes all objects look dead
 label: code-design
58013. save original list
58014. not being collected?
58015. mark objects that will be finalized
58016. thread is not marked or without upvalues
58017. ** mark root set and reset all gray lists, to start a new collection
58018. work is what was traversed by 'atomic'
58019. ** find last 'next' field in list 'p' list (to add elements in its end)
58020. error in __gc metamethod
58021. make table gray (again)
58022. value not marked yet?
58023. call one finalizer
58024. mark it as such
58025. restore state
58026. "sweep" object 'o'
58027. mark its metatable
58028. shrink it a little
58029. change 'sweepgc'
58030. **comment:** remove dead objects from weak tables
 label: code-design
58031. true if an object is marked in this traversal
58032. ** {===== ** Mark functions **
=====**
58033. traverse all finalizable objects
58034. remark upvalue's value
58035. ** Set a reasonable "time" to wait before starting a new GC cycle; cycle ** will start when memory use hits threshold. (Division by 'estimate' ** should be OK: it cannot be zero (because Lua cannot even start with ** less than PAUSEADJ bytes).
58036. else enter next state
58037. flip current white
58038. mark upvalue names
58039. clear values from resurrected weak tables
58040. avoid being called too often
58041. set flag
58042. mark main obj. as white to avoid other barriers
58043. finish propagate phase
58044. separate objects to be finalized
58045. sweep "regular" objects
58046. sweep objects with finalizers
58047. restore hooks
58048. **comment:** threads are never black
 label: code-design
58049. allow cache to be collected
58050. clear not-marked stack slice
58051. ** open upvalues point to values in a thread, so those values should ** be marked when the thread is traversed except in the atomic phase ** (because then the value cannot be changed by the thread and the ** thread may not be traversed again)
58052. ** get GC debt and convert it from Kb to 'work units' (avoid zero debt ** and overflows)
58053. get ephemeron list
58054. ** move all unreachable objects (or 'all' objects) that need ** finalization from list 'finobj' to list 'tobefnz' (to be finalized)
58055. no more gray objects?
58056. ** performs a basic GC step when collector is running
58057. object must be 1st in 'allgc' list!
58058. ** mark all objects in list of being-finalized
58059. mark it now
58060. insert into 'grayagain' list
58061. link at the end of 'tobefnz' list
58062. final traversal?
58063. search for pointer pointing to 'o'
58064. 'remarkupvals' may have removed thread from 'twups' list
58065. remove thread from the list
58066. get first element
58067. else call a few more next time
58068. finish any pending sweep phase to start a new cycle
58069. sweep phase
58070. restart counting
58071. keep table gray
58072. obj. is already marked...
58073. true if table has white keys
58074. overflow?
58075. table has white->white entries?
58076. nothing to traverse now
58077. table will have to be cleared
58078. black objects?
58079. stop counting (do not recount 'grayagain')
58080. not weak
58081. remove 'curr' from 'finobj' list

58082. remove value
58083. all weak
58084. is there still something to sweep?
58085. ** {===== ** GC control **
=====**
58086. can be marked in 'remarckupvals'
58087. no overflow
58088. propagate changes
58089. key is not marked (yet)?
58090. stack not completely built yet
58091. cost of calling one finalizer
58092. avoid GC steps
58093. error while running __gc?
58094. estimate of memory marked by 'atomic'
58095. mark literals
58096. and remove entry from table
58097. ** Performs a full GC cycle; if 'isemergency', set a flag to avoid ** some operations which could change the interpreter state in some ** unexpected ways (running finalizers and shrinking some structures). ** Before running the collection, check 'keepinvariant'; if it is true, ** there may be some objects marked as black, so the collector has ** to sweep all objects to turn them back to white (as white has not ** changed, nothing will be collected).
58098. stop debug hooks during GC metamethod
58099. ** barrier for assignments to closed upvalues. Because upvalues are ** shared among closures, it is impossible to know the color of all ** closures pointing to it. So, we assume that the object being assigned ** must be marked.
58100. mark local-variable names
58101. clear keys from all 'allweak' tables
58102. will have to revisit all ephemeron tables
58103. pause until next cycle
58104. restore invariant
58105. mark key
58106. sweep everything to turn them back to white
58107. ** link collectable object 'o' into list pointed by 'p'
58108. ** Enter first sweep phase. ** The call to 'sweeplist' tries to make pointer point to an object ** inside the list (instead of to the header), so that the real sweep do ** not need to skip objects created between "now" and the start of the ** real sweep.
58109. have to propagate again
58110. ** call a few (up to 'g->gcfignum') finalizers
58111. empty
58112. perform one single step
58113. convert 'work units' to Kb
58114. not running?
58115. maximum number of elements to sweep in each single step
58116. erase 'curr'
58117. ** one after last element in a hash array
58118. ** internal state for collector while inside the atomic phase. The ** collector should never be in this state while running regular code.
58119. update estimate
58120. ** call all pending finalizers
58121. nothing to be done
58122. object is "normal" again
58123. and (next line) call the finalizer
58124. object has been marked for finalization
58125. ** Possible states of the Garbage Collector
58126. ~100 small strings
58127. ** Does one step of collection when debt becomes positive. 'pre'/pos' ** allows some adjustments to be done only when needed. macro ** 'condchangemem' is used only for heavy tests (forcing a full ** GC cycle on every opportunity)
58128. object is white (type 0)
58129. ** \$Id: lgc.h,v 2.91 2015/12/21 13:02:14 roberto Exp \$ ** Garbage Collector ** See Copyright Notice in lua.h
58130. how much to allocate before next GC step
58131. **comment:** ** macro to tell when main invariant (white objects cannot point to black ** ones) must be kept. During a collection, the sweep ** phase may break the invariant, as objects turned white may point to ** still-black objects. The invariant is restored when sweep ends and ** all objects are white again.
label: code-design
58132. **comment:** ** some useful bit tricks
label: code-design
58133. ** Collectable objects may have one of three colors: white, which ** means the object is not marked; gray, which means the ** object is marked, but its references may be not marked; and ** black, which means that the object and all its references are marked. ** The main invariant of the garbage collector, while marking objects, ** is that a black object can never point to a white one. Moreover, ** any gray object must be in a "gray list" (gray, grayagain, weak, ** allweak, ephemeron) so that it can be visited again before finishing ** the collection cycle. These lists have no meaning when the invariant ** is not being enforced (e.g., sweep phase).
58134. object is white (type 1)
58135. bit 7 is currently used by tests (luaL_checkmemory)
58136. neither white nor black
58137. object is black
58138. **comment:** Layout for bit use in 'marked' field:
label: code-design
58139. more often than not, 'pre'/pos' are empty
58140. ** \$Id: linit.c,v 1.38 2015/01/05 13:48:33 roberto Exp \$ ** Initialization of libraries for lua.c and other clients ** See Copyright Notice in lua.h
58141. "require" functions from 'loadedlibs' and set results to global table
58142. ** these libs are loaded by lua.c and are readily available to any Lua ** program
58143. remove lib
58144. ** If you embed Lua in your program and need to open the standard ** libraries, call luaL_openlibs in your program. If you need a ** different set of libraries, copy this file to your project and edit ** it to suit your needs. ** ** You can also *preload* libraries, so that a later 'require' can ** open the library, which is already linked to the application. ** For that, do the following code: ** ** luaL_getsubtable(L, LUA_REGISTRYINDEX, "_PRELOAD"); ** luaL_pushcfunction(L, luaopen_modname); ** luaL_setfield(L, -2, modname); ** luaL_pop(L, 1); // remove _PRELOAD table
58145. create (and set) default files
58146. maximum length of a numeral
58147. current character (look ahead)
58148. ** {===== ** l_fseek: configuration for longer offsets **
=====**
58149. invalid format
58150. use standard output
58151. make sure argument is an open stream

58152. move 'n' and 'toclose' to their positions
58153. ** function to (not) close the standard files stdin, stdout, and stderr
58154. skip if char is '+'
58155. ** Add current char to buffer (if not out of space) and read next one
58156. finish string
58157. ** \$Id: liolib.c,v 2.149 2016/05/02 14:03:19 roberto Exp \$ ** Standard I/O (and system) library ** See Copyright Notice in lua.h
58158. true iff read something
58159. **comment:** ** Change this macro to accept other modes for 'fopen' besides ** the standard ones.
 label: code-design
58160. ensure stack space for all results and for auxlib's buffer
58161. +1 for ending '\0'
58162. ** maximum number of arguments to 'f:lines'/'io.lines' (it + 3 must fit ** in the limit for upvalues of a closure)
58163. close/not close file when finished
58164. at least one argument
58165. read next one
58166. pop new metatable
58167. {
58168. mark file handle as 'closed'
58169. mark stream as closed
58170. number
58171. push metatable
58172. ISO C definitions
58173. number of arguments to read
58174. number of elements in buffer 'buff'
58175. ** {===== ** l_popen spawns a new process connected to the current ** one through the file streams. ** ======**
58176. push arguments to 'g_read'
58177. }=====**
58178. ** functions for 'io' library
58179. try to read 'n' chars
58180. read file in chunks of LUAL_BUFFERSIZE bytes
58181. metatable.__index = metatable
58182. file
58183. save current char
58184. read entire file
58185. optimization: could be done exactly as for strings
58186. add file to module
58187. no-op when c == EOF
58188. fractional part
58189. ** function to close 'popen' files
58190. add ending newline to result
58191. ** When creating file handles, always creates a 'closed' file handle ** before opening the actual file; so, if there is a memory error, the ** handle is in a consistent state.
58192. to traverse/check mode
58193. ** methods for file handles
58194. 'n' is number of results
58195. {}
58196. file is already closed?
58197. line
58198. no argument?
58199. skip spaces
58200. ignore closed and incompletely open files
58201. numeral is hexadecimal
58202. is there error information?
58203. ** Read a sequence of (hex)digits
58204. prepare buffer to read whole block
58205. is this a valid number?
58206. Check whether 'mode' matches '[rwa]%^+?[L_MODEEXT]*'
58207. accepted extensions to 'mode' in 'fopen'
58208. create metatable for file handles
58209. ok
58210. get decimal point from locale
58211. always success
58212. ** function to close regular files
58213. file being read
58214. ** Read a number: first reads a valid prefix of a numeral into a buffer. ** Then it calls 'lua_stringtonumber' to check whether the format is ** correct and to convert it to a Lua number
58215. Windows (but not DDK) and Visual C++ 2005 or higher
58216. 2nd result is error message
58217. want a newline and have one?
58218. put file at index 1
58219. count initial '0' as a valid digit
58220. close buffer
58221. add file methods to new metatable
58222. ** {===== ** READ ** ======**
58223. add file to registry
58224. close it
58225. no memory errors can happen inside the lock
58226. invalidate result
58227. close it after iteration
58228. "result" to be removed
58229. keep file opened
58230. fail
58231. exponent digits
58232. exponent mark?
58233. file handle already on stack top
58234. skip optional '*' (for compatibility)

58235. open a new file
58236. exponent signal
58237. new module
58238. line with end-of-line
58239. number of chars actually read
58240. read fails
58241. first result is nil: EOF or error
58242. auxiliary structure used by 'read_number'
58243. check that it's a valid file handle
58244. integral part
58245. return them
58246. error
58247. optional signal
58248. push nil instead
58249. not a file
58250. should return at least a nil
58251. repeat until end of line
58252. do not close it after iteration
58253. }
58254. to return 1 result
58255. always accept a dot
58256. return current value
58257. unread look-ahead char
58258. no arguments?
58259. decimal point?
58260. ** Calls the 'close' function from a file handle. The 'volatile' avoids ** a bug in some versions of the Clang compiler (e.g., clang 3.0 for ** 32 bits).
58261. generator created file?
58262. push file at the stack top (to be returned)
58263. put it at index 1
58264. return ok if read something (either a newline or something else)
58265. buffer overflow?
58266. remove last result
58267. get default input
58268. ** Accept current char if it is in 'set' (of size 2)
58269. no file name?
58270. preallocate buffer
58271. read at least one value?
58272. remove read digits from buffer
58273. LUA_NUMBER
58274. short literal strings
58275. ** increment line number and skips newline sequence (any of ** \n, \r, \n\r, or \r\n)
58276. boolean value does not need GC barrier; table has no metatable, so it does not need to invalidate cache
58277. identifier or reserved word?
58278. '...'
58279. create env name
58280. exponent part?
58281. remove saved chars from buffer
58282. 'skip_sep' may dirty the buffer
58283. else short comment
58284. skip '\n' or '\r'
58285. fixed format (symbols and reserved words)?
58286. initialize buffer
58287. string starts with a newline?
58288. ** Check whether current char is in set 'set' (with two chars) and ** saves it
58289. '['=...' missing second bracket
58290. read up to 3 digits
58291. ** creates a new string and anchors it in scanner's table so that ** it will not be collected until the end of the compilation ** (by that time it should be anchored somewhere)
58292. temporarily anchor it in stack
58293. skip it
58294. use this one
58295. not in use yet?
58296. final character to be saved
58297. and discharge it
58298. read next token
58299. spaces
58300. long comment?
58301. reserved words are never collected
58302. string already present
58303. optional exponent sign
58304. skip 2nd '['
58305. long string or simply '['
58306. skip the 'z'
58307. remove '\\'
58308. ** skip a sequence '[=*[[' or ']=*]]; if sequence is well formed, return ** its number of '='s; otherwise, return a negative number (-1 iff there ** are no '='s after initial bracket)
58309. skip '\n\r' or '\r\n'
58310. ** \$Id: llex.c,v 2.96 2016/05/02 14:02:12 roberto Exp \$ ** Lexical Analyzer ** See Copyright Notice in lua.h
58311. escape sequences
58312. skip until end of line (or end of file)
58313. **comment:** to avoid warnings
 label: code-design
58314. names, strings, and numerals
58315. go through
58316. skip 2nd ']'
58317. result accumulator
58318. ORDER RESERVED
58319. skip ')'

58320. chars to be removed: '\', 'u', '{', and first digit
58321. reserved word
58322. get env name
58323. hexadecimal?
58324. initial line (for error message)
58325. **this function is quite liberal in what it accepts, as 'luaO_str2num' ** will reject ill-formed numerals.
58326. no look-ahead token
58327. will raise an error next loop
58328. previous call may dirty the buff.
58329. avoid wasting space
58330. digital escape 'ddd'
58331. skip long comment
58332. add current to buffer for error message
58333. error
58334. create new string
58335. reserved word?
58336. skip delimiter
58337. line breaks
58338. '..'
58339. re-use value previously stored
58340. entry for 'str'
58341. remove string from stack
58342. '-' or '--' (comment)
58343. ',', '..', '...', or number
58344. t[string] = true
58345. else is a comment
58346. must have at least one digit
58347. add 'buff' to string
58348. zap following span of spaces
58349. single-byte symbols?
58350. single-char tokens (+ - / ...)
58351. never collect this name
58352. is there a look-ahead token?
58353. keep delimiter (for error messages)
58354. keep '\\' for error messages
58355. skip 'u'
58356. ===== ** LEXICAL ANALYZER ** =====
58357. format error?
58358. ** \$Id: lex.h,v 1.79 2016/05/02 14:02:12 roberto Exp \$ ** Lexical Analyzer ** See Copyright Notice in lua.h
58359. environment variable name
58360. current character (charint)
58361. terminal symbols denoted by reserved words
58362. input stream
58363. line of last token 'consumed'
58364. semantics information
58365. dynamic structures used by the parser
58366. input line counter
58367. state of the lexer plus state of the parser when shared by all functions
58368. look ahead token
58369. number of reserved words
58370. current source name
58371. current token
58372. **comment:** to avoid collection/reuse strings
 label: code-design
58373. * WARNING: if you change the order of this enumeration, * grep "ORDER RESERVED"
58374. other terminal symbols
58375. buffer for tokens
58376. current function (parser)
58377. maximum value for size_t
58378. maximum size visible for Lua (must be representable in a lua_Integer)
58379. type to ensure maximum alignment
58380. **comment:** ** macro to control inclusion of some hard tests on stack reallocation
 label: test
58381. ** 'lu_mem' and 'l_mem' are unsigned/signed integers big enough to count ** the total memory used by Lua (in bytes). Usually, 'size_t' and ** 'ptrdiff_t' should work, but we use 'long' for 16-bit machines.
58382. ** The luai_num* macros define the primitive operations over numbers.
58383. the others are quite standard operations
58384. maximum value of an int
58385. ** non-return type
58386. ** assertion for checking API calls
58387. }{
58388. realloc stack keeping its size
58389. ** Size of cache for strings in the API. 'N' is the number of ** sets (better be a prime) and "M" is the size of each set (M == 1 ** makes a direct cache.)
58390. }
58391. { external definitions?
58392. ** conversion of pointer to unsigned integer: ** this is for hashing only; there is no problem if the integer ** cannot hold the whole pointer value
58393. **comment:** macro to avoid warnings about unused variables
 label: code-design
58394. ** Maximum length for short strings, that is, strings that are ** internalized. (Cannot be smaller than reserved words or tags for ** metamethods, as these strings must be internalized; ** #("function") = 8, #("__newindex") = 10.)
58395. ** Initial size for the string table (must be power of 2). ** The Lua core alone registers ~50 strings (reserved words + ** metaevent keys + a few others). Libraries would typically add ** a few dozens more.
58396. ** maximum depth for nested C calls and syntactical nested non-terminals ** in a program. (Value must fit in an unsigned short int.)
58397. ** type for virtual-machine instructions; ** must be an unsigned with (at least) 4 bytes (see details in lopcodes.h)
58398. types of 'usual argument conversions' for lua_Number and lua_Integer
58399. cast a signed lua_Integer to lua_Unsigned
58400. minimum size for string buffer

58401. ** modulo: defined as 'a - floor(a/b)*b'; this definition gives NaN when ** 'b' is huge, but the result should be 'a'. 'fmod' gives the result of ** 'a - trunc(a/b)*b', and therefore must be corrected when 'trunc(a/b) ** ~= floor(a/b)'. That happens when the division has a non-integer ** negative result, which is equivalent to the test below.

58402. ** these macros allow user-specific actions on threads when you defined ** LUAI_EXTRASPACE and need to do something extra when a thread is ** created/deleted/resumed/yielded.

58403. ** cast a lua_Unsigned to a signed lua_Integer; this cast is ** not strict ISO C, but two-complement architectures should ** work fine.

58404. internal assertions for in-house debugging

58405. 16-bit ints

58406. to avoid problems with conditions too long

58407. ** macros that are executed whenever program enters the Lua core ** ('lua_lock') and leaves the core ('lua_unlock')

58408. ** \$Id: llimits.h,v 1.141 2015/11/19 19:16:22 roberto Exp \$ ** Limits, basic types, and some other 'installation-dependent' definitions ** See Copyright Notice in lua.h

58409. chars used as small naturals (so that 'char' is reserved for characters)

58410. float division

58411. floor division (defined as 'floor(a/b)')

58412. exponentiation

58413. ** macro executed during Lua functions at points where the ** function can yield.

58414. **comment:** type casts (a macro highlights casts in the code)
label: code-design

58415. index of current maximum value

58416. special cases: -1 or 0

58417. no arguments

58418. lower and upper limits

58419. check number of arguments

58420. }=====

58421. number is its own integer part

58422. }

58423. fractional part (test needed for inf/-inf)

58424. integer is its own ceil

58425. Number between 0 and 1

58426. ** {===== ** Deprecated functions (for compatibility only) ** =====

58427. no fractional part

58428. ** Open math library

58429. ** This function uses 'double' (instead of 'lua_Number') to ensure that ** all bits from 'l_rand' can be represented, and that 'RANDMAX + 1.0' ** will keep full precision (ensuring that 'r' is always less than 1.0.)

58430. only upper limit

58431. number of arguments

58432. placeholders

58433. {

58434. does 'd' fit in an integer?

58435. value is not convertible to integer

58436. random integer in the interval [low, up]

58437. result is float

58438. index of current minimum value

58439. integer part (rounds toward zero)

58440. ** \$Id: lmathlib.c,v 1.117 2015/10/02 15:39:23 roberto Exp \$ ** Standard mathematical library ** See Copyright Notice in lua.h

58441. result is integer

58442. (2^31 - 1), following POSIX

58443. ** next function does not use 'modf', avoiding problems with 'double*' ** (which is not compatible with 'float*') when lua_Number is not ** 'double'.

58444. integer is its own floor

58445. discard first value to avoid undesirable correlations

58446. avoid overflow with 0x80000... / -1

58447. ** About the realloc function: ** void * frealloc (void *ud, void *ptr, size_t osize, size_t nsize); ** ('osize' is the old size, 'nsize' is the new size) *** *** * frealloc(ud, NULL, x, s) creates a new block of size 's' (no ** matter 'x'). *** *** * frealloc(ud, p, x, 0) frees the block 'p' ** (in this specific case, frealloc must return NULL); ** particularly, frealloc(ud, NULL, 0, 0) does nothing ** (which is equivalent to free(NULL) in ISO C) *** *** frealloc returns NULL if it cannot create or reallocate the area ** (any reallocation to an equal or smaller size cannot fail!)

58448. cannot fail when shrinking a block

58449. cannot double it?

58450. cannot grow even a little?

58451. update only when everything else is OK

58452. minimum size

58453. ** generic allocation routine.

58454. force a GC whenever possible

58455. ** \$Id: lmemp.c,v 1.91 2015/03/06 19:45:54 roberto Exp \$ ** Interface to Memory Manager ** See Copyright Notice in lua.h

58456. still have at least one free place

58457. try to free some memory...

58458. try again

58459. is state fully built?

58460. not to be called directly

58461. ** \$Id: lmemp.h,v 1.43 2014/12/19 17:26:14 roberto Exp \$ ** Interface to Memory Manager ** See Copyright Notice in lua.h

58462. **comment:** ** Arrays of chars do not need any test
label: test

58463. **comment:** ** This macro reallocs a vector 'b' from 'on' to 'n' elements, where ** each element has size 'e'. In case of arithmetic overflow of the ** product 'n'*'e', it raises an error (calling 'luAM_toobig'). Because ** 'e' is always constant, it avoids the runtime division MAX_SIZE(e). *** *** (The macro is somewhat complex to avoid warnings: The 'sizeof' ** comparison avoids a runtime comparison when overflow cannot occur. ** The compiler should be able to optimize the real test by itself, but ** when it does it, it may give a warning about "comparison is always ** false due to limited range of data type"; the +1 tricks the compiler, ** avoiding this warning but also this optimization.)
label: code-design

58464. no errors?

58465. ** Look for a C function named 'sym' in a dynamically loaded library ** 'path'. ** First, check whether the library is already loaded; if not, try ** to load it. ** Then, if 'sym' is '*', return true (as library has been loaded). ** Otherwise, look for symbol 'sym' in the library and push a ** C function with that symbol. ** Return 0 and 'true' or a function in the stack; in case of ** errors, return an error code and an error message in the stack.

58466. set CLIBS table in registry

58467. remove file name

58468. pop CLIBS table

58469. ** unload library 'lib'

58470. concatenate error message

58471. _LOADED[name] = true
58472. root not found
58473. unable to load library
58474. **comment:** concatenate error msg. entry
 label: code-design
58475. return nil + error message
58476. CLIBS[#CLIBS + 1] = plib
58477. create new metatable
58478. not found
58479. ** {===== ** This is an implementation of loadlib based on the dlfcn interface. ** The dlfcn interface is available in Linux, SunOS, Solaris, IRIX, FreeBSD, ** NetBSD, AIX 4.2, HPUX 11, and probably most other Unix flavors, at least ** as an emulation layer on top of native functions. **
=====

58480. run loader to load module
58481. non-empty separator?
58482. _LOADED[name]
58483. set field 'cpath'
58484. to build error message
58485. name is 1st argument (before search data)
58486. package is already loaded
58487. CLIBS[path] = plib
58488. **comment:** remove extra return
 label: code-design
58489. ** LUA_PATH_SEP is the character that separates templates in a path. ** LUA_PATH_MARK is the string that marks the substitution points in a ** template. ** LUA_EXEC_DIR in a Windows path is replaced by the executable's ** directory. ** LUA_IGMARK is a mark to ignore all before it when building the ** luaopen_ function name.
58490. put it in field 'searchers'
58491. unable to find function
58492. {
58493. use default
58494. iterate over available searchers to find a loader
58495. push 'package.searchers' to index 3 in the stack
58496. open failed
58497. copy new environment table to top
58498. **comment:** ** Macro to convert pointer-to-void* to pointer-to-function. This cast ** is undefined according to ISO C, but POSIX assumes that it works. ** (The '__extension__' in gnu compilers is only to avoid warnings.)
 label: code-design
58499. set finalizer for CLIBS table
58500. try alternative name
58501. }=====

58502. remove 'getfield' result
58503. module loader found
58504. remove function
58505. module set no value?
58506. **comment:** not used
 label: code-design
58507. does file exist and is readable?
58508. did it find a loader?
58509. fill it with predefined searchers
58510. put it in field 'loaders'
58511. no errors
58512. ** return registry.LUA_NOENV as a boolean
58513. look for last dot in module name
58514. pop CLIBS table and 'plib'
58515. module not found in this path
58516. non-nil return?
58517. {}
58518. searcher returned error message?
58519. return nil, error message, and where
58520. check whether table already has a _NAME field
58521. else must load package
58522. try to open file
58523. error codes for 'lookforfunc'
58524. return the loaded function
58525. set 'package' as upvalue for next lib
58526. ** system-dependent functions
58527. open lib into global table
58528. ** {===== ** Fallback for other systems **
=====

58529. return that file name
58530. _LOADED table will be at index 2
58531. create metatable for CLIBS
58532. mt._index = _G
58533. module._M = module
58534. no more searchers?
58535. auxiliary mark (for internal use)
58536. skip separators
58537. template
58538. is it there?
58539. return 'true'
58540. _LOADED[name] = returned value
58541. pop handle
58542. get calling function
58543. pop global table
58544. find next separator
58545. create error message
58546. module
58547. check loaded C libraries

58548. ** return registry.CLIBS[path]
58549. else create new function
58550. module loaded successfully?
58551. set field 'path'
58552. return open function and file name
58553. no; initialize it
58554. set field 'preload'
58555. for each handle, in reverse order
58556. open function not found
58557. ** optional flags for LoadLibraryEx
58558. real error
58559. no more templates
58560. placeholders
58561. remove nil
58562. ** {===== ** 'require' function ======**
58563. **comment:** will be 2nd argument to module
label: code-design
58564. ** registry.CLIBS[path] = plib -- for queries ** registry.CLIBS[#CLIBS + 1] = plib -- also keep a list of all libraries
58565. remove path template
58566. no environment variable?
58567. error message is on top of the stack
58568. must load library?
58569. pass name as argument to module loader
58570. error; error message is on stack top
58571. loading only library (no function)?
58572. use true as result
58573. store config information
58574. call it
58575. separator for open functions in C libraries
58576. ** Try to find a function named 'sym' in library 'lib'. ** Returns the function; in case of error, returns NULL plus an ** error string in the stack.
58577. **comment:** not used: symbols are 'global' by default
label: code-design
58578. ** LUA_CSUBSEP is the character that replaces dots in submodule names ** when searching for a C loader. ** LUA_LSUBSEP is the character that replaces dots in submodule names ** when searching for a Lua loader.
58579. get option (a function)
58580. }
58581. remove original string
58582. get handle CLIBS[n]
58583. avoid 'calling' extra info.
58584. table is an initialized module
58585. remove value
58586. replace it by 'dirsep'
58587. ** unique key for table in the registry that keeps handles ** for all loaded C libraries
58588. ** __gc tag method for CLIBS table: calls 'sys_unloadlib' for all lib ** handles in list CLIBS
58589. ** LUA_PATH_VAR and LUA_CPATH_VAR are the names of the environment ** variables that Lua check to set its paths.
58590. global symbols if 'sym'==*'
58591. else go ahead and try old-style name
58592. prefix for open functions in C libraries
58593. is root
58594. make a copy of 'searchers' table
58595. extra copy to be returned
58596. set _PACKAGE as package name (full module name minus last part)
58597. create 'package' table
58598. ** create table CLIBS to keep track of loaded C libraries, ** setting a finalizer to close all libraries when closing state.
58599. ** load C library in file 'path'. If 'seeglb', load with all names in ** the library global. ** Returns the library; in case of error, returns NULL plus an ** error string in the stack.
58600. ** changes the environment variable of calling function
58601. create CLIBS table
58602. not found?
58603. ** {===== ** This is an implementation of loadlib for Windows using native functions. ** ======**
58604. plib = CLIBS[path]
58605. ** Try to find a load function for module 'modname' at file 'filename'. ** First, change '!' to '_' in 'modname'; then, if 'modname' has ** the form X-Y (that is, it has an "ignore mark"), build a function ** name "luaopen_X" and look for it. (For compatibility, if that ** fails, it also tries "luaopen_Y".) If there is no ignore mark, ** look for a function named "luaopen_modname".
58606. return 'package' table
58607. get/create module table
58608. ** {===== ** 'module' function ======**
58609. create 'searchers' table
58610. last parameter
58611. remove both returns
58612. replace ";" by ";AUXMARK;" and then AUXMARK by default path
58613. set field 'loaded'
58614. set 'package' as upvalue for all searchers
58615. ** \$Id: loadlib.c,v 1.127 2015/11/23 11:30:45 roberto Exp \$ ** Dynamic library loader for Lua ** See Copyright Notice in lua.h ** ** This module contains an implementation of loadlib for Unix systems ** that have dlfcn, an implementation for Windows, and a stub for other ** systems.
58616. convert the top element to a string
58617. correct decimal point
58618. maximum length of a numeral
58619. true after seen a dot
58620. invalid format
58621. skip '0x'
58622. 1 if number is negative
58623. now there is one less bit available in first byte
58624. can read it without overflow?
58625. read exponent

58626. string; format as [string "source"]
58627. check '0x'
58628. add first byte
58629. exponent part?
58630. zero-terminated string
58631. 'literal' source
58632. skip trailing spaces
58633. a pointer
58634. try to convert
58635. add prefix
58636. skip '0x' and read numeral
58637. result (accumulator)
58638. skip 'p'
58639. should not fail when folding (compile time)
58640. }=====

58641. coarse steps
58642. an 'int' as a UTF-8 sequence
58643. keep it
58644. decimal
58645. $x = \text{ceil}(x / 16)$
58646. no digits?
58647. remove added bits
58648. try again
58649. number of chars of a literal string without the ending \0
58650. looks like an int?
58651. nothing recognized?
58652. signal
58653. an 'int'
58654. maximum length of the conversion of a number to a string
58655. could not perform raw operation; try metamethod
58656. ** convert an hexadecimal numeric string to a number, following ** C99 specification for 'strtod'
58657. add continuation bytes
58658. ** this function handles only "%d", "%c", "%f", "%p", and "%s" conventional formats, plus Lua-specific "%I" and "%U"
58659. number of significant digits
58660. conversion failed
58661. invalid format (no '0x')
58662. make relative to 's'
58663. exponent value
58664. reject 'inf' and 'nan'
58665. nothing is valid yet
58666. fine steps
58667. $\log_2[i] = \text{ceil}(\log_2(i - 1))$
58668. $x = \text{ceil}(x / 2)$
58669. too many digits; ignore, but still count for exponent
58670. file name
58671. find first new line (if any)
58672. still needs continuation byte?
58673. neither a dot nor a digit
58674. small one-line source?
58675. something wrong in the numeral
58676. do not accept it (as integer)
58677. hex?
58678. other operations
58679. valid up to here
58680. need continuation bytes
58681. ** converts an integer to a "floating point byte", represented as ** (eeeeexxx), where the real value is $(1xxx) * 2^{(eeeeee - 1)}$ if ** eeeee != 0 and (xxx) otherwise.
58682. number of bytes put in buffer (backwards)
58683. converts back
58684. add '...' before rest of name
58685. check signal
58686. decimal digit? correct exponent
58687. else try as a float
58688. success; return string size
58689. an 'int' as a character
58690. exponent signal
58691. second dot? stop loop
58692. each digit multiplies/divides value by 2^4
58693. ** Convert a number object to a string
58694. adds '.' to result
58695. stop at first newline
58696. operate only on floats
58697. overflow?
58698. ** {===== ** Lua's implementation for 'lua_str2number' **
=====

58699. a 'lua_Number'
58700. ** \$Id: lobject.c,v 2.111 2016/05/20 14:07:48 roberto Exp \$ ** Some generic functions over Lua objects ** See Copyright Notice in lua.h
58701. maximum number of significant digits to read (to avoid overflows even with single floats)
58702. should be enough space for a '%p'
58703. small enough?
58704. go to the end
58705. ascii?
58706. operate only on integers
58707. skip initial spaces
58708. number of non-significant digits
58709. truncate it
58710. non-significant digit (zero)?
58711. failed? may be a different locale
58712. exponent correction

58713. OK if no trailing characters
58714. save space for prefix+suffix+\0'
58715. exponent
58716. copy string to buffer
58717. invalid; must have at least one digit
58718. try as an integer
58719. maximum that fits in first byte
58720. string too long or no dot; fail
58721. **comment:** non-printable character; print its code
label: code-design
58722. ** Convert string 's' to a Lua number (put in 'result'). Return NULL ** on fail or the address of the ending '\0' on success. ** 'pmode' points to (and 'mode' contains) special things in the string: ** - 'x/X' means an hexadecimal numeral ** - 'n/N' means 'inf' or 'nan' (which should be rejected) ** - '.' just optimizes the search for the common case (nothing special) ** This function accepts both the current locale or a dot as the radix ** mark. If the conversion fails, it may mean number has a dot but ** locale accepts something else. In that case, the code copies 's' ** to a buffer (because 's' is read-only), changes the dot to the ** current locale radix mark, and tries to convert again.
58723. ** Computes ceil(log2(x))
58724. a 'lua_Integer'
58725. ** 'module' operation for hashing (size is always a power of 2)
58726. collectable objects
58727. Variant tags for numbers
58728. Macros to access values
58729. ** Header for string value; string bytes follow the end of this structure ** (aligned according to 'UTString'; see next).
58730. user value's tag
58731. constants used by the function
58732. to new object
58733. long strings
58734. reserved words for short strings; "has hash" for longs
58735. 2: declared vararg; 1: uses vararg
58736. whether it is in stack (register)
58737. Macros for internal tests
58738. index of upvalue (in stack or in outer function's list)
58739. linked list for hash table
58740. ** Tables
58741. integer numbers
58742. get string length from 'TString *s'
58743. Variant tags for strings
58744. macro defining a nil value
58745. ** Extra tags for non-values
58746. map from opcodes to source lines (debug information)
58747. length for short strings
58748. light userdata
58749. debug information
58750. ** Closures
58751. mark a tag as collectable
58752. ensures maximum alignment for strings
58753. copy a value into a key without messing up field 'next'
58754. used for debug information
58755. ** \$Id: lobject.h,v 2.116 2015/11/03 18:33:10 roberto Exp \$ ** Type definitions for Lua objects ** See Copyright Notice in lua.h
58756. C closure
58757. functions defined inside the function
58758. Lua closure
58759. size of 'upvalues'
58760. light C function
58761. size of 'p'
58762. ** different types of assignments, according to destination
58763. array part
58764. get string length from ' TValue *o'
58765. ** Tagged Values. This is the basic representation of values in Lua, ** an actual value plus a tag with its type.
58766. to stack (not from same stack)
58767. ** (address of) a fixed nil value
58768. ** Description of an upvalue for function prototypes
58769. type tag of a TValue with no variants (bits 0-3)
58770. Macros to test type
58771. light C functions
58772. ** Lua Upvalues
58773. for chaining (offset for next node)
58774. **comment:** first point where variable is dead
label: code-design
58775. ** tags for Tagged Values have the following use of bits: ** bits 0-3: actual tag (a LUA_T* value) ** bits 4-5: variant bits ** bit 6: whether value is collectable
58776. a dead value may get the 'gc' field, but cannot access its contents
58777. ensures maximum alignment for 'local' udata
58778. ** LUA_TFUNCTION variants: ** 0 - Lua function ** 1 - light C function ** 2 - regular C function (closure)
58779. from table to same table
58780. ** Common type for all collectable objects
58781. user value
58782. size of 'k'
58783. ** Get the address of memory block inside 'Udata'. ** (Access to 'ttuv_' ensures that value is really a 'Udata').
58784. short strings
58785. ** Description of a local variable for function prototypes ** (used for debug information)
58786. ** Ensures that address after this type is always fully aligned.
58787. first point where variable is active
58788. any free position is before this position
58789. size of buffer for 'luaO_utf8esc' function
58790. length for long strings
58791. tag with no variants (bits 0-3)
58792. upvalue name (for debug information)
58793. Variant tags for functions

58794. ** {===== types and prototypes ======**
=====

58795. size of 'array' array
58796. ** Get the actual string (array of bytes) from a 'TString'. ** (Access to 'extra' ensures that value is really a 'TString'.)
58797. type tag of a TValue (bits 0-3 for tags + variant bits 4-5)
58798. function prototypes
58799. Macros to set values
58800. ** Common type has only the common header
58801. booleans
58802. upvalue information
58803. float numbers
58804. removed keys in tables
58805. ** number of all possible tags (including LUA_TNONE but excluding DEADKEY)
58806. last-created closure with this prototype
58807. Bit mark for collectable types
58808. opcodes
58809. index to stack elements
58810. number of fixed parameters
58811. get the actual string (array of bytes) from a Lua value
58812. raw type tag of a TValue
58813. ** Header for userdata; memory area follows the end of this structure ** (aligned according to 'UUdata'; see next).
58814. from stack to (same) stack
58815. to table (define it as an expression to be used in macros)
58816. 1<<p means tagmethod(p) is not present
58817. number of bytes
58818. number of registers needed by this function
58819. list of upvalues
58820. ** Common Header for all collectable objects (in macro form, to be ** included in other objects)
58821. information about local variables (debug information)
58822. ** Union of all Lua values
58823. log2 of size of 'node' array
58824. ** Function Prototypes
58825. OP_LOADNIL
58826. OP_UNM
58827. OP_MOVE
58828. OP_IDIV
58829. OP_SELF
58830. OP_VARARG
58831. OP_RETURN
58832. OP_JMP
58833. OP_FORPREP
58834. OP_CLOSURE
58835. OP_LOADKX
58836. OP_DIV
58837. OP_NOT
58838. OP_LE
58839. OP_TFORCALL
58840. OP_LEN
58841. OP_EQ
58842. OP_SETUPVAL
58843. OP_POW
58844. OP_BNOT
58845. OP_BOR
58846. OP_LT
58847. OP_SETLIST
58848. OP_TFORLOOP
58849. OP_SHR
58850. ORDER OP
58851. T A B C mode opcode
58852. OP_EXTRAARG
58853. OP_CONCAT
58854. ** \$Id: lopcodes.c,v 1.55 2015/01/05 13:48:33 roberto Exp \$ ** Opcodes for Lua virtual machine ** See Copyright Notice in lua.h
58855. OP_SETTABLE
58856. OP_MUL
58857. OP_LOADK
58858. OP_SHL
58859. OP_BXOR
58860. OP_GETTABLE
58861. OP_LOADBOOL
58862. OP_GETTABUP
58863. OP_FORLOOP
58864. OP_BAND
58865. OP_MOD
58866. OP_NEWTABLE
58867. OP_SETTABUP
58868. OP_TEST
58869. OP_TAILCALL
58870. OP_ADD
58871. OP_CALL
58872. OP_SUB
58873. OP_GETUPVAL
58874. OP_TESTSET
58875. A B R(A), R(A+1), ..., R(A+B) := nil
58876. **comment:** argument is not used
 label: code-design
58877. ===== Notes: (*) In OP_CALL, if (B == 0) then B = top. If (C == 0), then 'top' is set to last_result+1, so next open instruction (OP_CALL, OP_RETURN, OP_SETLIST) may use 'top'. (*) In OP_VARARG, if (B == 0) then use actual number of varargs and set top (like in OP_CALL with C == 0). (*) In OP_RETURN, if (B == 0) then return up to 'top'. (*) In OP_SETLIST, if (B == 0)

then B = 'top'; if (C == 0) then next 'instruction' is EXTRAARG(real C). (*) In OP_LOADKX, the next 'instruction' is always EXTRAARG. (*) For comparisons, A specifies what condition the test should accept (true or false). (*) All 'skips' (pc++) assume that next instruction is a jump.

=====

58878. ** invalid register that fits in 8 bits
58879. number of list items to accumulate before a SETLIST instruction
58880. A B C R(A) := R(B)[RK(C)]
58881. A B C if ((RK(B) == RK(C)) ~=~ A) then pc++
58882. A C if not (R(A) <=> C) then pc++
58883. A B C if (R(B) <=> C) then R(A) := R(B) else pc++
58884. test whether value is a constant
58885. creates a mask with 'n' 1 bits at position 'p'
58886. A B C if ((RK(B) < RK(C)) ~=~ A) then pc++
58887. ** R(x) - register ** Kst(x) - constant (in constant table) ** RK(x) == if ISK(x) then Kst(INDEXK(x)) else R(x)
58888. A B C R(A+1) := R(B); R(A) := R(B)[RK(C)]
58889. A B C return R(A)(R(A+1), ... ,R(A+B-1))
58890. this bit 1 means constant (0 means register)
58891. ===== We assume that instructions are unsigned numbers. All instructions have an opcode in the first 6 bits. Instructions can have the following fields: 'A' : 8 bits 'B' : 9 bits 'C' : 9 bits 'Ax' : 26 bits ('A', 'B', and 'C' together) 'Bx' : 18 bits ('B' and 'C' together) 'sBx' : signed Bx A signed argument is represented in excess K; that is, the number value is the unsigned value minus K. K is exactly the maximum value for that argument (so that -max is represented by 0, and +max is represented by 2*max), which is half the maximum for the corresponding unsigned argument. =====
58892. A Bx R(A) := Kst(Bx)
58893. ** Macros to operate RK indices
58894. A B C R(A) := RK(B) << RK(C)
58895. basic instruction format
58896. A B C R(A) := UpValue[B][RK(C)]
58897. A C R(A+3), ... ,R(A+2+C) := R(A)(R(A+1), R(A+2));
58898. A B C R(A) := RK(B) ^ RK(C)
58899. A B R(A) := ~R(B)
58900. ** the following macros help to manipulate instructions
58901. A B C R(A)[RK(B)] := RK(C)
58902. A sBx if R(A+1) ~=~ nil then { R(A)=R(A+1); pc += sBx }
58903. A B C R(A) := RK(B) - RK(C)
58904. A sBx R(A)+=R(A+2); if R(A) <=? R(A+1) then { pc+=sBx; R(A+3)=R(A) }
58905. A B C R(A)[(C-1)*FPF+i] := R(A+i), 1 <= i <= B
58906. A B C R(A) := {} (size = B,C)
58907. A B C if ((RK(B) <= RK(C)) ~=~ A) then pc++
58908. A B UpValue[B] := R(A)
58909. A sBx R(A)=R(A+2); pc+=sBx
58910. ----- name args description -----
58911. A B C R(A) := RK(B) & RK(C)
58912. A B C R(A) := RK(B) / RK(C)
58913. ** size and position of opcode arguments.
58914. A B C R(A) := RK(B) | RK(C)
58915. A B C R(A) := RK(B) * RK(C)
58916. A B R(A) := not R(B)
58917. ** limits for opcode arguments. ** we use (signed) int to manipulate most arguments, ** so they must fit in LUA_BITSINT-1 bits (-1 for sign)
58918. argument is used
58919. A B R(A) := length of R(B)
58920. A B R(A) := R(B)
58921. A B C R(A) := RK(B) ~ RK(C)
58922. opcode names
58923. ** grep "ORDER OP" if you change these enums
58924. argument is a register or a jump offset
58925. A B C R(A) := (Bool)B; if (C) pc++
58926. gets the index of the constant
58927. creates a mask with 'n' 0 bits at position 'p'
58928. A B C R(A) := RK(B) // RK(C)
58929. comment: Ax extra (larger) argument for previous opcode
label: code-design
58930. argument is a constant or register/constant
58931. A B return R(A), ... ,R(A+B-2) (see note)
58932. A B C R(A) := RK(B) % RK(C)
58933. A B R(A) := -R(B)
58934. A Bx R(A) := closure(KPROTO[Bx])
58935. A B R(A) := UpValue[B]
58936. A B C R(A) := R(B)... ...R(C)
58937. A R(A) := Kst(extra arg)
58938. A B C UpValue[A][RK(B)] := RK(C)
58939. A B R(A), R(A+1), ... ,R(A+B-2) = vararg
58940. ** \$Id: lpcodes.h,v 1.148 2014/10/25 11:50:46 roberto Exp \$ ** Opcodes for Lua virtual machine ** See Copyright Notice in lua.h
58941. A B C R(A) := RK(B) >> RK(C)
58942. A B C R(A), ... ,R(A+C-2) := R(A)(R(A+1), ... ,R(A+B-1))
58943. code a constant index as a RK value
58944. ** masks for instruction properties. The format is: ** bits 0-1: op mode ** bits 2-3: C arg mode ** bits 4-5: B arg mode ** bit 6: instruction set register A ** bit 7: operator is a test (next instruction must be a jump)
58945. A sBx pc+=sBx; if (A) close all upvalues >= R(A - 1)
58946. 'sBx' is signed
58947. A B C R(A) := RK(B) + RK(C)
58948. options for Windows
58949. copy valid option to buffer
58950. ** ===== ** Configuration for time-related stuff **
=====

58951. next length
58952. options for ANSI C 89
58953. options for ISO C 99 and POSIX
58954. C99 specification
58955. copy specifier to 'cc'

58956. skip '!'
58957. {}
58958. invalid date?
58959. ======
58960. next block?
58961. }
58962. update fields with normalized values
58963. UTC?
58964. not a conversion specifier?
58965. return next item
58966. absent field; no default?
58967. ** { ====== ** Configuration for 'tmpnam': ** By default, Lua uses tmpnam
except when POSIX is available, where ** it uses mkstemp. ** ======
58968. field is not an integer?
58969. ** Set all fields from structure 'tm' in the table on top of the stack
58970. some other value?
58971. ** By default, Lua uses gmtime/localtime, except when POSIX is available, ** where it uses gmtime_r/localtime_r
58972. ** type to represent time_t in Lua
58973. match?
58974. get field and its type
58975. ** { ====== ** List of valid conversion specifiers for the 'strftime' function;
** options are grouped by length; group of length 2 start with '||'. ** ======
58976. if NULL push nil
58977. {
58978. get current time
58979. maximum value for date fields (to avoid arithmetic overflows with 'int')
58980. buffer for individual conversion specifiers
58981. called without args?
58982. maximum size for an individual 'strftime' item
58983. ISO C definitions
58984. does not set field
58985. **comment:** to avoid warnings
label: code-design
58986. ======
58987. ** \$Id: loslib.c,v 1.64 2016/04/18 13:06:55 roberto Exp \$ ** Standard Operating System library ** See Copyright Notice in lua.h
58988. ** { ====== ** Time/Date operations ** { year=%Y, month=%m, day=%d, hour=%H,
min=%M, sec=%S, ** wday=%w+1, yday=%j, isdst=? } ** ======
58989. 9 = number of fields
58990. undefined?
58991. **comment:** 'if' to avoid warnings for unreachable 'return'
label: code-design
58992. make sure table is at the top
58993. true if there is a shell
58994. 'else' part
58995. fix it at the last register
58996. new or old upvalue
58997. not found as local at current level; try upvalues
58998. anchor it (to avoid being collected)
58999. index is the local being assigned? (index cannot be upvalue)
59000. end of scope for declared variables
59001. ** check whether current token is in the follow set of a block. ** 'until' closes syntactical blocks, but do not close scope, ** so it is handled in separate.
59002. function created in next register
59003. '/ '//
59004. read first token
59005. must skip over 'then' part if condition is false
59006. '+' '-'
59007. global name?
59008. assume that locals are already out of scope
59009. create and...
59010. not found
59011. remove "near <token>" from final message
59012. param -> '...'
59013. ** check whether, in an assignment to an upvalue/local variable, the ** upvalue/local variable is begin used in a previous assignment to a ** table. If so, save
original upvalue/local value in a safe place and ** use this safe copy in the previous assignment.
59014. must enter block before 'goto'
59015. env[varname]
59016. stat -> forstat
59017. ** compiles the main function, which is a regular vararg function with an ** upvalue named LUA_ENV
59018. no value (yet)
59019. stat -> LOCAL NAME {' NAME} [= explist]
59020. skip WHILE
59021. free registers
59022. ======
59023. stat -> func | assignment
59024. no more levels?
59025. reserve register for parameters
59026. ** structure to chain all variables in the left-hand side of an ** assignment
59027. control variables
59028. left priority for each binary operator
59029. ** subexpr -> (simpleexp | unop subexpr) { binop subexpr } ** where 'binop' is any binary operator with a priority higher than 'limit'
59030. assignment -> '=' explist
59031. skip over block if condition is false
59032. skip DO
59033. local function?
59034. ~=, >, >=
59035. ...set environment upvalue
59036. skip other no-op statements
59037. forstat -> FOR (fornum | forlist) END

59038. cond -> exp
59039. create control variables
59040. chain
59041. last list item read
59042. skip double colon
59043. instruction to skip 'then' code (if condition is false)
59044. remove goto from pending list
59045. patch escape list to 'if' end
59046. only one single value?
59047. funcargs
59048. upvalues?
59049. remove scanner's table
59050. scope block
59051. generic for
59052. ** create a label named 'break' to resolve break statements
59053. simpleexp -> FLT | INT | STRING | NIL | TRUE | FALSE | ... | constructor | FUNCTION body | suffixedexp
59054. check all previous assignments
59055. correct pending gotos to current block and try to close it with visible labels
59056. 'then' part
59057. variable is local
59058. 'falses' are all equal here
59059. debug information will only see the variable after this point!
59060. stat -> ';' (empty statement)
59061. close the loop
59062. stat -> 'goto' NAME
59063. return no values
59064. includes call itself
59065. open call
59066. ** \$Id: lparser.c,v 2.153 2016/05/13 19:10:16 roberto Exp \$ ** Lua Parser ** See Copyright Notice in lua.h
59067. funcstat -> FUNCTION funcname body
59068. stat -> assignment ?
59069. numeric for?
59070. values must go to the stack
59071. funcname -> NAME {fieldsel} [':' NAME]
59072. set initial array size
59073. key is variable name
59074. tail call?
59075. may be needed for error messages
59076. last exp. provides the difference
59077. create and anchor TString
59078. check for repeated labels
59079. linked list of funcstates
59080. ** nodes for block list (list of active blocks)
59081. close it if label already defined
59082. IF cond THEN block
59083. stat -> breakstat
59084. update pending gotos to outer block
59085. block -> statlist
59086. flush
59087. call statement uses no results
59088. correct label?
59089. label not found; cannot close goto
59090. prototype of current function
59091. ** export pending gotos to outer level, to check them against ** outer labels; if the block being exited has upvalues, and ** the goto exits the scope of any variable (which can be the ** upvalue), close those variables being exited.
59092. statlist -> { stat [';'] }
59093. funcargs -> '(' [explist] ')'
59094. arg list is empty?
59095. create declared variables
59096. skip ';'
59097. skip the '['
59098. skip no-op statements
59099. stat -> retstat
59100. stat -> label
59101. stat -> DO block END
59102. create table for scanner
59103. else was LOCAL or UPVAL
59104. ** check whether new label 'lb' matches any pending gotos in current ** block; solves forward jumps
59105. inner block?
59106. optional step
59107. default is global
59108. there is no list item
59109. constructor
59110. ===== ** Rules for Constructors **
=====
59111. registers with returned values
59112. true if some variable in the block is an upvalue
59113. limit
59114. return first untreated operator
59115. skip 'for'
59116. **comment:** maximum number of local variables per function (must be smaller than 250, due to the bytecode format)
 label: code-design
59117. default step = 1
59118. close last argument
59119. fornum -> NAME = exp1,exp1[,exp1] forbody
59120. definition "happens" in the first line
59121. look up locals at current level
59122. ELSEIF cond THEN block

59123. function actually uses vararg
59124. ORDER OPR
59125. may be 'listfield' or 'recfield'
59126. stat -> RETURN [explist] [':']
59127. constructor -> '{' [field { sep field } [sep]] '} sep -> ',' | ';' | '
59128. variable (global, local, upvalue, or indexed)
59129. previous assignment will use safe copy
59130. followed by 'else'/'elseif'?
59131. check labels in current block for a match
59132. ls->t.token == '['
59133. finish scope
59134. index -> '[' expr ']'
59135. Find variable with given name 'n'. If it is an upvalue, add this upvalue into all intermediate functions.
59136. ======
59137. anchor it
59138. ... (right associative)
59139. because all strings are unified by the scanner, the parser can use pointer equality for string equality
59140. close it
59141. return all active values
59142. skip RETURN
59143. **comment:** remove extra values
 label: code-design
59144. ** generates an error for an undefined 'goto'; choose appropriate ** message when label name is a reserved word (which can only be 'break')
59145. stat -> funcstat
59146. all scopes should be correctly finished
59147. remove local labels
59148. stat -> func
59149. '^' (right associative)
59150. pending gotos in outer block?
59151. stat -> whilestat
59152. new local variable
59153. ** try to close a goto with existing labels; this solves backward jumps
59154. skip the dot or colon
59155. expression?
59156. initial value
59157. must jump over it
59158. label is last no-op statement in the block?
59159. is 'parlist' not empty?
59160. registers 0/1 are always valid
59161. closure is on the stack, too
59162. ====== ** Expression parsing **
59163. suffixedexp -> primaryexp { '.' NAME | '[' exp ']' | ':' NAME funcargs | funcargs }
59164. scope for loop and control variables
59165. label -> ':' NAME ':'
59166. try upper levels
59167. it is a global
59168. eventual position to save local variable
59169. enter its scope
59170. stat -> repeatstat
59171. param -> NAME
59172. 'return' must be last statement
59173. move to next one
59174. create a 'jump to here' to close upvalues
59175. do not count last expression (unknown number of elements)
59176. loop scope ('break' jumps to this point)
59177. field -> listfield | recfield
59178. vararg
59179. regular case (not goto/break)
59180. true if 'block' is a loop
59181. main function is always declared vararg
59182. assignment -> ',' suffixedexp assignment
59183. index of first label in this block
59184. GRAMMAR RULES
59185. repeatstat -> REPEAT block UNTIL cond
59186. and that is it
59187. skip optional semicolon
59188. check for repeated labels on the same block
59189. **comment:** extra space to call generator
 label: code-design
59190. optional return values
59191. avoid default
59192. finish loop
59193. fieldsel -> '[' | ':'] NAME
59194. ====== ** Rules for Statements **
59195. forbody -> DO block
59196. will jump to label if condition is true
59197. read condition
59198. create new entry for this label
59199. call remove function and arguments and leaves (unless changed) one result
59200. false conditions finish the loop
59201. 'goto' is the entire block?
59202. ** prototypes for recursive non-terminal functions
59203. index of first pending goto in this block
59204. fieldsel
59205. error
59206. skip REPEAT

59207. table descriptor
59208. test_then_block -> [IF | ELSEIF] cond THEN block
59209. stat -> ifstat
59210. gen, state, control, plus at least one declared var
59211. recfield -> (NAME | ['exp1']) = exp1
59212. right priority
59213. ** adds a new prototype into list of prototypes
59214. parlist -> [param { ';' param }]
59215. no more items pending
59216. ** codes instruction to create new closure in parent function. ** The OP_CLOSURE instruction must use the last available register, ** so that, if it invokes the GC, the GC knows which registers ** are in use at that time.
59217. at least one expression
59218. read sub-expression with higher priority
59219. whilestat -> WHILE cond DO block END
59220. loop block
59221. read condition (inside scope block)
59222. try existing upvalues
59223. must use 'seminfo' before 'next'
59224. primaryexp -> NAME | '(' expr ')'
59225. do not need barrier here
59226. priority for unary operators
59227. create 'self' parameter
59228. table is the upvalue/local being assigned now?
59229. body -> '(' parlist ')' block END
59230. expand while operators have priorities higher than 'limit'
59231. scope for declared variables
59232. this one must exist
59233. # active locals outside the block
59234. close last expression
59235. '[' exp1 ']'
59236. handle goto/break
59237. listfield -> exp
59238. forlist -> NAME {,NAME} IN explist forbody
59239. declared vararg
59240. base register for call
59241. skip break
59242. Mark block where variable at given level was defined (to emit close instructions later).
59243. local will be used as an upval
59244. index of new label being created
59245. total number of 'record' elements
59246. first variable name
59247. semantic error
59248. copy upvalue/local value to a temporary (in position 'extra')
59249. exit list for finished parts
59250. funcargs -> constructor
59251. return all values
59252. number of array elements pending to be stored
59253. skip LOCAL
59254. not found?
59255. set initial table size
59256. and, or
59257. ==, <, <=
59258. found?
59259. default assignment
59260. skip FUNCTION
59261. funcargs -> STRING
59262. stat -> localstat
59263. will be a new upvalue
59264. !' NAME funcargs
59265. skip IF or ELSEIF
59266. close pending breaks
59267. total number of array elements
59268. }=====

59269. explist -> expr { ',' expr }
59270. final return
59271. fix it at stack top
59272. '<<' '>>'
59273. assigning to a table?
59274. '&' '|' '^'
59275. ifstat -> IF cond THEN block {ELSEIF cond THEN block} [ELSE block] END
59276. parse main body
59277. create main closure
59278. '*' '%'
59279. get environment variable
59280. floating constant; nval = numerical float value
59281. constant nil
59282. whether 't' is register (VLOCAL) or upvalue (VUPVAL)
59283. constant false
59284. chain of current blocks
59285. local variable; info = local register
59286. number of elements in 'p'
59287. list of labels or gotos
59288. expression is a test/comparison; info = pc of corresponding jump instruction
59289. number of elements in 'k'
59290. for indexed variables (VINDEXED)
59291. label identifier
59292. array
59293. ** \$Id: lparser.h,v 1.76 2015/12/30 18:16:13 roberto Exp \$ ** Lua Parser ** See Copyright Notice in lua.h

59294. ** Expression and variable descriptor. ** Code generation for variables and expressions can be delayed to allow ** optimizations; An 'expdesc' structure describes a potentially-delayed ** variable/expression. It has a description of its "main" value plus a ** list of conditional jumps that can also produce its value (generated ** by short-circuit operators 'and'/'or').

59295. list of pending jumps to 'pc'

59296. upvalue variable; info = index of upvalue in 'upvalues'

59297. number of elements in 'f->locvars'

59298. current function header

59299. kinds of variables/expressions

59300. constant in 'k'; info = index of constant in 'k'

59301. for generic use

59302. when 'expdesc' describes the last expression a list, this kind means an empty list (so, no expression)

59303. dynamic structures used by the parser

59304. description of active local variable

59305. expression has its value in a fixed register; info = result register

59306. variable index in stack

59307. number of entries in use

59308. lexical state

59309. first free register

59310. enclosing function

59311. expression can put result in any register; info = instruction pc

59312. description of pending goto statements and label statements

59313. local level where it appears in current block

59314. indexed variable; ind.vt = whether 't' is register or upvalue; ind.t = table register or upvalue; ind.idx = key's R/K index

59315. for VKFLT

59316. 'label' of last 'jump label'

59317. next position to code (equivalent to 'ncode')

59318. vararg expression; info = instruction pc

59319. for VKINT

59320. integer constant; nval = numerical integer value

59321. table (register or upvalue)

59322. patch list of 'exit when false'

59323. list of active local variables

59324. state needed to generate code for a given function

59325. expression is a function call; info = instruction pc

59326. constant true

59327. patch list of 'exit when true'

59328. array size

59329. number of active local variables

59330. number of upvalues

59331. list of pending gotos

59332. position in code

59333. defined in lparser.c

59334. control of blocks

59335. list of active labels

59336. index of first local var (in Dyndata array)

59337. index (R/K)

59338. line where it appeared

59339. **comment:** avoid warnings about ISO C functions
 label: code-design

59340. use -D_XOPEN_SOURCE=0 to undefine it

59341. ** \$Id: lprefix.h,v 1.2 2014/12/29 16:54:13 roberto Exp \$ ** Definitions for Lua code that must come before any other header file ** See Copyright Notice in lua.h

59342. ** Allows manipulation of large files in gcc and some other compilers

59343. {

59344. ** Windows stuff

59345. ** Allows POSIX/XSI stuff

59346. }

59347. anchor it on L stack

59348. no GC while building state

59349. erase new stack

59350. public function

59351. free next

59352. allow gc

59353. memory allocation error: free partial state

59354. will make 'totalbytes == MAX_LMEM'

59355. collect all objects

59356. registry[LUA_RIDX_GLOBALS] = table of globals

59357. initialize L1 extra space

59358. heap variable

59359. create registry

59360. close all upvalues for this thread

59361. next's next

59362. create new thread

59363. only the main thread can be closed

59364. ** Compute an initial seed as random as possible. Rely on Address Space ** Layout Randomization (if present) to increase randomness..

59365. 200%

59366. initialize first ci

59367. ** thread state + extra space

59368. ** Create registry table and its predefined values

59369. remove 'next' from the list

59370. while there are two nexts

59371. stack not completely built yet

59372. registry[LUA_RIDX_MAINTHREAD] = L

59373. closing a fully built state?

59374. global variable

59375. local variable

59376. ** free half of the CallInfo structures not in use by a thread

59377. 'function' entry for this 'ci'

59378. keep next's next
59379. free stack array
59380. link it on list 'allgc'
59381. GC runs 'twice the speed' of memory allocation
59382. init stack
59383. ** a macro to help the creation of a unique random seed when a state is ** created; the seed is used to randomize hashes.
59384. thread has no upvalues
59385. ** Main thread combines a thread state and the global state
59386. ** free all CallInfo structures not in use by a thread
59387. ** \$Id: lstate.c,v 2.133 2015/11/13 12:16:51 roberto Exp \$ ** Global State ** See Copyright Notice in lua.h
59388. ** open parts of the state that may cause memory-allocation errors. ** ('g->version' != NULL flags that the state was completely build)
59389. free the entire 'ci' list
59390. ** set GCdebt to a new value keeping the value (totalbytes + GCdebt) ** invariant (and avoiding underflows in 'totalbytes')
59391. free main block
59392. temp = L
59393. temp = new table (global table)
59394. initialize stack array
59395. ** preinitialize a thread with consistent values without allocating ** any memory (to avoid errors)
59396. metatables for basic types
59397. thread
59398. to be called in unprotected errors
59399. current error recover point
59400. ** 'per thread' state
59401. assume that CIST_OAH has offset 0 and that 'v' is strictly 0/1
59402. randomized seed for hashes
59403. call was tail called
59404. number of elements
59405. function index in the stack
59406. ** Information about a call. ** When a thread yields, 'func' is adjusted to pretend that the ** top function has only the yielded values in its stack; in that ** case, the actual 'func' value is saved in field 'extra'. ** When a function calls another with a continuation, 'extra' keeps ** the function index so that, in case of errors, the continuation ** function can be called with the correct top.
59407. **comment:** call is a yieldable protected call
 label: code-design
59408. list of gray objects
59409. an estimate of the non-garbage memory in use
59410. number of finalizers to call in each GC step
59411. number of bytes currently allocated - GCdebt
59412. list of tables with weak values
59413. list of open upvalues in this stack
59414. list of collectable objects with finalizers
59415. list of all-weak tables
59416. last free slot in the stack
59417. call is running a Lua function
59418. stack base
59419. macro to convert a Lua object into a GCObject
59420. true if GC is running
59421. cache for strings in API
59422. ** Bits in CallInfo status
59423. ** Some notes about garbage-collected objects: All objects in Lua must ** be kept somehow accessible until being freed, so all objects always ** belong to one (and only one) of these lists, using field 'next' of ** the 'CommonHeader' for the link: ** ** 'allgc': all objects not marked for finalization; ** 'finobj': all objects marked for finalization; ** 'tobefnz': all objects ready to be finalized; ** 'fixedgc': all objects that are not to be collected (currently ** only small strings, such as reserved words).
59424. current error handling function (stack index)
59425. number of non-yieldable calls in stack
59426. list of all collectable objects
59427. extra stack space to handle TM calls and some other extras
59428. state of garbage collector
59429. actual number of total bytes allocated
59430. bytes allocated not yet compensated by the collector
59431. kinds of Garbage Collection
59432. defined in ldo.c
59433. array with tag-method names
59434. gc was forced by an allocation failure
59435. last hook called yielded
59436. only for Lua functions
59437. size of pause between successive GCs
59438. base for this function
59439. first free slot in the stack
59440. list of threads with open upvalues
59441. kind of GC running
59442. memory-error message
59443. only for C functions
59444. CallInfo for first level (C calling Lua)
59445. current position of sweep in list
59446. list of userdata to be GC
59447. list of objects to be traversed atomically
59448. macros to convert a GCObject into a specific value
59449. hash table for strings
59450. last pc traced
59451. auxiliary data to 'frealloc'
59452. number of items in 'ci' list
59453. ** Union of all collectable objects (only for conversions)
59454. using __lt for __le
59455. top for this function
59456. call is running on a fresh invocation of luaV_execute
59457. original value of 'allowhook'
59458. expected number of results from this function
59459. call is running a debug hook

59460. continuation in case of yields
59461. dynamic call link
59462. ** 'global state', shared by all threads of this state
59463. list of ephemeron tables (weak keys)
59464. common header
59465. **comment:** ** Atomic type (relative to signals) to better ensure that 'lua_sethook' ** is thread safe
 label: requirement
59466. call info for current function
59467. pointer to version number
59468. list of objects not to be collected
59469. context info. in case of yields
59470. GC 'granularity'
59471. number of nested C calls
59472. function to reallocate memory
59473. memory traversed by the GC
59474. ** \$Id: lstate.h,v 2.130 2015/12/16 16:39:38 roberto Exp \$ ** Global State ** See Copyright Notice in lua.h
59475. ** equality for long strings
59476. ** \$Id: lstring.c,v 2.56 2015/11/23 11:32:51 roberto Exp \$ ** String table (keeps all strings handled by Lua) ** See Copyright Notice in lua.h
59477. dead (but not collected yet)?
59478. same instance or...
59479. short string?
59480. shrink table if needed
59481. remove element from its list
59482. normal route
59483. chain it
59484. ** Clear API string cache. (Entries cannot be empty, so fill them with ** a non-collectable string.)
59485. total size of TString object
59486. rehash
59487. equal length and ...
59488. new element is first in the list
59489. otherwise 'memcmp'/'memcpy' are undefined
59490. no hash?
59491. ** Lua will use at most ~(2^LUAI_HASHLIMIT) bytes from a string to ** compute its hash
59492. ** checks whether short string exists and reuses it or creates a new one
59493. find previous element
59494. move out last element
59495. ** Initialize the string table and the string cache
59496. ** new string (with explicit length)
59497. ending 0
59498. found!
59499. ** resizes the string table
59500. fill cache with valid strings
59501. hash
59502. resurrect it
59503. it should never be collected
59504. that is it
59505. ** Create or reuse a zero-terminated string, first checking in the ** cache (using the string address as a key). The cache can contain ** only zero-terminated strings, so it is safe to use 'strcmp' to ** check hits.
59506. hit?
59507. new position
59508. now it has its hash
59509. grow table if needed
59510. vanishing slice should be empty
59511. replace it with something fixed
59512. ** creates a new string object
59513. save next
59514. recompute with new size
59515. will entry be collected?
59516. initial size of string table
59517. pre-create memory-error message
59518. for each node in the list
59519. equal contents
59520. ** equality for short strings, which are always internalized
59521. ** test whether a string is a reserved word
59522. ** \$Id: lstring.h,v 1.61 2015/11/03 15:36:01 roberto Exp \$ ** String table (keep all strings handled by Lua) ** See Copyright Notice in lua.h
59523. first byte
59524. 1st char will be checked by 'memchr'
59525. equal to '%g'
59526. 0 or more repetitions
59527. end ('\0') of source string
59528. return match(ms, s, ep + 1);
59529. accept empty?
59530. keep entire string
59531. number of bits in a character
59532. replacement type
59533. ensure it uses a dot
59534. correct 'l1' and 's1' to try again
59535. no dot?
59536. size of a lua_Integer
59537. fixed-length strings
59538. get string library
59539. integers
59540. set table as metatable for strings
59541. not found
59542. may overflow?
59543. ** {===== ** PATTERN MATCHING **

59544. change it to a dot

59545. state for 'gmatch'
59546. end of pattern?
59547. no modifiers?
59548. ** Ensures the 'buff' string uses a dot as the radix character.
59549. otherwise, skip one character
59550. add capture to accumulated result
59551. remove result from 'luaL_tolstring'
59552. accumulate total size of result
59553. 1st char is already checked
59554. zero-terminated string
59555. add zero at the end
59556. end of last match
59557. using goto's to optimize tail recursion
59558. can be -0...
59559. 'X' gets alignment from following option
59560. start
59561. position capture?
59562. is 'align' not a power of 2?
59563. {
59564. ** information to pack/unpack stuff
59565. mark to separate arguments from string buffer
59566. end capture
59567. pop metatable
59568. to store the format ('%...')
59569. ** maximum number of captures that a pattern can do during ** pattern-matching. This limit is arbitrary, but must fit in ** an unsigned char.
59570. no suffix
59571. true iff machine is little endian
59572. return match(ms, s + 1, ep);
59573. else use default format
59574. }=====

59575. value used for padding
59576. no; go to default
59577. also treat cases 'pnLlh'
59578. 0 or more repetitions (minimum)
59579. format item
59580. points to optional suffix
59581. end of subject
59582. is number negative?
59583. ** maximum size of each format specification (such as "%-099.99d")
59584. add whole match
59585. ** add length modifier into formats
59586. replacement count
59587. balanced string?
59588. pack length
59589. skip the '^'
59590. may have more after \0
59591. write as hexa ('%a')
59592. stack space for item + next position
59593. maximum recursion depth for 'match'
59594. recursive function
59595. no number?
59596. add "0x"
59597. need no alignment?
59598. number of results
59599. matches any char
59600. deprecated option
59601. points to what is next
59602. character count
59603. ** Read and classify next option. 'size' is filled with option's size.
59604. keep original text
59605. pop dummy string
59606. padding for alignment
59607. 's2' cannot be found after that
59608. ** Some sizes are better limited to fit in 'int', but must also fit in ** 'size_t'. (We assume that 'lua_Integer' cannot be smaller than 'int'.)
59609. to put formatted item
59610. ** Read an integer numeral and raises an error if it is larger ** than the maximum size for integers.
59611. fixed-size string
59612. get argument
59613. dummy structure to get native alignment requirements
59614. negative number need sign extension?
59615. empty interval; return no values
59616. ** Number of bits that goes into the first digit. It can be any value ** between 1 and 4; the following definition tries to align the number ** to nibble boundaries by making what is left after that first digit a ** multiple of 4.
59617. return match(ms, s, p + 4);
59618. mask for one character (NB 1's)
59619. string ends out of balance
59620. 1 or more repetitions
59621. else fail (s == NULL)
59622. ** Initialize Header
59623. check whether pattern has no special characters
59624. remove original value
59625. enforce maximum alignment
59626. padding
59627. default
59628. dummy union to get native endianness
59629. ** Read, classify, and fill other details about the next option. ** 'psize' is filled with option's size, 'notoalign' with its ** alignment requirements. ** Local variable 'size' gets the size to be aligned. (Kpadal option ** always gets its full alignment, other options are limited by ** the maximum alignment ('maxalign'). Kchar option needs no alignment ** despite its size.)

59630. init of source string
59631. number of strings pushed
59632. do a plain search
59633. return match(ms, s, p + 2)
59634. else return match(ms, s, ep + 1);
59635. pattern class plus optional suffix
59636. subject
59637. add first digit
59638. signed integers
59639. 'x' fraction and exponent
59640. matched once
59641. empty strings are everywhere
59642. more digits?
59643. add replacement to buffer
59644. **comment:** no precision and string is too long to be formatted
 label: code-design
59645. is the '\$' the last char in pattern?
59646. zero-terminated strings
59647. return match(ms, s, ep);
59648. macro to 'unsign' a character
59649. FALLTHROUGH
59650. optional
59651. ** Union for serializing floats
59652. skip string plus final '\0'
59653. %%
59654. look for a ']'
59655. skip precision
59656. fill alignment
59657. get integer part from 'x'
59658. start after string's end?
59659. pattern has a special character
59660. end ('\0') of pattern
59661. ** Pack integer 'n' with 'size' bytes and 'islittle' endianness. ** The final 'if' handles the case when 'size' is larger than ** the size of a Lua integer, correcting the extra sign-extension ** bytes if necessary (by default they would be zeros).
59662. copy table
59663. ms->level == 0, too
59664. ** Unpack an integer with 'size' bytes and 'islittle' endianness. ** If size is smaller than the size of a Lua integer and integer ** is signed, must do sign extension (propagating the sign to the ** higher bits); if size is larger than the size of a Lua integer, ** it must check the unread bytes to see whether they do not cause an ** overflow.
59665. no special chars found
59666. skip width
59667. float?
59668. does not match at least once?
59669. translate a relative string position: negative means back from end
59670. first n-1 copies (followed by separator)
59671. last copy (not followed by separator)
59672. 1 match already done
59673. to search for a '*s2' inside 's1'
59674. end
59675. skip ESC
59676. ** Copy 'size' bytes from 'src' to 'dest', correcting endianness if ** given 'islittle' is different from native endianness.
59677. corner case?
59678. fail
59679. match failed
59680. add result to accumulator
59681. current argument to pack
59682. keep them on closure to avoid being collected
59683. (re)prepare state for new match
59684. use hexa
59685. undo capture
59686. total number of captures (finished or unfinished)
59687. ** Read an integer numeral from string 'fmt' or return 'df' if ** there is no numeral
59688. format string
59689. table to be metatable for strings
59690. handle optional suffix
59691. (2 digits at most)
59692. else didn't match; reduce 1 repetition to try again
59693. ** {===== ** STRING FORMAT **
=====**
59694. ** Hexadecimal floating-point formatter
59695. add as many digits as needed
59696. avoids a negative 'l1'
59697. add string
59698. valid flags in a format specification
59699. match?
59700. start capture
59701. return what is left
59702. correct extra bytes
59703. total space used by option
59704. add radix point
59705. make it positive
59706. undo increment
59707. skip alignment
59708. ** Open string library
59709. usually, alignment follows size
59710. maximum size for the binary representation of an integer
59711. return default value
59712. **comment:** call never return, but to avoid warnings:

label: code-design

59713. must check unread bytes
59714. escaped sequences not in the format class[*+?-]?
59715. no-op (configuration or spaces)
59716. needs sign extension?
59717. number of substitutions
59718. add result to buffer
59719. keeps trying to match with the maximum repetitions
59720. unsigned integers
59721. copy it into 'u'
59722. explicit request or no special characters?
59723. add signal
59724. if number, convert it to string
59725. check end of string
59726. capture results (%0-%9)?
59727. skip escapes (e.g. "%!")
59728. number of bytes in added item
59729. try with one more repetition
59730. counts maximum expand for item
59731. ** \$Id: lstrlib.c,v 1.251 2016/05/20 14:13:21 roberto Exp \$ ** Standard library for string operations and pattern-matching ** See Copyright Notice in lua.h
59732. frontier?
59733. }
59734. strings with prefixed length
59735. add to buffer
59736. skip flags
59737. ** options for pack/unpack
59738. this digit goes before the radix point
59739. next position
59740. try locale point
59741. create "0" or "-0" followed by exponent
59742. match failed?
59743. add exponent
59744. real size smaller than lua_Integer?
59745. strings with length count
59746. metatable.__index = string
59747. move 'u' to final result, correcting endianness if needed
59748. ** {===== ** PACK/UNPACK **
=====**
59749. pattern
59750. floating-point numbers
59751. need overflow check?
59752. cannot find anything
59753. number of arguments
59754. do sign extension
59755. skip anchor character
59756. floating-point options
59757. '+' or no suffix
59758. LUA_TNUMBER or LUA_TSTRING
59759. enough for any float type
59760. close capture
59761. arithmetic overflow?
59762. current position
59763. control for recursive depth (to avoid C stack overflow)
59764. max replacements
59765. format the string into 'buff'
59766. empty 'memcpy' is not that cheap
59767. nil or false?
59768. ** Add integer part of 'x' to buffer and return new 'x'
59769. inf or NaN?
59770. dummy string
59771. ** Maximum size of each formatted item. This maximum size is produced ** by format("%.99f", -maxfloat), and is equal to 99 + 3 ('-', '.', ** and '\0') + number of decimal digits to represent maxfloat (which ** is maximum exponent + 1). (99+3+1 then rounded to 120 for "extra ** expenses", such as locale-dependent stuff)
59772. pad extra space
59773. match state
59774. skip string
59775. shrink array
59776. cannot find a free place?
59777. i is zero or a present index
59778. 'key' is an appropriate array index
59779. no more elements to count
59780. compute new size for array part
59781. number of keys in the array part
59782. is 'key' inside array part?
59783. grow table
59784. 2^i (candidate for optimal size)
59785. whatever called 'newkey' takes care of TM cache
59786. not found
59787. count as such
59788. count extra key
59789. that's it
59790. main position is taken?
59791. copy colliding node into free pos. (mp->next also goes)
59792. find 'i' and 'j' such that i is present and j is not
59793. key index in hash table
59794. try first array part
59795. ** Compute the optimal size for the array part of table 't'. 'nums' is a ** "count array" where 'nums[i]' is the number of integers in the table ** between $2^{(i-1)} + 1$ and 2^i . 'pna' enters with the total number of ** integer keys in the table and leaves with the number of keys that ** will go to the array part; return the optimal size.

59796. ** nums[i] = number of keys 'k' where $2^{i-1} < k \leq 2^i$
59797. value
59798. insert it as an integer
59799. yes; that's the index
59800. traverse each slice
59801. use specialized version
59802. doesn't need barrier/invalidate cache, as entry was already present in the table
59803. all elements up to 'optimal' will go to array part
59804. rechain to point to 'f'
59805. adjust upper limit
59806. correct 'next'
59807. optimal size (till now)
59808. elements added to 'nums' (can go to array part)
59809. index is int?
59810. new node will go into free position
59811. first iteration
59812. ** main search function
59813. 'key' did not match some condition
59814. array part must grow?
59815. else must find a boundary in hash part
59816. re-insert elements from hash part
59817. array part must shrink?
59818. normal case
59819. ** beware: when using this function you probably need to check a GC ** barrier and invalidate the TM cache.
59820. table was built with bad purposes: resort to linear search
59821. key not found
59822. **comment:** ** for some types, it is better to avoid modulus by power of 2, as ** they tend to have many 2 factors.
label: code-design
59823. is colliding node out of its main position?
59824. is 'key' an appropriate array index?
59825. **comment:** ** Implementation of tables (aka arrays, objects, or hash tables). ** Tables keep its elements in two parts: an array part and a hash part. ** Non-negative integer keys are all candidates to be kept in the array ** part. The actual size of the array is the largest 'n' such that ** more than half the slots between 1 and n are in use. ** Hash uses a mix of chained scatter table with Brent's variation. ** A main invariant of these tables is that, if an element is not ** in its main position (i.e. the 'original' position that its hash gives ** to it), then the colliding element is in its own main position. ** Hence even when the load factor reaches 100%, performance remains good.
label: code-design
59826. loop while keys can fill more than half of total size
59827. could not find a free place
59828. find previous
59829. ** Try to find a boundary in table 't'. A 'boundary' is an integer index ** such that t[i] is non-nil and t[i+1] is nil (and 0 if t[1] is nil).
59830. FALLTHROUGH
59831. ** search function for short strings
59832. total number of elements
59833. ** returns the index of a 'key' for table traversals. First goes all ** elements in the array part, then elements in the hash part. The ** beginning of a traversal is signaled by 0.
59834. number of elements to go to array part
59835. use common 'dummynode'
59836. find original element
59837. hash part
59838. count to traverse all array keys
59839. reset counts
59840. save old hash ...
59841. optimal size for array part
59842. free old hash
59843. resize the table to new computed sizes
59844. ** search function for integers
59845. all those keys are integer keys
59846. ** returns the 'main' position of an element in a table (that is, the index ** of its hash value)
59847. colliding node is in its own main position
59848. $2^{\lfloor \lg \cdot \rfloor}$
59849. key
59850. ** Maximum size of hash part is 2^{MAXHBITS} . MAXHBITS is the largest ** integer such that 2^{MAXHBITS} fits in a signed int. (Note that the ** maximum number of elements in a table, $2^{\text{MAXABITS}} + 2^{\text{MAXHBITS}}$, still ** fits comfortably in an unsigned int.)
59851. count elements in range $(2^{\lfloor \lg \cdot \rfloor}, 2^{\lfloor \lg \cdot \rfloor + 1}]$
59852. a non-nil value?
59853. ** "Generic" get version. (Not that generic: not valid for integers, ** which may be in array part, nor for floats with integral values.)
59854. counter
59855. count keys in array part
59856. now do a binary search between them
59857. chain new position
59858. is 'n' inf/-inf/NaN?
59859. ** Maximum size of array part (MAXASIZE) is 2^{MAXABITS} . MAXABITS is ** the largest integer such that MAXASIZE fits in an unsigned int.
59860. summation of 'nums'
59861. ** inserts a new key into a hash table; first, check whether key's main ** position is free. If not, check whether colliding node is in its main ** position or not: if it is not, move colliding node to an empty place and ** put new key in its main position; otherwise (colliding node is in its main ** position), new key goes to an empty position.
59862. overflow?
59863. no elements to hash part?
59864. all positions are free
59865. re-insert elements from vanishing slice
59866. hash elements are numbered after array ones
59867. ** \$Id: ltable.c,v 2.117 2015/11/19 19:16:22 roberto Exp \$ ** Lua tables (hash) ** See Copyright Notice in lua.h
59868. now 'mp' is free
59869. there is a boundary in the array part: (binary) search for it
59870. more than half elements present?
59871. ** ======
59872. that is easy...
59873. hash part is empty?

59874. insert key into grown table
59875. **comment:** key may be dead already, but it is ok to use it in 'next'
 label: code-design
59876. **** {=====** ** Rehash **
=====

59877. ** Hash for floating-point numbers. ** The main computation should be just ** n = frexp(n, &i); return (n * INT_MAX) + i ** but there are some numerical subtleties. ** In a two-complement representation, INT_MAX does not have an exact ** representation as a float, but INT_MIN does; because the absolute ** value of 'frexp' is smaller than 1 (unless 'n' is inf/NaN), the ** absolute value of the product 'frexp * -INT_MIN' is smaller or equal ** to INT_MAX. Next, the use of 'unsigned int' avoids overflows when ** adding 'i'; the use of '~u' (instead of '-u') avoids problems with ** INT_MIN.
59878. (1 <= key && key <= t->sizearray)
59879. create new hash part with appropriate size
59880. check whether 'key' is somewhere in the chain
59881. ** Count keys in array part of table 't': Fill 'nums[i]' with ** number of keys that will go into corresponding slice and return ** total number of non-nil keys.
59882. else...
59883. number of elements smaller than 2^i
59884. **comment:** for long strings, use generic case
 label: code-design
59885. ** returns the index for 'key' if 'key' is an appropriate key to live in ** the array part of the table, 0 otherwise.
59886. yes; move colliding node into free position
59887. count keys in hash part
59888. no more elements
59889. get a free place
59890. returns the key, given the value of a table entry
59891. **comment:** 'const' to avoid wrong writings that can mess up field 'next'
 label: code-design
59892. ** writable version of 'gkey'; allows updates to individual fields, ** but not to the whole (which has incompatible type)
59893. ** \$Id: ltable.h,v 2.21 2015/11/03 15:47:30 roberto Exp \$ ** Lua tables (hash) ** See Copyright Notice in lua.h
59894. next loop: repeat --j while P < a[j]
59895. result = t[pos]
59896. range/4
59897. t.n = number of elements
59898. otherwise, nothing to move
59899. lower interval is smaller?
59900. read
59901. number of elements to pack
59902. to be used later
59903. function
59904. move up elements
59905. ** Choose an element in the middle (2nd-3rd quarters) of [lo,up] ** "randomized" by 'rnd'
59906. non-trivial interval?
59907. ** Use 'time' and 'clock' as sources of "randomness". Because we don't ** know the types 'clock_t' and 'time_t', we cannot cast them to ** anything without risking overflows. A safe way to use their values ** is to copy them to an array of a known type and use the array values.
59908. arrays larger than 'RANLIMIT' may use randomized pivots
59909. only 2 elements?
59910. 2nd argument is the position
59911. length
59912. call function
59913. {
59914. Pivot index
59915. write
59916. tail call for [lo .. p - 1] (lower interval)
59917. read/write
59918. =====
59919. t[pos] = v
59920. a[lo .. p - 1] <= a[p] == P <= a[p + 1 .. up]
59921. push a[up - 1]
59922. must have metatable
59923. where to insert new element
59924. pop a[j]
59925. pop result
59926. insert new element at the end
59927. remove a[j]
59928. -2 to compensate function and 'a'
59929. sort elements 'lo', 'p', and 'up'
59930. get middle element (Pivot)
59931. **comment:** ** Operations that an object must define to mimic a table ** (some functions only need some of them)
 label: code-design
59932. is it not a table?
59933. ** Copy elements (1[f], ..., 1[e]) into (tt[t], tt[t+1], ...). Whenever ** possible, copy in increasing order, which is better for rehashing. ** "possible" means destination after original range, or smaller ** than origin, or copying to another table.
59934. try a new randomization
59935. make sure there are two arguments
59936. small interval or no randomize?
59937. type for array indices
59938. t[pos] = nil
59939. push function
59940. first key
59941. is there a 2nd argument?
59942. add last value (if interval was not empty)
59943. tail call for [p + 1 .. up] (upper interval)
59944. will be decremented before first use
59945. destination table
59946. swap a[up] - a[p]
59947. remove a[lo]
59948. no elements out of place?
59949. call recursively for lower interval
59950. assign elements
59951. swap Pivot (a[p]) with a[up - 1]

59952. ** {===== ** Quicksort ** (based on 'Algorithms in MODULA-3', Robert Sedgewick; **
 Addison-Wesley, 1993.) ====== **
 59953. t[pos] = t[pos + 1]
 59954. push last element
 59955. ** Produce a "random" 'unsigned int' to randomize pivot choice. This ** macro is used only when 'sort' detects a big imbalance in the result ** of a partition. (If
 you don't want/need this "randomness", ~0 is a ** good choice.)
 59956. remove a[i]
 59957. after the loop, a[j] <= P and a[j + 1 .. up] >= P
 59958. loop for tail recursion
 59959. must be a function
 59960. ** {===== ** Pack/unpack **
 ====== **
 59961. swap a[lo] - a[up]
 59962. push Pivot
 59963. size of smaller interval
 59964. size of 'e' measured in number of 'unsigned int's
 59965. partition too imbalanced?
 59966. ** Return true iff value at stack index 'a' is less than the value at ** index 'b' (according to the order of the sort).
 59967. tail call auxsort(L, lo, up, rnd)
 59968. pop metatable and tested metamethods
 59969. -1 to compensate function
 59970. ** \$Id: ltablib.c,v 1.93 2016/02/25 19:41:54 roberto Exp \$ ** Library for Table Manipulation ** See Copyright Notice in lua.h
 59971. ** QuickSort algorithm (recursive function)
 59972. next loop: repeat ++i while a[i] < P
 59973. after the loop, a[i] >= P and a[lo .. i - 1] < P
 59974. a[up] < a[p]?
 59975. return table
 59976. for larger intervals, it is worth a random pivot
 59977. only 3 elements?
 59978. a < b
 59979. already sorted
 59980. t[i] = t[i - 1]
 59981. loop invariant: a[lo .. i] <= P <= a[j .. up]
 59982. ** Does the partition: Pivot P is at the top of the stack. ** precondition: a[lo] <= P == a[up-1] <= a[up], ** so it only needs to do the partition from lo + 1 to up - 2.
 ** Pos-condition: a[lo .. i - 1] <= a[i] == P <= a[i + 1 .. up] ** returns 'i'.
 59983. push arg[i..e - 1] (to avoid overflows)
 59984. create result table
 59985. a[up] < a[lo]?
 59986. swap a[p] - a[lo]
 59987. a[p] < a[lo]?
 59988. otherwise, swap a[i] - a[j] to restore invariant and repeat
 59989. a[lo .. i - 1] <= P <= a[j + 1 .. i .. up]
 59990. called with only 2 arguments
 59991. empty range
 59992. middle element is a good pivot
 59993. }
 59994. remove value
 59995. remove both values
 59996. number of elements to move
 59997. first empty element
 59998. ** Check that 'arg' either is a table or can behave like one (that is, ** has a metatable with the required metamethods)
 59999. number of elements to pop
 60000. j < i but a[j] > P ??
 60001. a[i] < P but a[up - 1] == P ??
 60002. put it at index 1
 60003. swap pivot (a[up - 1]) with a[i] to satisfy pos-condition
 60004. return destination table
 60005. no function?
 60006. call recursively for upper interval
 60007. will be incremented before first use
 60008. get result
 60009. validate 'pos' if given
 60010. _G.unpack = table.unpack
 60011. force an error
 60012. number of elements minus 1 (avoid overflows)
 60013. try first operand
 60014. **comment:** call never returns, but to avoid warnings:
 label: code-design
 60015. **comment:** calls never return, but to avoid warnings:
 label: code-design
 60016. never collect these names
 60017. no metamethod
 60018. **comment:** ** function to be used with macro "fasttm": optimized for absence of ** tag methods
 label: code-design
 60019. cache this fact
 60020. no tag method?
 60021. metamethod may yield only when called from Lua code
 60022. is '__name' a string?
 60023. ** Return the name of the type of an object. For tables and userdata ** with metatable, use their '__name' metafield, if present.
 60024. use it as type name
 60025. else use standard type name
 60026. try second operand
 60027. ** \$Id: ltm.c,v 2.37 2016/02/26 19:20:15 roberto Exp \$ ** Tag methods ** See Copyright Notice in lua.h
 60028. this last case is used for tests only
 60029. push function (assume EXTRA_STACK)
 60030. 2nd argument
 60031. if has result, move it to its place
 60032. no result? 'p3' is third argument

60033. 1st argument
60034. FALLTHROUGH
60035. ORDER TM
60036. 3rd argument
60037. last tag method with fast access
60038. ** \$Id: ltm.h,v 2.22 2016/02/26 19:20:15 roberto Exp \$ ** Tag methods ** See Copyright Notice in lua.h
60039. * WARNING: if you change the order of this enumeration, * grep "ORDER TM" and "ORDER OP"
60040. number of elements in the enum
60041. ** lua_stdin_is_tty detects whether the standard input is a 'tty' (that ** is, whether we're running lua interactively).
60042. number of positive indices
60043. repeat until gets a complete statement
60044. to be available to 'laction'
60045. ** Calls 'require(name)' and stores the result in a global variable ** with the given name.
60046. push message handler
60047. remove it
60048. execute arguments -e and -l
60049. ** Read multiple lines until a complete Lua statement
60050. ** Message handler used to run all chunks
60051. get line
60052. no concatenated argument?
60053. ** Prints (calling the Lua 'print' function) any values on the stack
60054. keep history
60055. -v
60056. script "name" is '-'
60057. that is the message
60058. no 'progname' on errors in interactive mode
60059. bad option
60060. {
60061. ** \$Id: lua.c,v 1.226 2015/08/14 19:11:20 roberto Exp \$ ** Lua stand-alone interpreter ** See Copyright Notice in lua.h
60062. mark in error messages for incomplete statements
60063. change '=' to 'return'
60064. remove prompt
60065. try alternative name
60066. check that interpreter has correct version
60067. create table 'arg'
60068. reset hook
60069. original line
60070. '--'
60071. remove table from the stack
60072. already checked
60073. ** Create the 'arg' table, which stores all arguments from the ** command line ('argv'). It should be aligned so that, at index 0, ** it has 'argv[script]', which is the script name. The arguments ** to the script (everything after 'script') go to positive indices; ** other arguments (before the script name) go to negative indices. ** If there is no script name, assume interpreter's name as base.
60074. that produces a string?
60075. -i option?
60076. stop handling options
60077. get what it has
60078. }{
60079. create state
60080. '
60081. line ends with newline?
60082. if another SIGINT happens, terminate process
60083. reset C-signal handler
60084. -E
60085. ** Check whether 'status' is not OK and, if so, prints the error ** message on the top of the stack. It assumes that the error object ** is a string, as it was either generated by Lua or by 'msghandler'.
60086. -e
60087. non empty?
60088. else check option
60089. ** Read a line and try to load (compile) it first as an expression (by ** adding "return " in front of it) and second as a statement. Return ** the final status of load/call with the resulting function (if any) ** in the top of the stack.
60090. assume stdin is a tty
60091. ** Prints an error message, adding the program name in front of it ** (if present)
60092. set C-signal handler
60093. **comment:** bits of various argument indicators in 'args'
 label: code-design
60094. remove message
60095. add newline...
60096. remove modified line
60097. FALLTHROUGH
60098. does it have a metamethod
60099. no script name
60100. show prompt
60101. not an option?
60102. extra characters after '--'?
60103. no next argument or it is another option
60104. ** Push on the stack the contents of table 'arg' from 1 to #arg
60105. (-i implies -v)
60106. executes stdin as a file
60107. no input (prompt will be popped by caller)
60108. ** Try to compile line on the stack as 'return <line>;' on return, stack ** has either compiled chunk or original line (if compilation failed).
60109. try it
60110. function index
60111. ** Check whether 'status' signals a syntax error and the error ** message at the top of the stack ends with the above mark for ** incomplete statements.
60112. pop result from 'luaL_loadbuffer' and modified line
60113. error running LUA_INIT
60114. bad arg?
60115. ** Interface to 'lua_pcall', which sets appropriate message function ** and C-signal handler. Used to run all chunks.

60116. join them
60117. remove line from the stack
60118. remove message handler from the stack
60119. return the traceback
60120. no script name?
60121. ** lua_readline defines how to show a prompt and then read a line from ** the standard input. ** lua_saveline defines how to "save" a read line in a "history". **
 lua_freeline defines how to free a line read by lua_readline.
60122. execute main script (if there is one)
60123. running in interactive mode?
60124. cannot or should not try to add continuation line
60125. **comment:** ** Traverses all arguments from 'argv', returning a mask with those ** needed before running any Lua code (or an error code if it finds ** any invalid argument). 'first' returns the first not-handled argument ** (either the script name or a bad argument in case of error).
 label: code-design
60126. signal no errors
60127. **comment:** 'script' has index of bad arg.
 label: code-design
60128. do read-eval-print loop
60129. option '-v'?
60130. ** Processes options 'e' and 'l', which involve running Lua code. ** Returns 0 if some code raises an error.
60131. ** Hook set by signal function to stop the interpreter.
60132. option '-E'?
60133. ** Returns the string to be used as a prompt by the interpreter.
60134. ...between the two lines
60135. ** Do the REPL: repeatedly read (load) a line, evaluate (call) it, and ** print any results.
60136. **comment:** unused arg.
 label: code-design
60137. signal for libraries to ignore env. vars.
60138. 'return ...' did not work?
60139. stdin
60140. push arguments to script
60141. ** Main body of stand-alone interpreter (to be called in protected mode). ** Reads the options and handles them all.
60142. run LUA_INIT
60143. any result to be printed?
60144. no option '-E'?
60145. invalid option
60146. ** Prompt the user, read a line, and push it into the Lua stack.
60147. ** Function to be called at a C signal. Because a C signal cannot ** just change a Lua state (as there is no proper synchronization), ** this function only sets a hook that, when called, will stop the ** interpreter.
60148. something failed
60149. }
60150. both options need an argument
60151. to call 'pmain' in protected mode
60152. is error object not a string?
60153. -i
60154. call 'require(name)'
60155. try next 'argv'
60156. do the call
60157. for compatibility with 5.2, ...
60158. ISO C definition
60159. append a standard traceback
60160. no arguments?
60161. put it under function and args
60162. else...
60163. clear stack
60164. 2nd argument
60165. extra characters after 1st?
60166. global[name] = require return
60167. get result
60168. try as command, maybe with continuation lines
60169. 1st argument
60170. no input
60171. open standard libraries
60172. (t)
60173. minimum Lua stack available to a C function
60174. ** basic types
60175. ** push functions (C -> stack)
60176. (u) number of parameters
60177. ** Pseudo-indices ** (-LUAI_MAXSTACK is the minimum valid index; we keep some free empty ** space after that to help overflow detection)
60178. Functions to be called by the debugger in specific events
60179. mark for precompiled code ('<esc>Lua')
60180. unsigned integer type
60181. ** Type for C functions registered with Lua
60182. ** access functions (stack -> C)
60183. ** garbage-collection function and options
60184. ***** Copyright (C) 1994-2016 Lua.org, PUC-Rio. ** Permission is hereby granted, free of charge, to any person obtaining * a copy of this software and associated documentation files (the * "Software"), to deal in the Software without restriction, including * without limitation the rights to use, copy, modify, merge, publish, * distribute, sublicense, and/or sell copies of the Software, and to * permit persons to whom the Software is furnished to do so, subject to * the following conditions: * * The above copyright notice and this permission notice shall be * included in all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

60185. (S)
60186. ** \$Id: lua.h,v 1.331 2016/05/30 15:53:28 roberto Exp \$ ** Lua - A Scripting Language ** Lua.org, PUC-Rio, Brazil (<http://www.lua.org>) ** See Copyright Notice at the end of this file
60187. ** set functions (stack -> Lua)

60188. ()
60189. ** state manipulation
60190. ** Type for memory-allocation functions
60191. predefined values in the registry
60192. == compatibility macros for unsigned conversions ==
60193. active function
60194. (n) 'global', 'local', 'field', 'method'
60195. ** Type for continuation functions
60196. type for continuation-function contexts
60197. ** Event codes
60198. private part
60199. == Debug API ==
60200. type of numbers in Lua
60201. type for integer functions
60202. (u)
60203. ** get functions (Lua -> stack)
60204. (u) number of upvalues
60205. == some useful macros ==
60206. ==
60207. ** Type for functions that read/write blocks when loading/dumping Lua chunks
60208. ** 'load' and 'call' functions (load and run Lua code)
60209. option for multiple returns in 'lua_pcall' and 'lua_call'
60210. ** coroutine functions
60211. **comment:** ** generic extra include file
 label: code-design
60212. ** Comparison and arithmetic functions
60213. ** Event masks
60214. ==
60215. ** miscellaneous functions
60216. ** basic stack manipulation
60217. (n)
60218. activation record
60219. (S) 'Lua', 'C', 'main', 'tail'
60220. thread status
60221. ** RCS ident string
60222. ORDER TM, ORDER OP
60223. lua.hpp Lua header files for C++ <<extern "C">> not supplied automatically because Lua also compiles as C++
60224. show version
60225. ** \$Id: luac.c,v 1.75 2015/03/12 01:58:27 lhf Exp \$ ** Lua compiler (saves bytecodes to files; also lists bytecodes) ** See Copyright Notice in lua.h
60226. end of options; use stdin
60227. list bytecodes?
60228. cannot happen
60229. default program name
60230. ** \$Id: luac.c,v 1.75 2015/03/12 01:58:27 lhf Exp \$ ** print bytecodes ** See Copyright Notice in lua.h
60231. end of options; skip it
60232. default output file
60233. unknown option
60234. actual program name
60235. default output file name
60236. strip debug information?
60237. end of options; keep it
60238. list
60239. actual output file name
60240. output file
60241. parse only
60242. strip debug information
60243. dump bytecodes?
60244. #define LUA_COMPAT_FLOATSTRING
60245. == Configuration for Numbers. ** Change these definitions if no predefined LUA_FLOAT_* / LUA_INT_* ** satisfy your needs. ==
60246. @@ LUA_API is a mark for all core API functions. @@ LUALIB_API is a mark for all auxiliary library functions. @@ LUAMOD_API is a mark for all standard library opening functions. ** CHANGE them if you need to define those functions in some special way. ** For instance, if you want to create one Windows DLL with the core and ** the libraries, you may want to use the following definition (define ** LUA_BUILD_AS_DLL to get it).
60247. @@ LUAI_MAXSTACK limits the size of the Lua stack. ** CHANGE it if you need a different limit. This limit is arbitrary; ** its only purpose is to stop Lua from consuming unlimited stack ** space (and to reserve some numbers for pseudo-indices).
60248. ==
60249. }{ long double
60250. @@ l_sprintf is equivalent to 'snprintf' or 'sprintf' in C89. ** (All uses in Lua have only one format item.)
60251. @@ LUA_COMPAT_MODULE controls compatibility with previous ** module functions 'module' (Lua) and 'luaL_register' (C).
60252. no variant
60253. MacOS does not need -ldl
60254. {
60255. @@ LUA_NOCVTN2S/LUA_NOCVTS2N control how Lua performs some ** coercions. Define LUA_NOCVTN2S to turn off automatic coercion from ** numbers to strings. Define LUA_NOCVTS2N to turn off automatic ** coercion from strings to numbers.
60256. ==
60257. @@ LUA_COMPAT_IPAIRS controls the effectiveness of the __ipairs metamethod.
60258. == Search for "{{--}} to find all configurable definitions. ==
60259. @@ LUA_COMPAT_LOADSTRING defines the function 'loadstring' in the base ** library. You can rewrite 'loadstring(s)' as 'load(s)'.
60260. @@ LUAI_FUNC is a mark for all extern functions that are not to be ** exported to outside modules. @@ LUAI_DDEF and LUAI_DDEC are marks for all extern (const) variables ** that are not to be exported to outside modules (LUAI_DDEF for ** definitions and LUAI_DDEC for declarations). ** CHANGE them if you need to mark them in some special way. Elf/gcc ** (versions 3.2 and later) mark them as "hidden" to optimize access ** when Lua is compiled as a shared library. Not all elf targets support ** this attribute. Unfortunately, gcc does not offer a way to check ** whether the target offers that support, and those without support ** give a warning about it. To avoid these warnings, change to the ** default definition.
60261. ** In Windows, any exclamation mark (!) in the path is replaced by the ** path of the directory of the executable file of the current process.

60262. ** largest types available for C89 ('long' and 'double')
60263. ** ===== ** Compatibility with previous versions **
=====

60264. otherwise use 'long'
60265. **comment:** needs an extra library: -ldl
 label: code-design
60266. }{ double
60267. ** ===== ** Marks for exported symbols in the C code **
=====

60268. avoid undefined shifts
60269. ** use LUA_UACINT here to avoid problems with promotions (which ** can turn a comparison between unsigneds into a signed comparison)
60270. #define LUA_NOCVTS2N
60271. }{
60272. use presence of macro LLONG_MAX as proxy for C99 compliance
60273. use ISO C99 stuff
60274. }{ long long
60275. @@ LUA_COMPAT_MAXN defines the function 'maxn' in the table library.
60276. ** ===== ** System Configuration: macros to adapt (if needed) Lua to some ** particular platform, for instance compiling it with 32-bit numbers or ** restricting it to C89. **
=====

60277. ** \$Id: luacnf.h,v 1.255 2016/05/01 20:06:09 roberto Exp \$ ** Configuration file for Lua ** See Copyright Notice in lua.h
60278. @@ LUA_USE_APICHECK turns on several consistency checks on the C API. ** Define it as a help when debugging C code.
60279. Incompatibilities from 5.2 -> 5.3
60280. @@ LUA_USE_C89 controls the use of non-ISO-C89 features. ** Define it if you want Lua to avoid the use of a few C99 features ** or Windows-specific features on Windows.
60281. @@ LUA_C89_NUMBERS ensures that Lua uses the largest types available for ** C89 ('long' and 'double'); Windows always has ' __int64', so it does ** not need to use this case.
60282. #define LUA_USE_C89
60283. more often than not the libs go together with the core
60284. @@ LUA_PATH_DEFAULT is the default path that Lua uses to look for ** Lua libraries. @@ LUA_CPATH_DEFAULT is the default path that Lua uses to look for ** C libraries. ** CHANGE them if your machine has a non-conventional directory ** hierarchy or if you want to install your libraries in ** non-conventional directries.
60285. @@ lua_getlocaledecpoint gets the locale "radix character" (decimal point). ** Change that if you do not want to use C locales. (Code using this ** macro must include header 'locale.h'.)
60286. @@ LUA_COMPAT_APIINTCASTS controls the presence of macros for ** manipulating other integer types (lua_pushunsigned, lua_tounsigned, ** luaL_checkint, luaL_checklong, etc.)
60287. now the variable definitions
60288. @@ LUA_INTEGER is the integer type used by Lua. ** @@ LUA_UNSIGNED is the unsigned version of LUA_INTEGER. ** @@ LUA_UACINT is the result of an 'usual argument conversion' @@ over a LUA_INTEGER. @@ LUA_INTEGER_FRMLEN is the length modifier for reading/writing integers. @@ LUA_INTEGER_FMT is the format for writing integers. @@ LUA_MAXINTEGER is the maximum value for a LUA_INTEGER. @@ LUA_MININTEGER is the minimum value for a LUA_INTEGER. @@ lua_integer2str converts an integer to a string.
60289. predefined options for LUA_FLOAT_TYPE
60290. @@ macro 'lua_cpcall' emulates deprecated function lua_cpcall. ** You can call your C function directly (with light C functions).
60291. variants not available
60292. ** Local configuration. You can use this space to add your redefinitions ** without modifying the main part of the file.
60293. @@ lua_number2strx converts a float to an hexadecimal numeric string. ** In C99, 'sprintf' (with format specifiers "%a"/"%A") does that. ** Otherwise, you can leave 'lua_number2strx' undefined and Lua will ** provide its own implementation.
60294. **comment:** @@ The following macros supply trivial compatibility for some ** changes in the API. The macros themselves document how to ** change your code to avoid using them.
 label: documentation
60295. @@ LUA_EXTRASPACE defines the size of a raw memory area associated with ** a Lua state with very fast access. ** CHANGE it if you need a different size.
60296. @@ LUA_DIRSEP is the directory separator (for submodules). ** CHANGE it if your machine does not use "/" as the directory separator ** and is not Windows. (On Windows Lua automatically uses "\").
60297. @@ LUA_NUMBER is the floating-point type used by Lua. @@ LUA_UACNUMBER is the result of an 'usual argument conversion' @@ over a floating number. @@ l_mathlim(x) corrects limit name 'x' to the proper float type ** by prefixing it with one of FLT/DBL/LDBL. @@ LUA_NUMBER_FRMLEN is the length modifier for writing floats. @@ LUA_NUMBER_FMT is the format for writing floats. @@ lua_number2str converts a float to a string. @@ l_mathop allows the addition of an 'l' or 'f' to all math operations. @@ l_floor takes the floor of a float. @@ lua_str2number converts a decimal numeric string to a number.
60298. @@ LUA_COMPAT_MATHLIB controls the presence of several deprecated ** functions in the mathematical library.
60299. ** default configuration for 64-bit Lua ('long long' and 'double')
60300. @@ LUA_BITSINT defines the (minimum) number of bits in an 'int'.
60301. enable support for DLL
60302. enable goodies for regular Windows
60303. @@ LUA_KCONTEXT is the type of the context ('ctx') for continuation ** functions. It must be a numerical type; Lua will use 'intptr_t' if ** available, otherwise it will use 'ptrdiff_t' (the nearest thing to ** 'intptr_t' in C89)
60304. { int
60305. ** 32-bit integers and 'float'
60306. #define LUA_32BITS
60307. in Windows, can use specific Windows types
60308. **comment:** needs an extra library: -lreadline
 label: code-design
60309. @@ LUA_IDSIZE gives the maximum size for the description of the source @@ of a function in debug information. ** CHANGE it if you want a different size.
60310. even in C99 this type is optional
60311. @@ LUA_COMPAT_LOADERS controls the presence of table 'package.loaders'. ** You can replace it with 'package.searchers'.
60312. The following definitions are good for most cases here
60313. @@ LUA_QL describes how error messages quote program elements. ** Lua does not use these macros anymore; they are here for ** compatibility only.
60314. ** 'strtod' and 'opf' variants for math functions are not valid in ** C89. Otherwise, the macro 'HUGE_VALF' is a good proxy for testing the ** availability of these variants. ('math.h' is already included in ** all files that use these macros.)
60315. ** ===== ** Dependencies with C99 and other C details **
=====

60316. @@ LUA_COMPAT_5_2 controls other macros for compatibility with Lua 5.2. @@ LUA_COMPAT_5_1 controls other macros for compatibility with Lua 5.1.
 ** You can define it to get all options, or change specific options ** to fit your specific needs.
60317. @@ LUA_INT_TYPE defines the type for Lua integers. @@ LUA_FLOAT_TYPE defines the type for Lua floats. ** Lua should work fine with any mix of these options (if supported ** by your C compiler). The usual configurations are 64-bit integers ** and 'double' (the default), 32-bit integers and 'float' (for ** restricted platforms), and 'long/'double' (for C compilers not ** compliant with C99, which may not have support for 'long long').
60318. }
60319. @@ lua_str2number converts an hexadecimal numeric string to a number. ** In C99, 'strtod' does that conversion. Otherwise, you can ** leave 'lua_str2number' undefined and Lua will provide its own ** implementation.
60320. @@ LUA_COMPAT_LOG10 defines the function 'log10' in the math library. ** You can rewrite 'log10(x)' as 'log(x, 10)'.

60321. use 'int' if big enough
60322. == Configuration for Paths. ==
60323. == Macros that affect the API and must be stable (that is, must be the same when you compile Lua and when you compile code that links to Lua). You probably do not want/need to change them. ==
60324. predefined options for LUA_INT_TYPE
60325. @@ LUA_COMPAT_BITLIB controls the presence of library 'bit32'.
60326. 'int' always must have at least 16 bits
60327. @@ LUA_COMPAT_UNPACK controls the presence of global 'unpack'. ** You can replace it with 'table.unpack'.
60328. {} long
60329. #define LUA_NOCVTN2S
60330. { single float
60331. **comment:** needs some extra libraries
 label: code-design
60332. ** By default, Lua on Windows use (some) specific Windows features
60333. **comment:** @@ LUA_COMPAT_FLOATSTRING makes Lua format integral floats without a @@ a float mark (.0). ** This macro is not on by default even in compatibility mode, ** because this is not really an incompatibility.
 label: code-design
60334. empty
60335. == Language Variations ==
60336. @@ lua_numbertointeger converts a float number to an integer, or ** returns 0 if float is not within the range of a lua_Integer. ** (The range comparisons are tricky because of rounding. The tests ** here assume a two-complement representation, where MININTEGER always ** has an exact representation as a float; MAXINTEGER may not have one, ** and therefore its conversion to float may have an ill-defined value.)
60337. @@ LUA_32BITS enables Lua with 32-bit integers and 32-bit floats. You ** can also define LUA_32BITS in the make file, but changing here you ** ensure that all software connected to Lua will be compiled with the ** same configuration.
60338. broadly, Windows is C89
60339. **comment:** @@ LUAL_BUFFERSIZE is the buffer size used by the lauxlib buffer system. ** CHANGE it if it uses too much C-stack space. (For long double, ** 'string.format("%#.99f", 1e4932)' needs ~5030 bytes, so a ** smaller buffer would force a memory allocation for each call to ** 'string.format'.)
 label: code-design
60340. open all previous libraries
60341. ** \$Id: lualib.h,v 1.44 2014/02/06 17:32:33 roberto Exp \$ ** Lua standard libraries ** See Copyright Notice in lua.h
60342. reuse parent's source
60343. 1st char already checked
60344. ** \$Id: lundump.c,v 2.44 2015/11/02 16:09:30 roberto Exp \$ ** load precompiled Lua chunks ** See Copyright Notice in lua.h
60345. empty
60346. ** All high-level loads go through LoadVector; you can change it to ** adapt to the endianness of the input
60347. long string
60348. ** load precompiled chunk
60349. short string?
60350. load directly in final place
60351. no source in dump?
60352. larger than both
60353. dump one chunk; from ldump.c
60354. ** \$Id: lundump.h,v 1.45 2015/09/08 15:41:05 roberto Exp \$ ** load precompiled Lua chunks ** See Copyright Notice in lua.h
60355. data to catch conversion errors
60356. load one chunk; from lundump.c
60357. this is the official format
60358. ... and current position
60359. return nil ...
60360. do not move for 1st character
60361. no such character
60362. read next byte
60363. final result
60364. ** utf8len(s [, i [, j]]) --> number of characters that start in the ** range [i,j], or nil + current position if 's' is not well formed in ** that interval
60365. find beginning of next character
60366. start from here
60367. no more codepoints
60368. translate a relative string position: negative means back from end
60369. ascii?
60370. ** uchar(n1, n2, ...) -> char(n1)..char(n2)...
60371. to count number of continuation bytes
60372. did it find given character?
60373. add first byte
60374. from strlib
60375. to test next bit
60376. +1 to include first byte
60377. not a continuation byte?
60378. empty interval; return no values
60379. skip continuation bytes read
60380. ** \$Id: lutf8lib.c,v 1.15 2015/03/28 19:16:55 roberto Exp \$ ** Standard library for UTF-8 manipulation ** See Copyright Notice in lua.h
60381. (cannot pass final '\0')
60382. number of arguments
60383. pattern to match a single UTF-8 character
60384. invalid byte sequence
60385. placeholders
60386. ** offset(s, n, [i]) -> index where n-th character counting from ** position 'i' starts; 0 means character at 'i'.
60387. find beginning of previous character
60388. first iteration?
60389. (lua_Integer -> int) overflow?
60390. add lower 6 bits from cont. byte
60391. ** codepoint(s, [i, j]) -> returns codepoints for all characters ** that start in the range [i,j]
60392. and its continuations
60393. conversion error?
60394. move back
60395. find beginning of current byte sequence
60396. still have continuation bytes?

60397. ** Decode one UTF-8 sequence, returning NULL if byte sequence is invalid.
60398. optimize common case of single char
60399. skip current byte
60400. restore top
60401. save 't' table
60402. try 'le'
60403. true must be 1 !!
60404. remove new frame
60405. 'rs' is finished?
60406. not ($r < l$) ?
60407. clear mark
60408. cache will not break GC invariant?
60409. get metamethod
60410. restart luaV_execute over new Lua function
60411. **comment:** 'ls' is smaller than 'rs' ('rs' is not finished)
 label: requirement
60412. update internal index...
60413. all values are integer
60414. result is nil
60415. floating loop
60416. }=====

60417. ** create a new Lua closure, push it in the stack, and initialize ** its upvalues. Note that the closure is not cached if prototype is ** already black (which means that 'cache' was already cleared by the ** GC).
60418. call TM
60419. long string; copy strings directly to final result
60420. not an integral value?
60421. ** Main operation for concatenation: concat 'total' values in the stack, ** from 'L->top - total' up to 'L->top - 1'.
60422. ** \$Id: lvm.c,v 2.268 2016/02/05 19:59:14 roberto Exp \$ ** Lua virtual machine ** See Copyright Notice in lua.h
60423. ** finish execution of an opcode interrupted by an yield
60424. ** Try to convert a 'for' limit to an integer, preserving the ** semantics of the loop. ** (The following explanation assumes a non-negative step; it is valid ** for negative steps mutatis mutandis.) ** If the limit can be converted to an integer, rounding down, that is ** it. ** Otherwise, check whether the limit can be converted to a number. If ** the number is too large, it is OK to set the limit as LUA_MAXINTEGER, ** which means no limit. If the number is too negative, the loop ** should not run, because any initial integer value is larger than the ** limit. So, it sets the limit to LUA_MININTEGER. 'stopnow' corrects ** the extreme case when the initial value is LUA_MININTEGER, in which ** case the LUA_MININTEGER limit would still run the loop once.
60425. will try TM
60426. repeat until only 1 result left
60427. ** Integer division; return ' $m // n$ ', that is, $\text{floor}(m/n)$. ** C division truncates its result (rounds towards zero). ** ' $\text{floor}(q) == \text{trunc}(q)$ ' when ' $q \geq 0$ ' or when ' q ' is integer, ** otherwise ' $\text{floor}(q) == \text{trunc}(q) - 1$ '.
60428. complete required results with nil
60429. last stack slot filled by 'precall'
60430. key must be a string
60431. anchor new closure in stack
60432. 'l' is int and 'r' is float
60433. number of bits in an integer
60434. set its new value
60435. else primitive len
60436. increment index
60437. else loop
60438. fast track?
60439. '__newindex' metamethod
60440. create one
60441. both are float
60442. copy strings in stack from top - n up to top - 1 to buffer
60443. metamethod? break switch to call it
60444. local copy of function's base
60445. number of bits in the mantissa of a float
60446. -minint == maxint + 1
60447. ** Main operation 'ra' = '#rb'.
60448. length of string being copied
60449. invocation via reentry: continue execution
60450. ** Check whether integer 'i' is less than float 'f'. If 'i' has an ** exact representation as a float ('l_intfits'), compare numbers as ** floats. Otherwise, if 'f' is outside the range for integers, result ** is trivial. Otherwise, compare them as integers. (When 'i' has no ** float representation, either 'f' is "far away" from 'i' or 'f' has ** no precision left for a fractional part; either way, how 'f' is ** truncated is irrelevant.) When 'f' is NaN, comparisons must result ** in false.
60451. 'l' must be float
60452. no previous entry?
60453. 't' is a table
60454. without NaN, $(l \leq r) \iff \text{not}(r < l)$
60455. convert result from 'luaO_str2num' to an integer
60456. top when 'luaT_trybinTM' was called
60457. first element to concatenate
60458. try making all values floats
60459. preallocate it at once
60460. 'm/n' would be negative non-integer?
60461. not fit in integer?
60462. first operand is an empty string?
60463. float is smaller than min integer
60464. ** Try to convert a value to a float. The float case is already handled ** by the macro 'tonumber'.
60465. cached closure
60466. ** Finish the table access 'val = t[key]'. ** if 'slot' is NULL, 't' is not a table; otherwise, 'slot' points to ** t[k] entry (which must be nil).
60467. correct result for different rounding
60468. ** Check whether some integers may not fit in a float, that is, whether ** (maxinteger >> NBM) > 0 (that implies $(1 \ll NBM) \leq \text{maxinteger}$). ** (The shifts are done in parts to avoid shifting by more than the size ** of an integer. In a worst case, NBM == 113 for long double and ** sizeof(integer) == 32.)
60469. else will try the metamethod
60470. ** Compare two strings 'ls' x 'rs', returning an integer smaller-equal- ** -larger than zero if 'ls' is smaller-equal-larger than 'rs'. ** The code is a little tricky because it allows '0' in the strings ** and it uses 'strcoll' (to respect locales) for each segments ** of the strings.
60471. **comment:** ** check whether cached closure in prototype 'p' may be reused, that is, ** whether there is a cached closure with the same upvalues needed by ** new closure to be created.
 label: code-design

60472. perform C division
60473. conversion failed
60474. no metamethod
60475. is there a cached closure?
60476. second operand is empty?
60477. fill in its upvalues
60478. correct top (in case of previous open call)
60479. no vararg arguments
60480. **'l_intfits' checks whether a given integer can be converted to a ** float without rounding. Used in comparisons. Left undefined if ** all integers fit in a float precisely.
60481. 't' is not a table?
60482. close all upvalues from previous call
60483. check 'ls'
60484. reentry point when frame changes (call/return)
60485. top is one after last element (at top-2)
60486. ** Main operation for equality of Lua values; return 't1 == t2'. ** L == NULL means raw equality (no metamethods)
60487. values have same type and same variant
60488. not a number
60489. adjust results
60490. **comment:** counter to avoid infinite loops
 label: code-design
60491. result is first operand
60492. ** some macros for common tasks in 'luaV_execute'
60493. called frame
60494. **comment:** needs more space?
 label: code-design
60495. local reference to function's closure
60496. float division (always with floats)
60497. both strings longer than 'len'; go on comparing after the '\0'
60498. continue loop?
60499. are there elements to concat?
60500. Lua function
60501. 'r' is int and 'l' is float
60502. try the metamethod
60503. limit for table tag-method chains (to avoid loops)
60504. ** Main operation less than; return 'l < r'.
60505. index of first '\0' in both strings
60506. popped 'n' strings and pushed one
60507. no match?
60508. upvalue refers to local variable?
60509. minint <= f <= maxint ?
60510. f <= minint <= i (or 'f' is NaN) --> not(i < f)
60511. correct top
60512. mark the end of concat operands
60513. cannot convert to float?
60514. push cashed closure
60515. **comment:** wrong upvalue; cannot reuse closure
 label: code-design
60516. without NaN, (l < r) <--> not(r <= l)
60517. is metamethod a function?
60518. negate result
60519. next assignment may change this value
60520. ** Return 'l <= r', for numbers.
60521. no metamethod and (now) there is an entry with given key
60522. NaN <= i is always false
60523. needs ceil?
60524. two numbers with different variants
60525. is result a short string?
60526. nresults >= 0?
60527. move new frame into old one
60528. ** Integer modulus; return 'm % n'. (Assume that C '%' with ** negative operands follows C99 behavior. See previous comment ** about luaV_div.)
60529. ** try to convert a value to an integer, rounding according to 'mode': ** mode == 0: accepts only integral values ** mode == 1: takes the floor of the number ** mode == 2: takes the ceil of the number
60530. ** Check whether integer 'i' is less than or equal to float 'f'. ** See comments on previous function.
60531. function was tail called
60532. func. + 2 args (state and index)
60533. local 'ci' still from callee
60534. create a new one
60535. ** Shift left operation. (Shift right just negates 'y').
60536. jump back
60537. convert result of 'luaO_str2num' to a float
60538. if true, float is larger than max integer
60539. caller frame
60540. integer loop?
60541. inc. index
60542. n== -1; avoid overflow with 0x80000.../-1
60543. convert floor to ceil (remember: n != f)
60544. for test instructions, execute the jump instruction that follows it
60545. WARNING: any stack reallocation invalidates 'ra'
60546. compare them as floats
60547. not (r <= l) ?
60548. ...and external index
60549. is 't' a table?
60550. **comment:** less arguments than parameters?
 label: code-design
60551. string convertible to number?
60552. B == 0?
60553. try metamethod

60554. **comment:** macro used by 'luaV_concat' to ensure that element at 'o' is a string
label: code-design
60555. new closure is white, so we do not need a barrier here
60556. both are strings?
60557. not equal?
60558. both operands are numbers?
60559. fails if mode demands integral value
60560. copy strings to buffer
60561. table's metamethod
60562. minint < f <= maxint ?
60563. f >= maxint + 1 > i
60564. 'm/n' would be non-integer negative?
60565. both are integers
60566. put TM result in proper position
60567. done
60568. not the same variant?
60569. no TM?
60570. local reference to function's constant table
60571. call base
60572. try 'lt':
60573. go to next instruction
60574. finish its execution
60575. call it
60576. result is negated
60577. size already copied
60578. result is second op.
60579. get all var. arguments
60580. at least two non-empty string values; get as many as possible
60581. error
60582. tail call: put called frame (n) in place of caller one (o)
60583. else repeat (tail call 'luaV_finishget')
60584. concat them (may yield again)
60585. special cases: -1 or 0
60586. ** Return 'l < r', for numbers.
60587. skip next instruction (if C)
60588. shift right?
60589. called function
60590. not a table; check metamethod
60591. metamethod
60592. mark it is doing 'lt' for 'le'
60593. else try to access 'tm[key]'
60594. execute a jump instruction
60595. yet to concatenate
60596. NaN < i is always false
60597. objects are different
60598. "<=" using "<" instead?
60599. else repeat assignment over 'tm'
60600. save control variable
60601. no metamethod?
60602. required results
60603. got 'n' strings to create 1 new
60604. fetch an instruction and prepare its execution
60605. main loop of interpreter
60606. update 'base'
60607. caller function
60608. for each segment
60609. shift left
60610. check whether it has right upvalues
60611. 'luaV_concat' may invoke TMs and move the stack
60612. try to convert to float
60613. interrupted instruction
60614. strings are equal up to a '\0'
60615. C function?
60616. m % -1 == 0; avoid overflow with 0x80000...%-1
60617. correct base
60618. get upvalue from enclosing function
60619. collect total length and number of strings
60620. return cached closure (or NULL if no cached closure)
60621. save it on cache for reuse
60622. skip jump instruction
60623. limit of live values
60624. fresh invocation of 'luaV_execute'
60625. usually, let loops run
60626. 'ls' is finished?
60627. same for 'luaV_settable'
60628. f < minint <= i (or 'f' is NaN) --> not(i <= f)
60629. else previous instruction set top
60630. ** Finish a table assignment 't[key] = val'. ** If 'slot' is NULL, 't' is not a table. Otherwise, 'slot' points ** to the entry 't[key]', or to 'luaO_nilobject' if there is no such ** entry. (The value at 'slot' must be nil, otherwise 'luaV_fastset' ** would have done the job.)
60631. compare them as integers
60632. condition failed?
60633. l < r ?
60634. l <= r ?
60635. ** copy of 'luaV_gettable', but protecting the call to potential ** metamethod (which can reallocate the stack)
60636. ** {===== Function 'luaV_execute': main interpreter loop **
=====}
60637. external invocation: return
60638. number of elements handled in this pass (at least 2)

60639. old value must be nil
60640. previous call may change the stack
60641. create result
60642. only numbers can be equal with different variants
60643. ** Main operation less than or equal to; return 'l <= r'. If it needs ** a metamethod and there is no ' __le ', try ' __lt ', based on ** l <= r iff !(r < l) (assuming a total order). If the metamethod ** yields during this substitution, the continuation has to know ** about it (to negate the result of r < l); bit CIST_LEQ in the call ** status keeps that information.
60644. floor division
60645. move final result to final position
60646. ** fast track for 'gettable': if 't' is a table and 't[k]' is not nil, ** return 1 with 'slot' pointing to 't[k]' (final result). Otherwise, ** return 0 (meaning it will have to check metamethod) with 'slot' ** pointing to a nil 't[k]' (if 't' is a table) or NULL (otherwise). ** 'f' is the raw get function to use.
60647. ** You can define LUA_FLOORNI if you want to convert floats to integers ** by flooring them (instead of raising an error if they are not ** integral values)
60648. no conversion from numbers to strings
60649. not a table; 'slot' is NULL and result is 0
60650. ** \$Id: lvm.h,v 2.40 2016/01/05 16:07:21 roberto Exp \$ ** Lua virtual machine ** See Copyright Notice in lua.h
60651. ** Fast track for set table. If 't' is a table and 't[k]' is not nil, ** call GC barrier, do a raw 't[k]=v', and return true; otherwise, ** return false with 'slot' equal to NULL (if 't' is not a table) or ** 'nil'. (This is needed by 'luaV_finishget'.) Note that, if the macro ** returns true, there is no need to 'invalidateTMcache', because the ** call is not creating a new entry.
60652. ** standard implementation for 'gettable'
60653. else, do raw access
60654. result not nil?
60655. no conversion from strings to numbers
60656. try to read more
60657. no bytes in buffer?
60658. discount char being returned
60659. luaZ_fill consumed first byte; put it back
60660. no more input; return number of missing bytes
60661. min. between n and z->n
60662. ----- read ---
60663. ** \$Id: lzio.c,v 1.37 2015/09/08 15:41:05 roberto Exp \$ ** Buffered streams ** See Copyright Notice in lua.h
60664. end of stream
60665. current position in buffer
60666. Lua state (for reader)
60667. ** \$Id: lzio.h,v 1.31 2015/09/08 15:41:05 roberto Exp \$ ** Buffered streams ** See Copyright Notice in lua.h
60668. ----- Private Part -----
60669. reader function
60670. additional data
60671. bytes still unread
60672. read next n bytes
60673. Append the N-byte string in zIn to the end of the JsonString string ** under construction. Enclose the string in "..." and escape ** any double-quotes or backslash characters contained within the ** string.
60674. ** A NameContext defines a context in which to resolve table and column ** names. The context consists of a list of tables (the pSrcList) field and ** a list of named expression (pEList). The named expression list may ** be NULL. The pSrc corresponds to the FROM clause of a SELECT or ** to the table being operated on by INSERT, UPDATE, or DELETE. The ** pEList corresponds to the result set of a SELECT and is NULL for ** other statements. ** ** NameContexts can be nested. When resolving names, the inner-most ** context is searched first. If no match is found, the next outer ** context is checked. If there is still no match, the next context ** is checked. This process continues until either a match is found ** or all contexts are checked. When a match is found, the nRef member of ** the context containing the match is incremented. ** ** Each subquery gets a new NameContext. The pNext field points to the ** NameContext in the parent query. Thus the process of scanning the ** NameContext list corresponds to searching through successively outer ** subqueries looking for a match.
60675. The new root-page may not be allocated on a pointer-map page, or the ** PENDING_BYT page.
60676. ** Search the node hash table for node iNode. If found, return a pointer ** to it. Otherwise, return 0.
60677. ** This routine is called after a single SQL statement has been ** parsed and a VDBE program to execute that statement has been ** prepared. This routine puts the finishing touches on the ** VDBE program and resets the pParse structure for the next ** parse. ** ** Note that if an error occurred, it might be the case that ** no VDBE code was generated.
60678. Collating sequence must be the same on all columns
60679. PRAGMA cache_spill=ON
60680. Size of array apSegment
60681. ** The following are copied from sqliteInt.h. ** ** Constants for the largest and smallest possible 64-bit signed integers. ** These macros are designed to work correctly on both 32-bit and 64-bit ** compilers.
60682. Next method for this node.
60683. ** This function is a no-op if the pager is in exclusive mode and not ** in the ERROR state. Otherwise, it switches the pager to PAGER_OPEN ** state. ** ** If the pager is not in exclusive-access mode, the database file is ** completely unlocked. If the file is unlocked and the file-system does ** not exhibit the UNDELETEABLE_WHEN_OPEN property, the journal file is ** closed (if it is open). ** ** If the pager is in ERROR state when this function is called, the ** contents of the pager cache are discarded before switching back to ** the OPEN state. Regardless of whether the pager is in exclusive-mode ** or not, any journal file left in the file-system will be treated ** as a hot-journal and rolled back the next time a read-transaction ** is opened (by this or by any other connection).
60684. If there were FTS5_MIN_DLIDX_SIZE or more empty leaf pages written ** to the database, also write the doclist-index to disk.
60685. ** The winMemData structure stores information required by the Win32-specific ** sqlite3_mem_methods implementation.
60686. **comment:** Database connection for reporting malloc problems
 label: code-design
60687. Construct the in-memory representation schema tables (sqlite_master or ** sqlite_temp_master) by invoking the parser directly. The appropriate ** table name will be inserted automatically by the parser so we can just ** use the abbreviation "x" here. The parser will also automatically tag ** the schema table as read-only.
60688. ** Advance to the next WhereTerm that matches according to the criteria ** established when the pScan object was initialized by whereScanInit(). ** Return NULL if there are no more matching WhereTerms.
60689. eMode is always INCRINIT_NORMAL in single-threaded mode
60690. Bitmask of all indexable tables in the clause
60691. Contains aggregate functions
60692. OUT: Set for new term
60693. **comment:** ** Each entry in a RowSet is an instance of the following object. ** ** This same object is reused to store a linked list of trees of RowSetEntry ** objects. In that alternative use, pRight points to the next entry ** in the list, pLeft points to the tree, and v is unused. The ** RowSet.pForest value points to the head of this forest list.
 label: code-design
60694. If the either the page-size or sector-size in the journal-header is ** invalid, then the process that wrote the journal-header must have ** crashed before the header was synced. In this case stop reading ** the journal file here.
60695. Source-code line that generated this opcode
60696. Invoke the virtual table constructor
60697. ** An instance of the TreeView object is used for printing the content of ** data structures on sqlite3DebugPrintf() using a tree-like view.
60698. ** Internal function prototypes
60699. ** This #if does not rely on the SQLITE_OS_WINCE define because the ** corresponding section in "date.c" cannot use it.
60700. 95

60701. ***** End of stmt.c *****

60702. ** This routine is a helper for explainIndexRange() below *** pStr holds the text of an expression that we are building up one term ** at a time. This routine adds a new term to the end of the expression. ** Terms are separated by AND so add the "AND" text for second and subsequent ** terms only.

60703. Name of the database. Might be NULL

60704. INTERSECT is different from the others since it requires ** two temporary tables. Hence it has its own case. Begin ** by allocating the tables we will need.

60705. **comment:** Opcode: Insert P1 P2 P3 P4 P5 ** Synopsis: intkey=r[P3] data=r[P2] *** Write an entry into the table of cursor P1. A new entry is ** created if it doesn't already exist or the data for an existing ** entry is overwritten. The data is the value MEM_Blob stored in register ** number P2. The key is stored in register P3. The key must ** be a MEM_Int. *** If the OPFLAG_NCHANGE flag of P5 is set, then the row change count is ** incremented (otherwise not). If the OPFLAG_LASTROWID flag of P5 is set, ** then rowid is stored for subsequent return by the ** sqlite3_last_insert_rowid() function (otherwise it is unmodified). *** If the OPFLAG_USESEEKRESULT flag of P5 is set, the implementation might ** run faster by avoiding an unnecessary seek on cursor P1. However, ** the OPFLAG_USESEEKRESULT flag must only be set if there have been no prior ** seeks on the cursor or if the most recent seek used a key equal to P3. *** If the OPFLAG_ISUPDATE flag is set, then this opcode is part of an ** UPDATE operation. Otherwise (if the flag is clear) then this opcode ** is part of an INSERT operation. The difference is only important to ** the update hook. *** Parameter P4 may point to a Table structure, or may be NULL. If it is ** not NULL, then the update-hook (sqlite3_xUpdateCallback) is invoked ** following a successful insert. *** (WARNING/TODO: If P1 is a pseudo-cursor and P2 is dynamically ** allocated, then ownership of P2 is transferred to the pseudo-cursor ** and register P2 becomes ephemeral. If the cursor is changed, the ** value of register P2 will then change. Make sure this does not ** cause any problems.) *** This instruction only works on tables. The equivalent instruction ** for indices is OP_IdxInsert.

label: code-design

60706. Incremented when %_config is modified

60707. ** NOTE: On some sub-platforms, the InterlockedCompareExchange "function" ** is really just a macro that uses a compiler intrinsic (e.g. x64). ** So do not try to make this into a redefinable interface.

60708. Prefix parameter value (or NULL)

60709. ** Free all memory associated with the Vdbe passed as the second argument, ** except for object itself, which is preserved. *** The difference between this function and sqlite3VdbeDelete() is that ** VdbeDelete() also unlinks the Vdbe from the list of VMs associated with ** the database connection and frees the object itself.

60710. Array of PmaReaders to merge data from

60711. Block id for new block

60712. Definitions of all built-in pragmas

60713. **comment:** ** CAPI3REF: Name Of The Folder Holding Temporary Files *** ^{If this global variable is made to point to a string which is ** the name of a folder (a.k.a. directory), then all temporary files ** created by SQLite when using a built-in [sqlite3_vfs | VFS] ** will be placed in that directory.}^ If this variable ** is a NULL pointer, then SQLite performs a search for an appropriate ** temporary file directory. *** Applications are strongly discouraged from using this global variable. ** It is required to set a temporary folder on Windows Runtime (WinRT). ** But for all other platforms, it is highly recommended that applications ** neither read nor write this variable. This global variable is a relic ** that exists for backwards compatibility of legacy applications and should ** be avoided in new projects. *** It is not safe to read or modify this variable in more than one ** thread at a time. It is not safe to read or modify this variable ** if a [database connection] is being used at the same time in a separate ** thread. ** It is intended that this variable be set once ** as part of process initialization and before any SQLite interface ** routines have been called and that this variable remain unchanged ** thereafter. *** ^{The [temp_store_directory pragma] may modify this variable and cause ** it to point to memory obtained from [sqlite3_malloc].}^ Furthermore, ** the [temp_store_directory pragma] always assumes that any string ** that this variable points to is held in memory obtained from ** [sqlite3_malloc] and the pragma may attempt to free that memory ** using [sqlite3_free]. ** Hence, if this variable is modified directly, either it should be ** made NULL or made to point to memory obtained from [sqlite3_malloc] ** or else the use of the [temp_store_directory pragma] should be avoided. ** Except when requested by the [temp_store_directory pragma], SQLite ** does not free the memory that sqlite3_temp_directory points to. If ** the application wants that memory to be freed, it must do ** so itself, taking care to only do so after all [database connection] ** objects have been destroyed. *** ^{Note to Windows Runtime users:The temporary directory must be set ** prior to calling [sqlite3_open] or [sqlite3_open_v2]. Otherwise, various ** features that require the use of temporary files may fail. Here is an ** example of how to do this using C++ with the Windows Runtime:*** <blockquote><pre>** LPCWSTR zPath = Windows::Storage::ApplicationData::Current-> ** TemporaryFolder->Path->Data(); ** char zPathBuf#91;MAX_PATH + 1#93;; ** memset(zPathBuf, 0, sizeof(zPathBuf)); ** WideCharToMultiByte(CP_UTF8, 0, zPath, -1, zPathBuf, sizeof(zPathBuf), ** NULL, NULL); ** sqlite3_temp_directory = sqlite3_mprintf("%s", zPathBuf); ** </pre></blockquote>

label: code-design

60714. Optional list of view column names

60715. ** The code in this file is only used if we are compiling multithreaded ** on a Win32 system.

60716. ** This method is the destructor for stmt_cursor objects.

60717. 1430

60718. synopsis: intkey=r[P3] data=r[P2]

60719. ** Locate the in-memory structure that describes a particular database ** table given the name of that table and (optionally) the name of the ** database containing the table. Return NULL if not found. Also leave an ** error message in pParse->zErrMsg. *** The difference between this routine and sqlite3FindTable() is that this ** routine leaves an error message in pParse->zErrMsg where ** sqlite3FindTable() does not.

60720. Table used for PK definition

60721. True if the VM needs to be recompiled

60722. Heap storage space

60723. IN/OUT: Pointer to output buffer

60724. ** Invalidate the overflow page-list cache for all cursors opened ** on the shared btree structure pBt.

60725. **comment:** ** Make a copy of a string in memory obtained from sqliteMalloc(). These ** functions call sqlite3MallocRaw() directly instead of sqliteMalloc(). This ** is because when memory debugging is turned on, these two functions are ** called via macros that record the current file and line number in the ** ThreadData structure.

label: code-design

60726. Mode to pass to walCheckpoint()

60727. ***** Begin file fts3_tokenizer1.c *****

60728. One of the JSON_ type values

60729. For ONEPASS, no need to store the rowid/primary-key. There is only ** one, so just keep it in its register(s) and fall through to the ** delete code.

60730. The target name

60731. Not part of an OR optimization

60732. **comment:** Finally, jump back to the beginning of the executable code.

label: code-design

60733. ** Target size for allocation chunks.

60734. Index sort order

60735. SYNC_NORMAL or SYNC_FULL otherwise

60736. ** Translate a single byte of Hex into an integer. ** This routine only works if h really is a valid hexadecimal ** character: 0..9a..fA..F

60737. String or BLOB exceeds size limit

60738. Disable the API redefinition in sqlite3ext.h

60739. Convert the filename to the system encoding.

60740. 251

60741. If this is an INSERT on a view with an INSTEAD OF INSERT trigger, ** do not attempt any conversions before assembling the record. ** If this is a real table, attempt conversions as required by the ** table column affinities.

60742. Variable initialization

60743. (305) as ::= ID|STRING

60744. Move the cursor to the first entry in the table. Return SQLITE_OK ** on success. Set *pRes to 0 if the cursor actually points to something ** or set *pRes to 1 if the table is empty.

60745. case_operand ::= expr

60746. READONLY

60747. Index of new first cell on page
60748. Original value of iSize
60749. Mask of query phrases covered
60750. Get the field width
60751. Number of free bytes on the page
60752. Write the index of a new slot here
60753. True for a read-only page
60754. Opcode: Next P1 P2 P3 P4 P5 *** Advance cursor P1 so that it points to the next key/data pair in its ** table or index. If there are no more key/value pairs then fall through ** to the following instruction. But if the cursor advance was successful, ** jump immediately to P2. *** The Next opcode is only valid following an SeekGT, SeekGE, or ** OP_Rewind opcode used to position the cursor. Next is not allowed ** to follow SeekLT, SeekLE, or OP_Last. *** The P1 cursor must be for a real table, not a pseudo-table. P1 must have ** been opened prior to this opcode or the program will segfault. *** The P3 value is a hint to the btree implementation. If P3==1, that ** means P1 is an SQL index and that this instruction could have been ** omitted if that index had been unique. P3 is usually 0. P3 is ** always either 0 or 1. *** P4 is always of type P4_ADVANCE. The function pointer points to ** sqlite3BtreeNext(). *** If P5 is positive and the jump is taken, then event counter ** number P5-1 in the prepared statement is incremented. *** See also: Prev, NextIfOpen
60755. Return code from subfunctions
60756. ** Return true if the heap is currently under memory pressure - in other ** words if the amount of heap used is close to the limit set by ** sqlite3_soft_heap_limit().
60757. 252
60758. Number of opcodes for progress callback
60759. Report errors here, if not NULL
60760. Btree to change journal mode of
60761. Number of elements summed
60762. ***** End of parse.h *****
60763. Allocate space for the conch filename and initialize the name to ** the name of the original database file.
60764. pVbeBranchArg
60765. The text encoding used by this database
60766. Table to be opened
60767. Sector size doesn't matter for temporary files. Also, the file ** may not have been opened yet, in which case the OsSectorSize() ** call will segfault.
60768. Verify that the page list is in ascending order
60769. **comment:** Figure out where the doclist for this term ends
 label: code-design
60770. Compare pReadr1 and pReadr2. Store the result in variable iRes.
60771. If eStatementOp is non-zero, then a statement transaction needs to ** be committed or rolled back. Call sqlite3VdbeCloseStatement() to ** do so. If this operation returns an error, and the current statement ** error code is SQLITE_OK or SQLITE_CONSTRAINT, then promote the ** current statement error code.
60772. ** Added for 3.4.1
60773. ** A RowSet in an instance of the following structure. *** A typedef of this structure is found in sqliteInt.h.
60774. Reclaim memory used by a Stat4Sample
60775. INTO
60776. Keep advancing iterators until they all point to the same document
60777. **comment:** Temporary memory used by this routine
 label: code-design
60778. Result set of the query
60779. **comment:** If the db-size header field is incorrect (as it may be if an old ** client has been writing the database file), update it now. Doing ** this sooner rather than later means the database size can safely ** re-read the database size from page 1 if a savepoint or transaction ** rollback occurs within the transaction.
 label: code-design
60780. nCellKey will always be between 0 and 0xffffffff because of the way ** that btreeParseCellPtr() and sqlite3GetVarint32() are implemented
60781. #include "whereInt.h"
60782. ** Cause a function to throw an error if it was called from OP_PureFunc ** rather than OP_Function. *** OP_PureFunc means that the function must be deterministic, and should ** throw an error if it is given inputs that would make it non-deterministic. *** This routine is invoked by date/time functions that use non-deterministic ** features such as 'now'.
60783. Find the checksum values to use as input for the recalculating the ** first checksum. If the first frame is frame 1 (implying that the current ** transaction restarted the wal file), these values must be read from the ** wal-file header. Otherwise, read them from the frame header of the ** previous frame.
60784. If this is the first entry to be added to this hash-table, zero the ** entire hash table and aPgno[] array before proceeding.
60785. shared memory and mmap are enabled
60786. The where clause
60787. ** The xSavepoint() method. *** Flush the contents of the pending-terms table to disk.
60788. Flags for sub-WHERE clause
60789. RHS of the comparison
60790. True if RTree holding integer coordinates
60791. OUT: Byte offset of token in input buffer
60792. ** Values passed as the 5th argument to allocateBtreePage()
60793. Offset in aData[] of next change
60794. ** This is called as part of registering the FTS5 module with database ** connection db. It registers several user-defined scalar functions useful ** with FTS5. **
 ** If successful, SQLITE_OK is returned. If an error occurs, some other ** SQLite error code is returned instead.
60795. Try to keep nFreeSlot above this
60796. Freeblock off the end of the page
60797. # include "os_win.h"
60798. Special case: A DELETE without a WHERE clause deletes everything. ** It is easier just to erase the whole table. Prior to version 3.6.5, ** this optimization caused the row change count (the value returned by ** API function sqlite3_count_changes) to be set incorrectly. *** The "rcauth==SQLITE_OK" terms is the ** IMPLEMENTATION-OF: R-17228-37124 If the action code is SQLITE_DELETE and ** the callback returns SQLITE_IGNORE then the DELETE operation proceeds but ** the truncate optimization is disabled and all rows are deleted ** individually.
60799. ** An SQL user-function registered to do the work of an DETACH statement. The ** three arguments to the function come directly from a detach statement: ***
 DETACH DATABASE x *** SELECT sqlite_detach(x)
60800. First of nPk cells holding PRIMARY KEY value
60801. ** Handle the special case of a compound-select that originates from a ** VALUES clause. By handling this as a special case, we avoid deep ** recursion, and thus do not need to enforce the SQLITE_LIMIT_COMPOUND_SELECT ** on a VALUES clause. *** Because the Select object originates from a VALUES clause: ** (1) It has no LIMIT or OFFSET ** (2) All terms are UNION ALL ** (3) There is no ORDER BY clause
60802. child to parent column mapping
60803. ***** Continuing where we left off in pager.c *****
60804. ** Argument ppHead contains a pointer to the current head of a query ** expression tree being parsed. pPrev is the expression node most recently ** inserted into the tree. This function adds pNew, which is always a binary ** operator node, into the expression tree based on the relative precedence ** of pNew and the existing nodes of the tree. This may result in the head ** of the tree changing, in which case *ppHead is set to the new root node.
60805. 135
60806. The set of notindexed= columns
60807. Number of elements in list aMerge
60808. Indexed column is an expression
60809. Make pRoot, the root page of the b-tree, writable. Allocate a new ** page that will become the new right-child of pPage. Copy the contents ** of the node stored on pRoot into the new child page.

60810. same as TK_NE, synopsis: IF r[P3] != r[P1]
60811. Argument to parse
60812. True if %_docsizes table exists
60813. Methods for the module
60814. Row is being deleted from this table
60815. Used by this thread only
60816. Space for azColumn strings
60817. If this condition is true then the largest rowid for the current ** term may not be stored on the current page. So search forward to ** see where said rowid really is.
60818. ** Prepare the two insert statements - Fts5Storage.pInsertContent and ** Fts5Storage.pInsertDocsize - if they have not already been prepared. ** Return
 SQLITE_OK if successful, or an SQLite error code if an error ** occurs.
60819. Total pages involved in the balance
60820. 76
60821. varint containing suffix size
60822. Bit values for PgHdr.flags
60823. **comment:** ** CAPI3REF: Loadable Extension Thunk *** A pointer to the opaque sqlite3_api_routines structure is passed as ** the third parameter to entry
 points of [loadable extensions]. This ** structure must be typedefed in order to work around compiler warnings ** on some platforms.
 label: code-design
60824. ** Function prototypes
60825. Opcode: SeekGT P1 P2 P3 P4 * ** Synopsis: key=r[P3@P4] *** If cursor P1 refers to an SQL table (B-Tree that uses integer keys), ** use the value in register
 P3 as a key. If cursor P1 refers ** to an SQL index, then P3 is the first in an array of P4 registers ** that are used as an unpacked index key. *** Reposition
 cursor P1 so that it points to the smallest entry that ** is greater than the key value. If there are no records greater than ** the key and P2 is not zero, then jump to
 P2. *** This opcode leaves the cursor configured to move in forward order, ** from the beginning toward the end. In other words, the cursor is ** configured to
 use Next, not Prev. *** See also: Found, NotFound, SeekLt, SeekGe, SeekLe
60826. Schema required - "main" is default
60827. ** Clear the PGHDR_NEED_SYNC and PGHDR_WRITEABLE flag from all dirty pages.
60828. For looping over bytes of pCell
60829. SQLITE_STATUS_PAGECACHE_SIZE
60830. Linked list of VDBEs with the same Vdbe.db
60831. Size of zRep
60832. isCommit flag passed to sqlite3PagerMovepage
60833. First in array of registers
60834. Position of next token
60835. FOR
60836. **comment:** If the Select object is really just a simple VALUES() list with a ** single row (the common case) then keep that one row of values ** and discard the
 other (unused) parts of the pSelect object
 label: code-design
60837. Most jump operations do a goto to this spot in order to update ** the pOp pointer.
60838. If changing the rowid value, or if there are foreign key constraints ** to process, delete the old record. Otherwise, add a noop OP_Delete ** to invoke the pre-
 update hook. ** That (regNew==regnewRowid+1) is true is also important for the ** pre-update hook. If the caller invokes preupdate_new(), the returned **
 value is copied from memory cell (regNewRowid+1+iCol), where iCol ** is the column index supplied by the user.
60839. ** iAbsLevel is an absolute level that may be assumed to exist within ** the database. This function checks if it is the largest level number ** within its index.
 Assuming no error occurs, *pbMax is set to 1 if ** iAbsLevel is indeed the largest level, or 0 otherwise, and SQLITE_OK ** is returned. If an error occurs, an
 error code is returned and the ** final value of *pbMax is undefined.
60840. If pszEst is not NULL, store an estimate of the field size. The ** estimate is scaled so that the size of an integer is 1.
60841. SELECT to read row iDel from %_data
60842. Call xSync() on the wal file. This causes SQLite to sync the ** directory in which the target database and the wal file reside, in ** case it has not been synced since
 the rename() call in ** rmuMoveOalFile().
60843. If there is cached matchinfo() data, but the format string for the ** cache does not match the format string for this request, discard ** the cached data.
60844. Various elements of the SELECT copied into local variables for ** convenience
60845. The entire doclist will fit on the current leaf.
60846. Escape " characters
60847. ***** End of sqlite3.h *****
60848. Return code from sqlite3_create_collation_x()
60849. Authenticated as a normal user
60850. OUT: True if *might* be new term
60851. **comment:** Register used by each index. 0 for unused indices
 label: code-design
60852. Has a child
60853. This routine only runs while holding the checkpoint lock. And ** it only runs if there is actually content in the log (mxFrame>0).
60854. Root of NEAR/AND cluster
60855. PRIMARY KEY index, or NULL for rowid tables
60856. Append the PK part of the WHERE clause
60857. IMP: R-52756-41993 This function is a wrapper around the ** sqlite3_total_changes() C/C++ interface.
60858. ** Return the text of a syntax error message on a JSON path. Space is ** obtained from sqlite3_malloc().
60859. 45
60860. Acquire the mutex before continuing
60861. in1: P1 is an input
60862. Use write-ahead logging
60863. ** Recursively delete a Trigger structure
60864. Write a pointer to the page here
60865. ** Create/destroy an Fts5Index object.
60866. Total number of entries in hash table
60867. Interpret the [schema.] part of the pragma statement. iDb is the ** index of the database this pragma is being applied to in db.aDb[].
60868. ** If no value has been provided for SQLITE_MAX_WORKER_THREADS, or if ** SQLITE_TEMP_STORE is set to 3 (never use temporary files), set it ** to
 zero.
60869. SQLITE OMIT_DISKIO
60870. ** Position the pCsr->pStmt statement so that it is on the row ** of the %_content table that contains the last match. Return ** SQLITE_OK on success.
60871. Number of rows in the result
60872. ** This function is called on each phrase after the position lists for ** any deferred tokens have been loaded into memory. It updates the phrases ** current position
 list to include only those positions that are really ** instances of the phrase (after considering deferred tokens). If this ** means that the phrase does not appear in
 the current row, doclist.pList ** and doclist.nList are both zeroed. *** SQLITE_OK is returned if no error occurs, otherwise an SQLite error code.
60873. nbr of bytes to lock
60874. A column-list is terminated by either a 0x01 or 0x00.
60875. 10..17

60879. RTREE_COORD_REAL32 or RTREE_COORD_INT32
60880. Prev() is noop if negative. Next() is noop if positive. ** Error code if eState==CURSORFAULT
60881. The database containing the table to be locked
60882. IMMEDIATE
60883. ** Call sqlite3ExprAnalyzeAggregates() for every expression in an ** expression list. Return the number of errors. ** ** If an error is found, the analysis is cut short.
60884. An "... AND (...)" expression
60885. ** Return TRUE if the given SQL string ends in a semicolon. ** ** Special handling is required for CREATE TRIGGER statements. ** Whenever the CREATE TRIGGER keywords are seen, the statement ** must end with ";END;". ** ** This implementation uses a state machine with 8 states: ** ** (0) INVALID We have not yet seen a non-whitespace character. ** ** (1) START At the beginning or end of an SQL statement. This routine ** returns 1 if it ends in the START state and 0 if it ends ** in any other state. ** ** (2) NORMAL We are in the middle of a statement which ends with a single ** semicolon. ** ** (3) EXPLAIN The keyword EXPLAIN has been seen at the beginning of ** a statement. ** ** (4) CREATE The keyword CREATE has been seen at the beginning of a ** statement, possibly preceded by EXPLAIN and/or followed by ** TEMP or TEMPORARY ** ** (5) TRIGGER We are in the middle of a trigger definition that must be ** ended by a semicolon, the keyword END, and another semicolon. ** ** (6) SEMI We've seen the first semicolon in the ";END;" that occurs at ** the end of a trigger definition. ** ** (7) END We've seen the ";END" of the ";END;" that occurs at the end ** of a trigger definition. ** ** Transitions between states above are determined by tokens extracted ** from the input. The following tokens are significant: ** ** (0) tkSEMI A semicolon. ** (1) tkWS Whitespace. ** (2) tkOTHER Any other SQL token. ** (3) tkEXPLAIN The "explain" keyword. ** (4) tkCREATE The "create" keyword. ** (5) tkTEMP The "temp" or "temporary" keyword. ** (6) tkTRIGGER The "trigger" keyword. ** (7) tkEND The "end" keyword. ** ** Whitespace never causes a state transition and is always ignored. ** This means that a SQL string of all whitespace is invalid. ** ** If we compile with SQLITE OMIT_TRIGGER, all of the computation needed ** to recognize the end of a trigger can be omitted. All we have to do ** is look for a semicolon that is not part of an string or comment.

60886. PREVENTS-HARMLESS-OVERREAD
60887. The title text
60888. ** An instance of this object stores information about each a single database ** page that has been loaded into memory. The information in this object ** is derived from the raw on-disk page content. ** ** As each database page is loaded into memory, the pager allocates an ** instance of this object and zeros the first 8 bytes. (This is the ** "extra" information associated with each page of the pager.) ** ** Access to all fields of this structure is controlled by the mutex ** stored in MemPage.pBt->mutex.
60889. Used by: default_cache_size
60890. Restrict matches to these columns
60891. ** Fill the InitData structure with an error message that indicates ** that the database is corrupt.
60892. DEFERRED => ID
60893. _FTS3_HASH_H
60894. ** Implementation of the sqlite_source_id() function. The result is a string ** that identifies the particular version of the source code used to build ** SQLite.
60895. Delete any triggers created to implement actions for this FK.
60896. Invoke the callback once if the
60897. xDLError
60898. Allocate required output space
60899. Read in the first 8 bytes of the journal header. If they do not match ** the magic string found at the start of each journal header, return ** SQLITE_DONE. If an IO error occurs, return an error code. Otherwise, ** proceed.
60900. Connection to a write-ahead log (WAL) file. ** There is one object of this type for each pager.
60901. True if error has occurred, else false
60902. Last statement is orphaned TEMP trigger
60903. ** Return true if currently inside an sqlite3_declare_vtab() call.
60904. Disable mutex on connections
60905. **comment:** ** 2010 February 1 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the implementation of a write-ahead log (WAL) used in ** "journal_mode=WAL" mode. ** ** WRITE-AHEAD LOG (WAL) FILE FORMAT ** ** A WAL file consists of a header followed by zero or more "frames". ** Each frame records the revised content of a single page from the ** database file. All changes to the database are recorded by writing ** frames into the WAL. Transactions commit when a frame is written that ** contains a commit marker. A single WAL can and usually does record ** multiple transactions. Periodically, the content of the WAL is ** transferred back into the database file in an operation called a ** "checkpoint". ** ** A single WAL file can be used multiple times. In other words, the ** WAL can fill up with frames and then be checkpointed and then new ** frames can overwrite the old ones. A WAL always grows from beginning ** toward the end. Checksums and counters attached to each frame are ** used to determine which frames within the WAL are valid and which ** are leftovers from prior checkpoints. ** ** The WAL header is 32 bytes in size and consists of the following eight ** big-endian 32-bit unsigned integer values: ** ** 0: Magic number. 0x377f0682 or 0x377f0683 ** 4: File format version. Currently 3007000 ** 8: Database page size. Example: 1024 ** 12: Checkpoint sequence number ** 16: Salt-1, random integer incremented with each checkpoint ** 20: Salt-2, a different random integer changing with each ckpt ** 24: Checksum-1 (first part of checksum for first 24 bytes of header). ** 28: Checksum-2 (second part of checksum for first 24 bytes of header). ** ** Immediately following the wal-header are zero or more frames. Each ** frame consists of a 24-byte frame-header followed by a <page-size> bytes ** of page data. The frame-header is six big-endian 32-bit unsigned ** integer values, as follows: ** ** 0: Page number. ** 4: For commit records, the size of the database image in pages ** after the commit. For all other records, zero. ** 8: Salt-1 (copied from the header) ** 12: Salt-2 (copied from the header) ** 16: Checksum-1. ** 20: Checksum-2. ** ** A frame is considered valid if and only if the following conditions are ** true: ** ** (1) The salt-1 and salt-2 values in the frame-header match ** salt values in the wal-header ** ** (2) The checksum values in the final 8 bytes of the frame-header ** exactly match the checksum computed consecutively on the ** WAL header and the first 8 bytes and the content of all frames ** up to and including the current frame. ** ** The checksum is computed using 32-bit big-endian integers if the ** magic number in the first 4 bytes of the WAL is 0x377f0683 and it ** is computed using little-endian if the magic number is 0x377f0682. ** The checksum values are always stored in the frame header in a ** big-endian format regardless of which byte order is used to compute ** the checksum. The checksum is computed by interpreting the input as ** an even number of unsigned 32-bit integers: x[0] through x[N]. The ** algorithm used for the checksum is as follows: ** ** for i from 0 to n-1 step 2: ** s0 += x[i] + s1; ** s1 += x[i+1] + s0; ** endfor ** ** Note that s0 and s1 are both weighted checksums using fibonacci weights ** in reverse order (the largest fibonacci weight occurs on the first element ** of the sequence being summed.) The s1 value spans all 32-bit ** terms of the sequence whereas s0 omits the final term. ** ** On a checkpoint, the WAL is first VFS.xSync-ed, then valid content of the ** WAL is transferred into the database, then the database is VFS.xSync-ed. ** The VFS.xSync operations serve as write barriers - all writes launched ** before the xSync must complete before any write that launches after the ** xSync begins. ** ** After each checkpoint, the salt-1 value is incremented and the salt-2 ** value is randomized. This prevents old and new frames in the WAL from ** being considered valid at the same time and being checkpointing together ** following a crash. ** ** READER ALGORITHM ** ** To read a page from the database (call it page number P), a reader ** first checks the WAL to see if it contains page P. If so, then the ** last valid instance of page P that is followed by a commit frame ** or is a commit frame itself becomes the value read. If the WAL ** contains no copies of page P that are valid and which are a commit ** frame or are followed by a commit frame, then page P is read from ** the database file. ** ** To start a read transaction, the reader records the index of the last ** valid frame in the WAL. The reader uses this recorded "mxFrame" value ** for all subsequent read operations. New transactions can be appended ** to the WAL, but as long as the reader uses its original mxFrame value ** and ignores the newly appended content, it will see a consistent snapshot ** of the database from a single point in time. This technique allows ** multiple concurrent readers to view different versions of the database ** content simultaneously. ** ** The reader algorithm in the previous paragraphs works correctly, but ** because frames for page P can appear anywhere within the WAL, the ** reader has to scan the entire WAL looking for page P frames. If the ** WAL is large (multiple megabytes is typical) that scan can be slow, ** and read performance suffers. To overcome this problem, a separate ** data structure called the wal-index is maintained to expedite the ** search for frames of a particular page. ** ** WAL-INDEX FORMAT ** ** Conceptually, the wal-index is shared memory, though VFS implementations ** might choose to implement the wal-index using a mmapped file. Because ** the wal-index is shared memory, SQLite does not support journal_mode=WAL ** on a network filesystem. All users of the database must be able to ** share memory. ** ** The wal-index is transient. After a crash, the wal-index can (and should ** be) reconstructed from the original WAL file. In fact, the VFS is required ** to either truncate or zero the header of the wal-index when the last ** connection to it closes. Because the wal-index is transient, it can ** use an architecture-specific format; it does not have to be cross-platform. ** Hence, unlike the database and WAL file formats which store all values ** as big endian, the wal-index can store multi-byte values in the native ** byte order of the host computer. ** ** The purpose of the wal-index is to answer this question quickly: Given ** a page number P and a maximum frame index M, return the index of the ** last frame in the wal before frame M for page P in the WAL, or return ** NULL if there are no frames for page P in the WAL prior to M. ** ** The wal-index consists of a header region, followed by an

one or ** more index blocks. *** The wal-index header contains the total number of frames within the WAL ** in the mxFrame field. *** Each index block except for the first contains information on ** HASHTABLE_NPAGE frames. The first index block contains information on ** HASHTABLE_NPAGE_ONE frames. The values of HASHTABLE_NPAGE_ONE and ** HASHTABLE_NPAGE are selected so that together the wal-index header and ** first index block are the same size as all other index blocks in the ** wal-index. *** Each index block contains two sections, a page-mapping that contains the ** database page number associated with each wal frame, and a hash-table ** that allows readers to query an index block for a specific page number. ** The page-mapping is an array of HASHTABLE_NPAGE (or HASHTABLE_NPAGE_ONE ** for the first index block) 32-bit page numbers. The first entry in the **first index-block contains the database page number corresponding to the ** first frame in the WAL file. The first entry in the second index block ** in the WAL file corresponds to the (HASHTABLE_NPAGE_ONE+1)th frame in ** the log, and so on. *** The last index block in a wal-index usually contains less than the full ** complement of HASHTABLE_NPAGE (or HASHTABLE_NPAGE_ONE) page-numbers, ** depending on the contents of the WAL file. This does not change the ** allocated size of the page-mapping array - the page-mapping array merely ** contains unused entries. *** Even without using the hash table, the last frame for page P ** can be found by scanning the page-mapping sections of each index block ** starting with the last index block and moving toward the first, and ** within each index block, starting at the end and moving toward the ** beginning. The first entry that equals P corresponds to the frame ** holding the content for that page. ** ** The hash table consists of HASHTABLE_NSLOT 16-bit unsigned integers. ** HASHTABLE_NSLOT = 2*HASHTABLE_NPAGE, and there is one entry in the ** hash table for each page number in the mapping section, so the hash ** table is never more than half full. The expected number of collisions ** prior to finding a match is 1. Each entry of the hash table is an ** 1-based index of an entry in the mapping section of the same ** index block. Let K be the 1-based index of the largest entry in ** the mapping section. (For index blocks other than the last, K will ** always be exactly HASHTABLE_NPAGE (4096) and for the last index block ** K will be (mxFrame%HASHTABLE_NPAGE).) Unused slots of the hash table ** contain a value of 0. ** ** To look for page P in the hash table, first compute a hash iKey on ** P as follows: *** iKey = (P * 383) % HASHTABLE_NSLOT ** ** Then start scanning entries of the hash table, starting with iKey ** (wrapping around to the beginning when the end of the hash table is ** reached) until an unused hash slot is found. Let the first unused slot ** be at index iUnused. (iUnused might be less than iKey if there was ** wrap-around.) Because the hash table is never more than half full, ** the search is guaranteed to eventually hit an unused entry. Let ** iMax be the value between iKey and iUnused, closest to iUnused, ** where aHash[iMax]==P. If there is no iMax entry (if there exists ** no hash slot such that aHash[i]==p) then page P is not in the ** current index block. Otherwise the iMax-th mapping entry of the ** current index block corresponds to the last entry that references ** page P. ** ** A hash search begins with the last index block and moves toward the ** first index block, looking for entries corresponding to page P. On ** average, only two or three slots in each index block need to be ** examined in order to either find the last entry for page P, or to ** establish that no such entry exists in the block. Each index block ** holds over 4000 entries. So two or three index blocks are sufficient ** to cover a typical 10 megabyte WAL file, assuming 1K pages. 8 or 10 ** comparisons (on average) suffice to either locate a frame in the ** WAL or to establish that the frame does not exist in the WAL. This ** is much faster than scanning the entire 10MB WAL. ** ** Note that entries are added in order of increasing K. Hence, one ** reader might be using some value K0 and a second reader that started ** at a later time (after additional transactions were added to the WAL ** and to the wal-index) might be using a different value K1, where K1>K0. ** Both readers can use the same hash table and mapping section to get ** the correct result. There may be entries in the hash table with ** K>K0 but to the first reader, those entries will appear to be unused ** slots in the hash table and so the first reader will get an answer as ** if no values greater than K0 had ever been inserted into the hash table ** in the first place - which is what reader one wants. Meanwhile, the ** second reader using K1 will see additional values that were inserted ** later, which is exactly what reader two wants. ** ** When a rollback occurs, the value of K is decreased. Hash table entries ** that correspond to frames greater than the new K value are removed ** from the hash table at this point.

label: code-design

- 60906. Overflow page pointer-map entry page
- 60907. SQLITE_VDBE_H
- 60908. ** PRAGMA compile_options *** Return the names of all compile-time options used in this build, ** one option per row.
- 60909. ** The methods above are in versions 1 through 3 of the sqlite_vfs object. ** New fields may be appended in future versions. The iVersion ** value will increment whenever this happens.
- 60910. Position read from poslist
- 60911. ** Install the diff hooks on the session object passed as the only ** argument.
- 60912. The doNotSpill NOSYNC bit is set during times when doing a sync of ** journal (and adding a new header) is not allowed. This occurs ** during calls to sqlite3PagerWrite() while trying to journal multiple ** pages belonging to the same sector. ** ** The doNotSpill ROLLBACK and OFF bits inhibits all cache spilling ** regardless of whether or not a sync is required. This is set during ** a rollback or by user request, respectively. ** ** Spilling is also prohibited when in an error state since that could ** lead to database corruption. In the current implementation it ** is impossible for sqlite3PcacheFetch() to be called with createFlag=3 ** while in the error state, hence it is impossible for this routine to ** be called in the error state. Nevertheless, we include a NEVER() ** test for the error state as a safeguard against future changes.
- 60913. OUT: Selected child node
- 60914. ** Obtain an SQLite statement handle that may be used to read data from the ** %_content table.
- 60915. Page number of free-list trunk page
- 60916. ** 2001 September 22 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** **** This is the implementation of generic hash-tables used in SQLite. ** We've modified it slightly to serve as a standalone hash table ** implementation for the full-text indexing module.
- 60917. The special operator TK_ROW means use the rowid for the first ** column in the FROM clause. This is used by the LIMIT and ORDER BY ** clause processing on UPDATE and DELETE statements.
- 60918. Open a cursor on zDoc/nDoc. Check if there are (nSnippet+nDesired) ** or more tokens in zDoc/nDoc.
- 60919. ** If this Btree is a candidate for shared cache, try to find an ** existing BtShared object that we can share with
- 60920. val now holds the upper bound of the coordinate pair
- 60921. Root node of tree for this task
- 60922. The node from which to extract a coordinate
- 60923. Coordinate decoded
- 60924. Total cost of this path ignoring sorting costs
- 60925. EV: R-19803-45884
- 60926. ** Load the Parse object passed as the first argument with an error ** message of the form: *** "sub-select returns N columns - expected M"
- 60927. EXPLAIN
- 60928. ***** End of fts3_write.c *****
- 60929. Size of input text in bytes
- 60930. Minimum required file size
- 60931. ** CAPI3REF: Testing Interface ** ** ^The sqlite3_test_control() interface is used to read out internal ** state of SQLite and to inject faults into SQLite for testing ** purposes. ^The first parameter is an operation code that determines ** the number, meaning, and operation of all subsequent parameters. ** ** This interface is not for use by applications. It exists solely ** for verifying the correct operation of the SQLite library. Depending ** on how the SQLite library is compiled, this interface might not exist. ** ** The details of the operation codes, their meanings, the parameters ** they take, and what they do are all subject to change without notice. ** Unlike most of the SQLite API, this function is not guaranteed to ** operate consistently from one release to the next.
- 60932. Looping over other all cursors
- 60933. Height of sub-trees Reinsert() has run on
- 60934. .szMalloc =
- 60935. This next condition is true if data has already been loaded from ** the sqlite_stat4 table. In this case ignore stat3 data.
- 60936. ** CAPI3REF: Enable Or Disable Extended Result Codes ** METHOD: sqlite3 ** ** ^The sqlite3_extended_result_codes() routine enables or disables the ** [extended result codes] feature of SQLite. ^The extended result ** codes are disabled by default for historical compatibility.
- 60937. nxplist ::= nxplist COMMA expr
- 60938. Page numbers of new pages before shuffling
- 60939. Page cache memory
- 60940. IN/OUT: Integer value
- 60941. The common case
- 60942. ** xBestIndex - Analyze a WHERE and ORDER BY clause.
- 60943. **comment:** Insert code to test every subexpression that can be completely ** computed using the current set of tables. ** ** This loop may run between one and three times, depending on the ** constraints to be generated. The value of stack variable iLoop ** determines the constraints coded by each iteration, as follows:

*** iLoop==1: Code only expressions that are entirely covered by pIdx. *** iLoop==2: Code remaining expressions that do not contain correlated ** sub-queries.
** iLoop==3: Code all remaining expressions. *** An effort is made to skip unnecessary iterations of the loop.

label: code-design

60944. Return the unixInodeInfo object here

60945. ** The maximum depth of an expression tree. This is limited to ** some extent by SQLITE_MAX_SQL_LENGTH. But sometime you might ** want to place more severe limits on the complexity of an ** expression. *** A value of 0 used to mean that the limit was not enforced. ** But that is no longer true. The limit is now strictly enforced ** at all times.

60946. **comment:** Read the first three 32-bit fields of the journal header: The nRec ** field, the checksum-initializer and the database size at the start ** of the transaction. Return an error code if anything goes wrong.

label: code-design

60947. **comment:** ** Change the size of an existing memory allocation. *** For this debugging implementation, we *always* make a copy of the ** allocation into a new place in memory. In this way, if the ** higher level code is using pointer to the old allocation, it is ** much more likely to break and we are much more likng to find ** the error.

label: code-design

60948. READWRITE

60949. Guaranteed by the code generator

60950. ** The smallest possible node-size is (512-64)==448 bytes. And the largest ** supported cell size is 48 bytes (8 byte rowid + ten 4 byte coordinates). ** Therefore all non-root nodes must contain at least 3 entries. Since ** 2^40 is greater than 2^64, an r-tree structure always has a depth of ** 40 or less.

60951. x.pSelect is valid (otherwise x.pList is)

60952. ** Allocate an Fts3MultiSegReader for each token in the expression headed ** by pExpr. *** An Fts3SegReader object is a cursor that can seek or scan a range of ** entries within a single segment b-tree. An Fts3MultiSegReader uses multiple ** Fts3SegReader objects internally to provide an interface to seek or scan ** within the union of all segments of a b-tree. Hence the name. *** If the allocated Fts3MultiSegReader just seeks to a single entry in a ** segment b-tree (if the term is not a prefix or it is a prefix for which ** there exists prefix b-tree of the right length) then it may be traversed ** and merged incrementally. Otherwise, it has to be merged into an in-memory ** doclist and then traversed.

60953. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** An tokenizer for SQL *** This file contains C code that splits an SQL input string up into ** individual tokens and sends those tokens one-by-one over to the ** parser for analysis.

60954. Name of left-most parent key column

60955. ** This routine prepares a memory cell for modification by breaking ** its link to a shallow copy and by marking any current shallow ** copies of this cell as invalid. *** This is used for testing and debugging only - to make sure shallow ** copies are not misused.

60956. Opcode: Param P1 P2 * * * * * This opcode is only ever present in sub-programs called via the ** OP_Program instruction. Copy a value currently stored in a memory ** cell of the calling (parent) frame to cell P2 in the current frames ** address space. This is used by trigger programs to access the new.* ** and old.* values. *** The address of the cell in the parent frame is determined by adding ** the value of the P1 argument to the value of the P1 argument to the ** calling OP_Program instruction.

60957. ** CAPI3REF: Close A BLOB Handle ** DESTRUCTOR: sqlite3_blob *** ^This function closes an open [BLOB handle]. ^The BLOB handle is closed ** unconditionally. Even if this routine returns an error code, the ** handle is still closed.^ ** ^If the blob handle being closed was opened for read-write access, and if ** the database is in auto-commit mode and there are no other open read-write ** blob handles or active write statements, the current transaction is ** committed. ^If an error occurs while committing the transaction, an error ** code is returned and the transaction rolled back. *** Calling this function with an argument that is not a NULL pointer or an ** open blob handle results in undefined behaviour. ^Calling this routine ** with a null pointer (such as would be returned by a failed call to ** [sqlite3_blob_open()]) is a harmless no-op. ^Otherwise, if this function ** is passed a valid open blob handle, the values returned by the ** sqlite3_errcode() and sqlite3errmsg() functions are set before returning.

60958. ** Write value iVal encoded as a varint to the PMA. Return ** SQLITE_OK if successful, or an SQLite error code if an error occurs.

60959. ** Declare that a function has been overloaded by a virtual table. *** If the function already exists as a regular global function, then ** this routine is a no-op. If the function does not exist, then create ** a new one that always throws a run-time error. *** When virtual tables intend to provide an overloaded function, they ** should call this routine to make sure the global function exists. ** A global function must exist in order for name resolution to work ** properly.

60960. Reader marks

60961. **comment:** Failure to set the bits in the InJournal bit-vectors is benign. ** It merely means that we might do some extra work to journal a ** page that does not need to be journaled. Nevertheless, be sure ** to test the case where a malloc error occurs while trying to set ** a bit in a bit vector.

label: code-design

60962. Size of read buffer in bytes

60963. ** This function is used to set the schema of a virtual table. It is only ** valid to call this function from within the xCreate() or xConnect() of a ** virtual table module.

60964. Table that change applies to

60965. ** Create the %_stat table if it does not already exist.

60966. Initialize the count of rows to be inserted

60967. ** Register a new tokenizer. This is the implementation of the ** fts5_api.xCreateTokenizer() method.

60968. Move level -1 to level iAbsLevel

60969. Allocate a page. The page that currently resides at pgnoRoot will ** be moved to the allocated page (unless the allocated page happens ** to reside at pgnoRoot).

60970. Write current value here

60971. ** A bitmap is an instance of the following structure. *** This bitmap records the existence of zero or more bits ** with values between 1 and iSize, inclusive. ** ** There are three possible representations of the bitmap. ** If iSize<=BITVEC_NBIT, then Bitvec.u.aBitmap[] is a straight ** bitmap. The least significant bit is bit 1. *** If iSize>BITVEC_NBIT and iDivisor==0 then Bitvec.u.aHash[] is ** a hash table that will hold up to BITVEC_MXHASH distinct values. *** Otherwise, the value i is redirected into one of BITVEC_NPTR ** sub-bitmaps pointed to by Bitvec.u.apSub[]. Each subbitmap ** handles up to iDivisor separate values of i. apSub[0] holds ** values between 1 and iDivisor. apSub[1] holds values between ** iDivisor+1 and 2*iDivisor. apSub[N] holds values between ** N*iDivisor+1 and (N+1)*iDivisor. Each subbitmap is normalized ** to hold deal with values between 1 and iDivisor.

60972. .u =

60973. ** Convert an SQL-style quoted string into a normal string by removing ** the quote characters. The conversion is done in-place. If the ** input does not begin with a quote character, then this routine ** is a no-op. *** The input string must be zero-terminated. A new zero-terminator ** is added to the dequoted string. ** ** The return value is -1 if no dequoting occurs or the length of the ** dequoted string, exclusive of the zero terminator, if dequoting does ** occur. *** 2002-Feb-14: This routine is extended to remove MS-Access style ** brackets from around identifiers. For example: "[a-b-c]" becomes ** "a-b-c".

60974. ** Return the current time as a Julian Day number in *pTimeOut.

60975. Doclist-index flag (1 bit)

60976. ** This routine processes the join information for a SELECT statement. ** ON and USING clauses are converted into extra terms of the WHERE clause. **

NATURAL joins also create extra WHERE clause terms. *** The terms of a FROM clause are contained in the Select.pSrc structure. ** The left most table is the first entry in Select.pSrc. The right-most ** table is the last entry. The join operator is held in the entry to ** the left. Thus entry 0 contains the join operator for the join between ** entries 0 and 1. Any ON or USING clauses associated with the join are ** also attached to the left entry. *** This routine returns the number of errors encountered.

60977. This routine should not be called if a prior error has occurred. ** But if (due to a coding error elsewhere in the system) it does get ** called, just return the same error code without doing anything.

60978. ONEPASS_OFF, or _SINGLE, or _MULTI

60979. ** Release any memory resources held by the Mem. Both the memory that is ** free by Mem.xDel and the Mem.zMalloc allocation are freed. *** Use this routine prior to clean up prior to abandoning a Mem, or to ** reset a Mem back to its minimum memory utilization. *** Use sqlite3VdbeMemSetNull() to release just the Mem.xDel space ** prior to inserting new content into the Mem.

60980. List for thread to write to a PMA

60981. ** Check to see if virtual table module pMod can be have an eponymous ** virtual table instance. If it can, create one if one does not already ** exist. Return non-zero if the eponymous virtual table instance exists ** when this routine returns, and return zero if it does not exist. *** An eponymous virtual table instance is one that is named after its ** module, and more importantly, does not require a CREATE VIRTUAL TABLE ** statement in order to come into existance.

Eponymous virtual table ** instances always exist. They cannot be DROP-ed. *** Any virtual table module for which xConnect and xCreate are the same ** method can have an eponymous virtual table instance.

60982. Allocate and populate the array of LcsIterator objects. The array ** contains one element for each matchable phrase in the query. *

60983. IN/OUT: Pointer to string buffer

60984. min/max aggregates seen. See note above

60985. Character set tests (like isspace(), isalpha() etc.)

60986. **comment:** ** The query generator uses an array of instances of this structure to ** help it analyze the subexpressions of the WHERE clause. Each WHERE ** clause subexpression is separated from the others by AND operators, ** usually, or sometimes subexpressions separated by OR. *** All WhereTerms are collected into a single WhereClause structure. ** The following identity holds: *** WhereTerm.pWC->a[WhereTerm.idx] == WhereTerm *** When a term is of the form: ** X <op> <expr> ** where X is a column name and <op> is one of certain operators, ** then WhereTerm.leftCursor and WhereTerm.u.leftColumn record the ** cursor number and column number for X. WhereTerm.eOperator records ** the <op> using a bitmask encoding defined by WO_xxx below. The ** use of a bitmask encoding for the operator allows us to search ** quickly for terms that match any of several different operators. *** A WhereTerm might also be two or more subterms connected by OR: *** (t1.X <op> <expr>) OR (t1.Y <op> <expr>) OR *** In this second case, wtFlag has the TERM_ORINFO bit set and eOperator==WO_OR ** and the WhereTerm.u.pOrInfo field points to auxiliary information that ** is collected about the OR clause. *** If a term in the WHERE clause does not match either of the two previous ** categories, then eOperator==0. The WhereTerm.pExpr field is still set ** to the original subexpression content and wtFlags is set up appropriately ** but no other fields in the WhereTerm object are meaningful. *** When eOperator!=0, prereqRight and prereqAll record sets of cursor numbers, ** but they do so indirectly. A single WhereMaskSet structure translates ** cursor number into bits and the translated bit is stored in the prereq ** fields. The translation is used in order to maximize the number of ** bits that will fit in a Bitmask. The VDBE cursor numbers might be ** spread out over the non-negative integers. For example, the cursor ** numbers might be 3, 8, 9, 10, 20, 23, 41, and 45. The WhereMaskSet ** translates these sparse cursor numbers into consecutive integers ** beginning with 0 in order to make the best possible use of the available ** bits in the Bitmask. So, in the example above, the cursor numbers ** would be mapped into integers 0 through 7. *** The number of terms in a join is limited by the number of bits ** in prereqRight and prereqAll. The default is 64 bits, hence SQLite ** is only able to process joins with 64 or fewer tables.

label: code-design

60987. **comment:** ** This function is called when the PmaReader corresponding to pIncr has ** finished reading the contents of aFile[0]. Its purpose is to "refill" ** aFile[0] such that the PmaReader should start rereading it from the ** beginning. *** For single-threaded objects, this is accomplished by literally reading ** keys from pIncr->pMerger and repopulating aFile[0]. ** For multi-threaded objects, all that is required is to wait until the ** background thread is finished (if it is not already) and then swap ** aFile[0] and aFile[1] in place. If the contents of pMerger have not ** been exhausted, this function also launches a new background thread ** to populate the new aFile[1]. *** SQLITE_OK is returned on success, or an SQLite error code otherwise.

label: code-design

60988. Hash table of column names

60989. IN/OUT: Decrement if row is deleted

60990. ** If the field type is eTGENERIC, then convert to either etEXP ** or etFLOAT, as appropriate.

60991. xSqllog

60992. DATABASE => ID

60993. ** Return the number of consecutive JsonNode slots need to represent ** the parsed JSON at pNode. The minimum answer is 1. For ARRAY and ** OBJECT types, the number might be larger. ** Appended elements are not counted. The value returned is the number ** by which the JsonNode counter should increment in order to go to the ** next peer value.

60994. Insertion failed because database is full

60995. SQLITE_ENABLE_STMT_SCANSTATUS

60996. TermOffset associated with next token

60997. True if no shared-cache backends

60998. **comment:** ** The value of the "pending" byte must be 0x40000000 (1 byte past the ** 1-gibabyte boundary) in a compatible database. SQLite never uses ** the database page that contains the pending byte. It never attempts ** to read or write that page. The pending byte page is set aside ** for use by the VFS layers as space for managing file locks. ** During testing, it is often desirable to move the pending byte to ** a different position in the file. This allows code that has to ** deal with the pending byte to run on files that are much smaller ** than 1 GiB. The sqlite3_test_control() interface can be used to ** move the pending byte. ** ** IMPORTANT: Changing the pending byte to any value other than ** 0x40000000 results in an incompatible database file format! ** Changing the pending byte during operation will result in undefined ** and incorrect behavior.

label: code-design

60999. ***** Include os.h in the middle of sqliteInt.h *****

61000. Custom rank function args

61001. Available memory

61002. Tail of table name e.g. "data", "config"

61003. ** Tokenize the nul-terminated string zText and add all tokens to the ** pending-terms hash-table. The docid used is that currently stored in ** p->iPrevDocid, and the column is specified by argument iCol. ** If successful, SQLITE_OK is returned. Otherwise, an SQLite error code.

61004. Next entry is on the current page

61005. Mask of type bits

61006. If this is an SQLITE_CHECKPOINT_RESTART or TRUNCATE operation, and the ** entire wal file has been copied into the database file, then block ** until all readers have finished using the wal file. This ensures that ** the next process to write to the database restarts the wal file.

61007. 275

61008. #include "sqlite3session.h"

61009. 360

61010. MEM cell holding data for the record to be inserted

61011. Name of table containing column, or NULL

61012. pName1->z might be NULL, but not pName1 itself

61013. 167

61014. (double)0 In case of SQLITE OMIT_FLOATING_POINT...

61015. ** Allocate cursors for the pTab table and all its indices and generate ** code to open and initialized those cursors. ** The cursor for the object that contains the complete data (normally ** the table itself, but the PRIMARY KEY index in the case of a WITHOUT ** ROWID table) is returned in *pDataCur. The first index cursor is ** returned in *pIdxCur. The number of indices is returned. ** Use iBase as the first cursor (either the *pDataCur for rowid tables ** or the first index for WITHOUT ROWID tables) if it is non-negative. ** If iBase is negative, then allocate the next available cursor. ** For a rowid table, *pDataCur will be exactly one less than *pIdxCur. ** For a WITHOUT ROWID table, *pDataCur will be somewhere in the range ** of *pIdxCurs, depending on where the PRIMARY KEY index appears on the ** pTab->pIndex list. ** If pTab is a virtual table, then this routine is a no-op and the ** *pDataCur and *pIdxCur values are left uninitialized.

61016. The function we are evaluating for match quality

61017. Determine the auto-incr-merge setting if unknown. If enabled, ** estimate the number of leaf blocks of content to be written

61018. (273) cmdlist ::= cmdlist ecmd

61019. Verify that the number of entries in the hash table exactly equals ** the number of entries in the mapping region.

61020. ** This function is used to count the entries in a column-list (a ** delta-encoded list of term offsets within a single column of a single ** row). When this function is called, *ppCollist should point to the ** beginning of the first varint in the column-list (the varint that ** contains the position of the first matching term in the column data). ** Before returning, *ppCollist is set to point to the first byte after ** the last varint in the column-list (either the 0x00 signifying the end ** of the position-list, or the 0x01 that precedes the column number of ** the next column in the position-list). ** The number of elements in the column-list is returned.

61021. If this is part of a compound SELECT, check that it has the right ** number of expressions in the select list.

61022. Inequality constraint at range start

61023. 1240

61024. Write the page pointer here

61025. Nul-terminated UTF-8 string <id>

61026. ** xSetOutputs callback used by detail=full and detail=col tables when no ** column filters are specified.

61027. Estimated average row size in bytes

61028. #include <unicode/ustring.h>
61029. ** Rtree virtual table module xConnect method.
61030. Log base 2 of minimum allocation size in bytes
61031. Root page of current object
61032. ** Return the SQL associated with a prepared statement with ** bound parameters expanded. Space to hold the returned string is ** obtained from sqlite3_malloc(). The caller is responsible for ** freeing the returned string by passing it to sqlite3_free(). *** The SQLITE_TRACE_SIZE_LIMIT puts an upper bound on the size of ** expanded bound parameters.
61033. Restriction (7)
61034. If one was specified, code the WHEN clause. If it evaluates to false ** (or NULL) the sub-vdbe is immediately halted by jumping to the ** OP_Halt inserted at the end of the program.
61035. Container of this cell
61036. State variables
61037. **comment:** Not used on win32
label: code-design
61038. **comment:** ** The VACUUM command is used to clean up the database, ** collapse free space, etc. It is modelled after the VACUUM command ** in PostgreSQL. The VACUUM command works as follows: *** (1) Create a new transient database file ** (2) Copy all content from the database being vacuumed into ** the new transient database file ** (3) Copy content from the transient database back into the ** original database. *** The transient database requires temporary disk space approximately ** equal to the size of the original database. The copy operation of ** step (3) requires additional temporary disk space approximately equal ** to the size of the original database for the rollback journal. ** Hence, temporary disk space that is approximately 2x the size of the ** original database is required. Every page of the database is written ** approximately 3 times: Once for step (2) and twice for step (3). ** Two writes per page are required in step (3) because the original ** database content must be written into the rollback journal prior to ** overwriting the database with the vacuumed content. *** Only 1x temporary space and only 1x writes would be required if ** the copy of step (3) were replaced by deleting the original database ** and renaming the transient database as the original. But that will ** not work if other processes are attached to the original database. ** And a power loss in between deleting the original and renaming the ** transient would cause the database file to appear to be deleted ** following reboot.
label: code-design
61039. ** Free an outstanding memory allocation. *** This function assumes that the necessary mutexes, if any, are ** already held by the caller. Hence "Unsafe".
61040. The next action
61041. neardist_opt ::=
61042. 172
61043. ** Open a connection to the WAL file zWalName. The database file must ** already be opened on connection pDbFd. The buffer that zWalName points ** to must remain valid for the lifetime of the returned Wal* handle. *** A SHARED lock should be held on the database file when this function ** is called. The purpose of this SHARED lock is to prevent any other ** client from unlinking the WAL or wal-index file. If another process ** were to do this just after this client opened one of these files, the ** system would be badly broken. *** If the log file is successfully opened, SQLITE_OK is returned and ** *ppWal is set to point to a new WAL handle. If an error occurs, ** an SQLite error code is returned and *ppWal is left unmodified.
61044. ** Allowed values of unixFile.fsFlags
61045. If this DELETE cannot use the ONEPASS strategy, this is the ** end of the WHERE loop
61046. Union of locks held by connections other than "p"
61047. Dynamically allocated space for aArg[]
61048. ** Generate either a changeset (if argument bPatchset is zero) or a patchset ** (if it is non-zero) based on the current contents of the session object ** passed as the first argument. *** If no error occurs, SQLITE_OK is returned and the new changeset/patchset ** stored in output variables *pnChangeset and *ppChangeset. Or, if an error ** occurs, an SQLite error code is returned and both output variables set ** to 0.
61049. ** 2003 January 11 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used to implement the
sqlite3_set_authorizer() ** API. This facility is an optional feature of the library. Embedded ** systems that do not need this facility may omit it by recompiling ** the library with -DSQLITE_OMIT_AUTHORIZATION=1
61050. ** Initialize memory that will be converted into a BtCursor object. *** The simple approach here would be to memset() the entire object ** to zero. But it turns out that the apPage[] and aiIdx[] arrays ** do not need to be zeroed and they are large, so we can save a lot ** of run-time by skipping the initialization of those elements.
61051. Buffer to assemble frame-header in
61052. Filename as passed to xOpen
61053. The content of the page
61054. xClose
61055. An ascii-range separator character. End of token.
61056. Mask for the current column
61057. Copy data from page to buffer (a read operation)
61058. CROSS
61059. ** If the REQUIRE_RESEEK flag is set on the cursor passed as the first ** argument, close and reopen all Fts5IndexIter iterators that the cursor ** is using. Then attempt to move the cursor to a rowid equal to or later ** (in the cursors sort order - ASC or DESC) than the current rowid. *** If the new rowid is not equal to the old, set output parameter *pbSkip ** to 1 before returning. Otherwise, leave it unchanged. *** Return SQLITE_OK if successful or if no reseek was required, or an ** error code if an error occurred.
61060. IN/OUT: Record pointer
61061. Mem.z points to an agg function context
61062. ** Invalidate the overflow cache of the cursor passed as the first argument. ** on the shared btree structure pBt.
61063. Address of jump over the delete logic
61064. #ifndef SQLITE_OMIT_TRIGGER
61065. ** Set an Fts3SegReader cursor to point at EOF.
61066. xFindFunction - function overloading
61067. Opcode: Return P1 * * * * * Jump to the next instruction after the address in register P1. After ** the jump, register P1 becomes undefined.
61068. ** Implementation of API function xQueryPhrase().
61069. Cursor for IPK b-tree
61070. Next WhereLoop object in the WhereClause
61071. If a temporary file is requested, it is not opened immediately. ** In this case we accept the default page size and delay actually ** opening the file until the first call to OsWrite(). *** This branch is also run for an in-memory database. An in-memory ** database is the same as a temp-file that is never written out to ** disk and uses an in-memory rollback journal. *** This branch also runs for files marked as immutable.
61072. Current read buffer
61073. ** The following is the implementation of an SQL function that always ** fails with an error message stating that the function is used in the ** wrong context. The sqlite3_overload_function() API might construct ** SQL function that use this routine so that the functions will exist ** for name resolution but are actually overloaded by the xFindFunction ** method of virtual tables.
61074. Temporary storage for the page
61075. This branch may be executed with Pager.journalMode==MEMORY if ** a hot-journal was just rolled back. In this case the journal ** file should be closed and deleted. If this connection writes to ** the database file, it will do so using an in-memory journal.
61076. 1060
61077. ** Return TRUE if the given string is a row-id column name.
61078. 100's digit
61079. ** Free an sqlite3_value object
61080. TK_LE
61081. Size of zDoc in bytes

61082. If the pager is already in exclusive-mode, the WAL module will use ** heap-memory for the wal-index instead of the VFS shared-memory ** implementation.
Take the exclusive lock now, before opening the WAL ** file, to make sure this is safe.

61083. a &= (0x7f<<29)|(0x7f<<15)|(0xff);

61084. Once all the cookies have been verified and transactions opened, ** obtain the required table-locks. This is a no-op unless the ** shared-cache feature is enabled.

61085. Amount of space allocated to zMasterPtr[]

61086. Use BtreeGetReserveNoMutex() for the source b-tree, as although it is ** guaranteed that the shared-mutex is held by this thread, handle ** p->pSrc may not actually be the owner.

61087. ** A read or write transaction may or may not be active on database handle ** db. If a transaction is active, commit it. If there is a ** write-transaction spanning more than one database file, this routine ** takes care of the master journal trickery.

61088. ** Bits for the mask used as the idxNum value by xBestIndex/xFilter.

61089. Set the VdbeCursor.isTable variable. Previous versions of ** SQLite used to check if the root-page flags were sane at this point ** and report database corruption if they were not, but this check has ** since moved into the btree layer.

61090. Locate the cursor number of the Current table

61091. IMP: R-65497-44594

61092. Iterate through all segments, from oldest to newest. Add them to ** the new Fts5Level object so that pLvl->aSeg[0] is the oldest ** segment in the data structure.

61093. Flags for OsSync() (or 0)

61094. Array of size pConfig->nCol

61095. If there is not a recovery running in another thread or process ** then convert BUSY errors to WAL_RETRY. If recovery is known to ** be running, convert BUSY to BUSY_RECOVERY. There is a race here ** which might cause WAL_RETRY to be returned even if BUSY_RECOVERY ** would be technically correct. But the race is benign since with ** WAL_RETRY this routine will be called again and will probably be ** right on the second iteration.

61096. Currently executing extension function

61097. xClose - close a cursor

61098. The loop to adjust downward

61099. A rowid/docid conflict.

61100. ** Remove from the column cache any entries that were added since the ** the previous sqlite3ExprCachePush operation. In other words, restore ** the cache to the state it was in prior the most recent Push.

61101. Child node (or 0 if this is a leaf)

61102. ** Flush all data associated with the SegmentWriter object pWriter to the ** database. This function must be called after all terms have been added ** to the segment using fts3SegWriterAdd(). If successful, SQLITE_OK is ** returned. Otherwise, an SQLite error code.

61103. EVIDENCE-OF: R-24089-57979 If a page contains no cells (which is only ** possible for a root page of a table that contains no rows) then the ** offset to the cell content area will equal the page size minus the ** bytes of reserved space.

61104. First in list of all tokenizer modules

61105. ** Attempt to add, subtract, or multiply the 64-bit signed value iB against ** the other 64-bit signed integer at *pA and store the result in *pA. ** Return 0 on success. Or if the operation would have resulted in an ** overflow, leave *pA unchanged and return 1.

61106. Free idxStr using sqlite3_free() if true

61107. plus_num ::= PLUS INTEGER|FLOAT

61108. True for first term on the node

61109. Load the initial configuration

61110. Another SELECT immediately to our left

61111. Statement cursor is currently pointing at

61112. Size of pSorter in bytes

61113. **comment:** ** CAPI3REF: Standard File Control Opcodes ** KEYWORDS: {file control opcodes} {file control opcode} *** These integer constants are opcodes for the xFileControl method ** of the [sqlite3_io_methods] object and for the [sqlite3_file_control()] ** interface. *** ** [[SQLITE_FCNTL_LOCKSTATE]] ** The [SQLITE_FCNTL_LOCKSTATE] opcode is used for debugging. This ** opcode causes the xFileControl method to write the current state of ** the lock (one of [SQLITE_LOCK_NONE], [SQLITE_LOCK_SHARED], ** [SQLITE_LOCK_RESERVED], [SQLITE_LOCK_PENDING], or [SQLITE_LOCK_EXCLUSIVE]) ** into an integer that the pArg argument points to. This capability ** is used during testing and is only available when the SQLITE_TEST ** compile-time option is used. *** [[SQLITE_FCNTL_SIZE_HINT]] ** The [SQLITE_FCNTL_SIZE_HINT] opcode is used by SQLite to give the VFS ** layer a hint of how large the database file will grow to be during the ** current transaction. This hint is not guaranteed to be accurate but it ** is often close. The underlying VFS might choose to preallocate database ** file space based on this hint in order to help writes to the database ** file run faster. *** [[SQLITE_FCNTL_CHUNK_SIZE]] ** The [SQLITE_FCNTL_CHUNK_SIZE] opcode is used to request that the VFS ** extends and truncates the database file in chunks of a size specified ** by the user. The fourth argument to [sqlite3_file_control()] should ** point to an integer (type int) containing the new chunk-size to use ** for the nominated database. Allocating database file space in large ** chunks (say 1MB at a time), may reduce file-system fragmentation and ** improve performance on some systems. *** [[SQLITE_FCNTL_FILE_POINTER]] ** The [SQLITE_FCNTL_FILE_POINTER] opcode is used to obtain a pointer ** to the [sqlite3_file] object associated with a particular database ** connection. See also [SQLITE_FCNTL_JOURNAL_POINTER]. *** [[SQLITE_FCNTL_JOURNAL_POINTER]] ** The [SQLITE_FCNTL_JOURNAL_POINTER] opcode is used to obtain a pointer ** to the [sqlite3_file] object associated with the journal file (either ** the [rollback journal] or the [write-ahead log]) for a particular database ** connection. See also [SQLITE_FCNTL_FILE_POINTER]. *** [[SQLITE_FCNTL_SYNC OMITTED]] ** No longer in use. *** [[SQLITE_FCNTL_SYNC]] ** The [SQLITE_FCNTL_SYNC] opcode is generated internally by SQLite and ** sent to the VFS immediately before the xSync method is invoked on a ** database file descriptor. Or, if the xSync method is not invoked ** because the user has configured SQLite with ** [PRAGMA synchronous | PRAGMA synchronous=OFF] it is invoked in place ** of the xSync method. In most cases, the pointer argument passed with ** this file-control is NULL. However, if the database file is being synced ** as part of a multi-database commit, the argument points to a nul-terminated ** string containing the transactions master-journal file name. VFSes that ** do not need this signal should silently ignore this opcode. Applications ** should not call [sqlite3_file_control()] with this opcode as doing so may ** disrupt the operation of the specialized VFSes that do require it. *** [[SQLITE_FCNTL_COMMIT_PHASETWO]] ** The [SQLITE_FCNTL_COMMIT_PHASETWO] opcode is generated internally by SQLite ** and sent to the VFS after a transaction has been committed immediately ** but before the database is unlocked. VFSes that do not need this signal ** should silently ignore this opcode. Applications should not call ** [sqlite3_file_control()] with this opcode as doing so may disrupt the ** operation of the specialized VFSes that do require it. *** [[SQLITE_FCNTL_WIN32_AV_RETRY]] ** ^The [SQLITE_FCNTL_WIN32_AV_RETRY] opcode is used to configure automatic ** retry counts and intervals for certain disk I/O operations for the ** windows [VFS] in order to provide robustness in the presence of ** anti-virus programs. By default, the windows VFS will retry file read, ** file write, and file delete operations up to 10 times, with a delay ** of 25 milliseconds before the first retry and with the delay increasing ** by an additional 25 milliseconds with each subsequent retry. This ** opcode allows these two values (10 retries and 25 milliseconds of delay) ** to be adjusted. The values are changed for all database connections ** within the same process. The argument is a pointer to an array of two ** integers where the first integer is the new retry count and the second ** integer is the delay. If either integer is negative, then the setting ** is not changed but instead the prior value of that setting is written ** into the array entry, allowing the current retry settings to be ** interrogated. The zDbName parameter is ignored. *** [[SQLITE_FCNTL_PERSIST_WAL]] ** ^The [SQLITE_FCNTL_PERSIST_WAL] opcode is used to set or query the ** persistent [WAL | Write Ahead Log] setting. By default, the auxiliary ** write ahead log and shared memory files used for transaction control ** are automatically deleted when the latest connection to the database ** closes. Setting persistent WAL mode causes those files to persist after ** close. Persisting the files is useful when other processes that do not ** have write permission on the directory containing the database file want ** to read the database file, as the WAL and shared memory files must exist ** in order for the database to be readable. The fourth parameter to ** [sqlite3_file_control()] for this opcode should be a pointer to an integer. ** That integer is 0 to disable persistent WAL mode or 1 to enable persistent ** WAL mode. If the integer is -1, then it is overwritten with the current ** WAL persistence setting. *** [[SQLITE_FCNTL_POWERSAFE_OVERWRITE]] ** ^The [SQLITE_FCNTL_POWERSAFE_OVERWRITE] opcode is used to set or query the ** persistent "powersafe-overwrite" or "PSOW" setting. The PSOW setting ** determines the [SQLITE_IOCAP_POWERSAFE_OVERWRITE] bit of the ** xDeviceCharacteristics methods. The fourth parameter to ** [sqlite3_file_control()] for this opcode should be a pointer to an integer. ** That integer is 0 to disable zero-damage mode or 1 to enable zero-damage ** mode. If the integer is -1, then it is overwritten with the current ** zero-damage mode setting. *** [[SQLITE_FCNTL_OVERWRITE]] ** ^The [SQLITE_FCNTL_OVERWRITE] opcode is invoked by SQLite after opening ** a write transaction to indicate that, unless it is rolled back for some ** reason, the entire database file will be overwritten by the current ** transaction. This is used by VACUUM operations. *** [[SQLITE_FCNTL_VFSNAME]] ** ^The [SQLITE_FCNTL_VFSNAME] opcode can be used to obtain the names of ** all [VFSes] in the VFS stack. The names are of all VFS shims and the ** final bottom-level VFS are written into memory obtained from ** [sqlite3_malloc()] and the result is stored in the char* variable ** that the fourth parameter of [sqlite3_file_control()] points to. ** The caller is responsible for freeing the memory when done. As with ** all file-control

actions, there is no guarantee that this will actually do anything. Callers should initialize the char* variable to a NULL pointer in case this file-control is not implemented. This file-control is intended for diagnostic use only.

[SQLITE_FCNTL_VFS_POINTER] opcode finds a pointer to the top-level [VFSes] currently in use. The argument X in [SQLITE_FCNTL_VFS_POINTER] must be of type "[sqlite3_vfs]". This opcodes will set *X to a pointer to the top-level VFS. When there are multiple VFS shims in the stack, this opcode finds the upper-most shim only.

Whenever a [PRAGMA] statement is parsed, an [SQLITE_FCNTL_PRAGMA] file control is sent to the open [sqlite3_file] object corresponding to the database file to which the pragma statement refers. The argument to the [SQLITE_FCNTL_PRAGMA] file control is an array of pointers to strings (char**) in which the second element of the array is the name of the pragma and the third element is the argument to the pragma or NULL if the pragma has no argument. The handler for an [SQLITE_FCNTL_PRAGMA] file control can optionally make the first element of the char** argument point to a string obtained from [sqlite3_mprintf()] or the equivalent and that string will become the result of the pragma or the error message if the pragma fails. If the [SQLITE_FCNTL_PRAGMA] file control returns [SQLITE_NOTFOUND], then normal [PRAGMA] processing continues. If the [SQLITE_FCNTL_PRAGMA] file control returns [SQLITE_OK], then the parser assumes that the VFS has handled the PRAGMA itself and the parser generates a no-op prepared statement if result string is NULL, or that returns a copy of the result string if the string is non-NULL. If the [SQLITE_FCNTL_PRAGMA] file control returns any result code other than [SQLITE_OK] or [SQLITE_NOTFOUND], that means that the VFS encountered an error while handling the [PRAGMA] and the compilation of the PRAGMA fails with an error. The [SQLITE_FCNTL_PRAGMA] file control occurs at the beginning of pragma statement analysis and so it is able to override built-in [PRAGMA] statements.

[SQLITE_FCNTL_BUSYHANDLER] file-control may be invoked by SQLite on the database file handle shortly after it is opened in order to provide a custom VFS with access to the connections busy-handler callback. The argument is of type (void**) - an array of two (void *) values. The first (void *) actually points to a function of type (int (*) (void *)). In order to invoke the connections busy-handler, this function should be invoked with the second (void *) in the array as the only argument. If it returns non-zero, then the operation should be retried. If it returns zero, the custom VFS should abandon the current operation.

[SQLITE_FCNTL_TEMPFILENAME] Application can invoke the [SQLITE_FCNTL_TEMPFILENAME] file-control to have SQLite generate a temporary filename using the same algorithm that is followed to generate temporary filenames for TEMP tables and other internal uses. The argument should be a char** which will be filled with the filename written into memory obtained from [sqlite3_malloc0]. The caller should invoke [sqlite3_free0] on the result to avoid a memory leak.

[SQLITE_FCNTL_MMAP_SIZE] The [SQLITE_FCNTL_MMAP_SIZE] file control is used to query or set the maximum number of bytes that will be used for memory-mapped I/O. The argument is a pointer to a value of type sqlite3_int64 that is an advisory maximum number of bytes in the file to memory map. The pointer is overwritten with the old value. The limit is not changed if the value originally pointed to is negative, and so the current limit can be queried by passing in a pointer to a negative number. This file-control is used internally to implement [PRAGMA mmap_size].

[SQLITE_FCNTL_TRACE] The [SQLITE_FCNTL_TRACE] file control provides advisory information to the VFS about what the higher layers of the SQLite stack are doing. This file control is used by some VFS activity tracing [shims]. The argument is a zero-terminated string. Higher layers in the SQLite stack may generate instances of this file control if the [SQLITE_USE_FCNTL_TRACE] compile-time option is enabled.

[SQLITE_FCNTL_HAS_MOVED] The [SQLITE_FCNTL_HAS_MOVED] file control interprets its argument as a pointer to an integer and it writes a boolean into that integer depending on whether or not the file has been renamed, moved, or deleted since it was first opened.

[SQLITE_FCNTL_WIN32_GET_HANDLE] The [SQLITE_FCNTL_WIN32_GET_HANDLE] opcode can be used to obtain the underlying native file handle associated with a file handle. This file control interprets its argument as a pointer to a native file handle and writes the resulting value there.

[SQLITE_FCNTL_WIN32_SET_HANDLE] The [SQLITE_FCNTL_WIN32_SET_HANDLE] opcode is used for debugging. This opcode causes the xFileControl method to swap the file handle with the one pointed to by the pArg argument. This capability is used during testing and only needs to be supported when SQLITE_TEST is defined.

[SQLITE_FCNTL_WAL_BLOCK] The [SQLITE_FCNTL_WAL_BLOCK] is a signal to the VFS layer that it might be advantageous to block on the next WAL lock if the lock is not immediately available. The WAL subsystem issues this signal during rare circumstances in order to fix a problem with priority inversion. Applications should use this file-control.

[SQLITE_FCNTL_ZIPVFS] The [SQLITE_FCNTL_ZIPVFS] opcode is implemented by zipvfs only. All other VFS should return SQLITE_NOTFOUND for this opcode.

[SQLITE_FCNTL_RBU] The [SQLITE_FCNTL_RBU] opcode is implemented by the special VFS used by the RBU extension only. All other VFS should return SQLITE_NOTFOUND for this opcode.

label: code-design

61114. ** This is the implementation of a scalar SQL function used to test the expression parser. It should be called as follows:
- ```
<expr>, <column 1>, ...);
```
- The first argument, <tokenizer>, is the name of the fts3 tokenizer used to parse the query expression (see README.tokenizers). The second argument is the query expression to parse. Each subsequent argument is the name of a column of the fts3 table that the query expression may refer to. For example:
- ```
SELECT fts3_exprtest('simple', 'Bill col2:Bloggs', 'col1', 'col2');
```
61115. Checksum of string zMaster
61116. OUT: True if plan uses an IN(...) op
61117. Initial size of write buffer
61118. Data is coming from a SELECT or from a multi-row VALUES clause. Generate a co-routine to run the SELECT.
61119. Index of a mem5.aPool[] slot
61120. Page number of the frame
61121. ** 2015-03-02 ** The author disclaims copyright to this source code. In place of a legal notice, here is a blessing: May you do good and not evil. May you find forgiveness for yourself and forgive others. May you share freely, never taking more than you give.
- ***** This file contains code that is specific to Wind River's VxWorks
61122. Statement to iterate through segments
61123. Operator for the combined expression
61124. Unique file ID
61125. ** This routine sets the progress callback for an Sqlite database to the given callback function with the given argument. The progress callback will be invoked every nOps opcodes.
61126. If values are not available for all fields of the index to the left of this one, no estimate can be made. Return SQLITE_NOTFOUND.
61127. If triggers are not supported by this compile then the statement machine used to detect the end of a statement is much simpler
61128. Omit this entire file if SQLITE_ENABLE_UNLOCK_NOTIFY is not defined.
61129. Default values for second and subsequent columns need to match.
61130. expr ::= expr PLUS|MINUS expr
61131. conch host and lock path match
61132. OUT: Rank function name
61133. ifndef SQLITE OMIT_TRIGGER
61134. Checksum based on contents of indexes
61135. Number of indices
61136. Database name (e.g. "main")
61137. 770
61138. EVIDENCE-OF: R-22035-46182 The SQLITE_CONFIG_GETPCACHE2 option takes a single argument which is a pointer to an sqlite3_pcachemethods2 object. SQLite copies of the current page cache implementation into that object.
61139. Only delete if aRegIdx!=0 && aRegIdx[i]>0
61140. ** The input, pIn, is a binary tree (or subtree) of RowSetEntry objects. Convert this tree into a linked list connected by the pRight pointers and return pointers to the first and last elements of the new list.
61141. Allocated size of aFirst[]
61142. SELECT used to determine types and collations
61143. ** Make sure pMem->z points to a writable allocation of at least min(n,32) bytes. If the bPreserve argument is true, then copy of the content of pMem->z into the new allocation. pMem must be either a string or blob if bPreserve is true. If bPreserve is false, any prior content in pMem->z is discarded.
61144. If we reach this point it means that the new candidate path needs to be added to the set of best-so-far paths.
61145. **comment:** ** CAPI3REF: Compiling An SQL Statement ** KEYWORDS: {SQL statement compiler} ** METHOD: sqlite3 ** CONSTRUCTOR: sqlite3_stmt ** To execute an SQL statement, it must first be compiled into a byte-code program using one of these routines. Or, in other words, these routines are constructors for the [prepared statement] object. ** The preferred routine to use is [sqlite3_prepare_v20]. The [sqlite3_prepare0] interface is legacy and

should be avoided. ** [sqlite3_prepare_v3()] has an extra "prepFlags" option that is used ** for special purposes. *** The use of the UTF-8 interfaces is preferred, as SQLite currently ** does all parsing using UTF-8. The UTF-16 interfaces are provided ** as a convenience. The UTF-16 interfaces work by converting the ** input text into UTF-8, then invoking the corresponding UTF-8 interface. *** The first argument, "db", is a [database connection] obtained from a ** prior successful call to [sqlite3_open()], [sqlite3_open_v2()] or ** [sqlite3_open16()]. The database connection must not have been closed. *** The second argument, "zSql", is the statement to be compiled, encoded ** as either UTF-8 or UTF-16. The sqlite3_prepare(), sqlite3_prepare_v2(), ** and sqlite3_prepare_v3() ** interfaces use UTF-8, and sqlite3_prepare16(), sqlite3_prepare16_v2(), ** and sqlite3_prepare16_v3() use UTF-16. *** ^If the nByte argument is negative, then zSql is read up to the ** first zero terminator. ^If nByte is positive, then it is the ** number of bytes read from zSql. ^If nByte is zero, then no prepared ** statement is generated. ** If the caller knows that the supplied string is nul-terminated, then ** there is a small performance advantage to passing an nByte parameter that ** is the number of bytes in the input string <i>including</i> ** the nul-terminator. *** ^If pzTail is not NULL then *pzTail is made to point to the first byte ** past the end of the first SQL statement in zSql. These routines only ** compile the first statement in zSql, so *pzTail is left pointing to ** what remains uncompiled. *** ^ppStmt is left pointing to a compiled [prepared statement] that can be ** executed using [sqlite3_step()]. ^If there is an error, *ppStmt is set ** to NULL. ^If the input text contains no SQL (if the input is an empty ** string or a comment) then *ppStmt is set to NULL. ** The calling procedure is responsible for deleting the compiled ** SQL statement using [sqlite3_finalize()] after it has finished with it. ** ppStmt may not be NULL. *** ^On success, the sqlite3_prepare() family of routines return [SQLITE_OK]; ** otherwise an [error code] is returned. *** The sqlite3_prepare_v2(), sqlite3_prepare_v3(), sqlite3_prepare16_v2(), ** and sqlite3_prepare16_v3() interfaces are recommended for all new programs. ** The older interfaces (sqlite3_prepare() and sqlite3_prepare16()) ** are retained for backwards compatibility, but their use is discouraged. *** ^In the "vX" interfaces, the prepared statement ** that is returned (the [sqlite3_stmt] object) contains a copy of the ** original SQL text. This causes the [sqlite3_step()] interface to ** behave differently in three ways: *** ** ** ^If the database schema changes, instead of returning [SQLITE_SCHEMA] as it ** always used to do, [sqlite3_step()] will automatically recompile the SQL ** statement and try to run it again. As many as [SQLITE_MAX_SCHEMA_RETRY] ** retries will occur before sqlite3_step() gives up and returns an error. ** ** ** ^When an error occurs, [sqlite3_step()] will return one of the detailed ** [error codes] or [extended error codes]. ^The legacy behavior was that ** [sqlite3_step()] would only return a generic [SQLITE_ERROR] result code ** and the application would have to make a second call to [sqlite3_reset()] ** in order to find the underlying cause of the problem. With the "v2" prepare ** interfaces, the underlying reason for the error is returned immediately. ** ** ** ^If the specific value bound to [parameter | host parameter] in the ** WHERE clause might influence the choice of query plan for a statement, ** then the statement will be automatically recompiled, as if there had been ** a schema change, on the first [sqlite3_step()] call following any change ** to the [sqlite3_bind_text | bindings] of that [parameter]. ** ^The specific value of WHERE-clause [parameter] might influence the ** choice of query plan if the parameter is the left-hand side of a [LIKE] ** or [GLOB] operator or if the parameter is compared to an indexed column ** and the [SQLITE_ENABLE_STAT3] compile-time option is enabled. ** ** <p>sqlite3_prepare_v3() differs from sqlite3_prepare_v2() only in having ** the extra prepFlags parameter, which is a bit array consisting of zero or ** more of the [SQLITE_PERSISTENT|SQLITE_PREPARE_*] flags. ^The ** sqlite3_prepare_v2() interface works exactly the same as ** sqlite3_prepare_v3() with a zero prepFlags parameter. **

label: code-design

61146. There is no prefix-length field for first term in a node

61147. Term to compare to

61148. Write new OR root to *(ppOr)++

61149. Register keeping track of errors remaining

61150. Estimated output rows per loop

61151. 298

61152. Number terms in the ORDER BY clause

61153. **comment:** ** CAPI3REF: Configuration Options ** KEYWORDS: {configuration option} *** These constants are the available integer configuration options that ** can be passed as the first argument to the [sqlite3_config()] interface. *** New configuration options may be added in future releases of SQLite. ** Existing configuration options might be discontinued. Applications ** should check the return code from [sqlite3_config()] to make sure that ** the call worked.

The [sqlite3_config()] interface will return a ** non-zero [error code] if a discontinued or unsupported configuration option ** is invoked. *** ** [[SQLITE_CONFIG_SINGLETHREAD]] <dt>SQLITE_CONFIG_SINGLETHREAD</dt> ** <dd>There are no arguments to this option. ^This option sets the ** [threading mode] to Single-thread. In other words, it disables ** all mutexing and puts SQLite into a mode where it can only be used ** by a single thread. ^If SQLite is compiled with ** the [SQLITE_THREADS=SAFE | SQLITE_THREADS=SAFE=0] compile-time option then ** it is not possible to change the [threading mode] from its default ** value of Single-thread and so [sqlite3_config()] will return ** [SQLITE_ERROR] if called with the SQLITE_CONFIG_SINGLETHREAD ** configuration option.</dd> *** [[SQLITE_CONFIG_MULTITHREAD]]

<dt>SQLITE_CONFIG_MULTITHREAD</dt> ** <dd>There are no arguments to this option. ^This option sets the ** [threading mode] to Multi-thread. In other words, it disables ** mutexing on [database connection] and [prepared statement] objects. ** The application is responsible for serializing access to ** [database connections] and [prepared statements]. But other mutexes ** are enabled so that SQLite will be safe to use in a multi-threaded ** environment as long as no two threads attempt to use the same ** [database connection] at the same time. ^If SQLite is compiled with ** the [SQLITE_THREADS=SAFE | SQLITE_THREADS=SAFE=0] compile-time option then ** it is not possible to set the Multi-thread [threading mode] and ** [sqlite3_config()] will return [SQLITE_ERROR] if called with the ** SQLITE_CONFIG_MULTITHREAD configuration option.</dd> *** [[SQLITE_CONFIG_SERIALIZED]]

<dt>SQLITE_CONFIG_SERIALIZED</dt> ** <dd>There are no arguments to this option. ^This option sets the ** [threading mode] to Serialized. In other words, this option enables ** all mutexes including the recursive ** mutexes on [database connection] and [prepared statement] objects. ** In this mode (which is the default when SQLite is compiled with ** [SQLITE_THREADS=SAFE=1]) the SQLite library will itself serialize access ** to [database connections] and [prepared statements] so that the ** application is free to use the same [database connection] or the ** same [prepared statement] in different threads at the same time. ** ^If SQLite is compiled with ** the [SQLITE_THREADS=SAFE | SQLITE_THREADS=SAFE=0] compile-time option then ** it is not possible to set the Serialized [threading mode] and ** [sqlite3_config()] will return [SQLITE_ERROR] if called with the ** SQLITE_CONFIG_SERIALIZED configuration option.</dd> *** [[SQLITE_CONFIG_MALLOC]] <dt>SQLITE_CONFIG_MALLOC</dt> ** <dd> ^The SQLITE_CONFIG_MALLOC option takes a single argument which is ** a pointer to an instance of the [sqlite3_mem_methods] structure. ** The argument specifies ** alternative low-level memory allocation routines to be used in place of ** the memory allocation routines built into SQLite.^& SQLite makes ** its own private copy of the content of the [sqlite3_mem_methods] structure ** before the [sqlite3_config()] call returns.</dd> *** [[SQLITE_CONFIG_GETMALLOC]]

<dt>SQLITE_CONFIG_GETMALLOC</dt> ** <dd> ^The SQLITE_CONFIG_GETMALLOC option takes a single argument which ** is a pointer to an instance of the [sqlite3_mem_methods] structure. ** The [sqlite3_mem_methods] ** structure is filled with the currently defined memory allocation routines. ^& This option can be used to overload the default memory allocation ** routines with a wrapper that simulates memory allocation failure or ** tracks memory usage, for example. </dd> *** [[SQLITE_CONFIG_MEMSTATUS]] <dt>SQLITE_CONFIG_MEMSTATUS</dt> ** <dd> ^The

SQLITE_CONFIG_MEMSTATUS option takes single argument of type int, ** interpreted as a boolean, which enables or disables the collection of ** memory allocation statistics. ^(When memory allocation statistics are ** disabled, the following SQLite interfaces become non-operational: ** ** [sqlite3_memory_used()] ** [sqlite3_memory_highwater()] ** [sqlite3_soft_heap_limit64()] ** [sqlite3_status64()] **) ^& Memory allocation statistics are enabled by default unless SQLite is ** compiled with [SQLITE_DEFAULT_MEMSTATUS]=0 in which case memory ** allocation statistics are disabled by default. ** </dd> *** [[SQLITE_CONFIG_SCRATCH]] <dt>SQLITE_CONFIG_SCRATCH</dt> ** <dd> ^The

SQLITE_CONFIG_SCRATCH option specifies a static memory buffer ** that SQLite can use for scratch memory. ^(There are three arguments ** to SQLITE_CONFIG_SCRATCH: A pointer an 8-byte ** aligned memory buffer from which the scratch allocations will be ** drawn, the size of each scratch allocation (sz), ** and the maximum number of scratch allocations (N).)^& The first argument must be a pointer to an 8-byte aligned buffer ** of at least sz*N bytes of memory. ** ^& SQLite will not use more than one scratch buffers per thread. ** ^& SQLite will never request a scratch buffer that is more than 6 ** times the database page size. ** ^If SQLite needs additional ** scratch memory beyond what is provided by this configuration option, then ** [sqlite3_malloc()] will be used to obtain the memory needed.<p> ** ^When the application provides any amount of scratch memory using ** SQLITE_CONFIG_SCRATCH, SQLite avoids unnecessary large ** [sqlite3_malloc|heap allocations]. ** This can help [Robson proof|prevent memory allocation failures] due to heap ** fragmentation in low-memory embedded systems. ** </dd> *** [[SQLITE_CONFIG_PAGECACHE]] <dt>SQLITE_CONFIG_PAGECACHE</dt> ** <dd> ^The

SQLITE_CONFIG_PAGECACHE option specifies a memory pool ** that SQLite can use for the database page cache with the default page ** cache implementation. ** This configuration option is a no-op if an application-define page ** cache implementation is loaded using the [SQLITE_CONFIG_PCACHE2]. ** ^There are three arguments to SQLITE_CONFIG_PAGECACHE: A pointer to ** 8-byte aligned memory (pMem), the size of each page cache line (sz), ** and the number of cache lines (N). ** The sz argument should be the size of the largest database page ** (a power of two between 512 and 65536) plus some extra bytes for each ** page header. ^The number of extra bytes needed by the page header ** can be determined using [SQLITE_CONFIG_PCACHE_HDRSZ]. ** ^It is harmless, apart from the wasted memory, ** for the sz parameter to be larger than necessary. The pMem ** argument must be either a NULL pointer or a pointer to an 8-byte ** aligned block of memory of at least sz*N bytes, otherwise ** subsequent behavior is undefined. ** ^When pMem is not NULL, SQLite will strive to use the memory provided ** to satisfy page cache needs, falling back to [sqlite3_malloc()] if ** a page cache line is larger than sz bytes or if all of the pMem buffer ** is exhausted. ** ^If pMem is NULL and N is non-zero, then each database connection **

does an initial bulk allocation for page cache memory ** from [sqlite3_malloc()] sufficient for N cache lines if N is positive or ** of -1024*N bytes if N is negative. ^If additional ** page cache memory is needed beyond what is provided by the initial ** allocation, then SQLite goes to [sqlite3_malloc()] separately for each ** additional cache line. </dd> ** ** [[SQLITE_CONFIG_HEAP]] <dt>SQLITE_CONFIG_HEAP</dt> ** <dd> ^The SQLITE_CONFIG_HEAP option specifies a static memory buffer ** that SQLite will use for all of its dynamic memory allocation needs ** beyond those provided for by [SQLITE_CONFIG_SCRATCH] and ** [[SQLITE_CONFIG_PAGECACHE]]. ** ^The SQLITE_CONFIG_HEAP option is only available if SQLite is compiled ** with either [SQLITE_ENABLE_MEMSYS3] or [SQLITE_ENABLE_MEMSYS5] and returns ** [[SQLITE_ERROR]] if invoked otherwise. ** ^There are three arguments to SQLITE_CONFIG_HEAP: ** An 8-byte aligned pointer to the memory, ** the number of bytes in the memory buffer, and the minimum allocation size. ** ^If the first pointer (the memory pointer) is NULL, then SQLite reverts ** to using its default memory allocator (the system malloc() implementation), ** undoing any prior invocation of [SQLITE_CONFIG_MALLOC]. ^If the ** memory pointer is not NULL then the alternative memory ** allocator is engaged to handle all of SQLites memory allocation needs. ** The first pointer (the memory pointer) must be aligned to an 8-byte ** boundary or subsequent behavior of SQLite will be undefined. ** The minimum allocation size is capped at 2**12. Reasonable values ** for the minimum allocation size are 2**5 through 2**8. </dd> ** ** [[SQLITE_CONFIG_MUTEX]] <dt>SQLITE_CONFIG_MUTEX</dt> ** <dd> ^The SQLITE_CONFIG_MUTEX option takes a single argument which is a ** pointer to an instance of the [sqlite3_mutex_methods] structure. ** The argument specifies alternative low-level mutex routines to be used ** in place of the mutex routines built into SQLite. ^SQLite makes a copy of ** the content of the [sqlite3_mutex_methods] structure before the call to ** [sqlite3_config()] returns. ^If SQLite is compiled with ** the [SQLITE_THREADSAFE | SQLITE_THREADS=0] compile-time option then ** the entire mutexing subsystem is omitted from the build and hence calls to ** [sqlite3_config()] with the SQLITE_CONFIG_MUTEX configuration option will ** return [SQLITE_ERROR]. </dd> ** ** [[SQLITE_CONFIG_GETMUTEX]] <dt>SQLITE_CONFIG_GETMUTEX</dt> ** <dd> ^The SQLITE_CONFIG_GETMUTEX option takes a single argument which ** is a pointer to an instance of the [sqlite3_mutex_methods] structure. The ** [sqlite3_mutex_methods] ** structure is filled with the currently defined mutex routines.)^ ** This option can be used to overload the default mutex allocation ** routines with a wrapper used to track mutex usage for performance ** profiling or testing, for example. ^If SQLite is compiled with ** the [SQLITE_THREADSAFE | SQLITE_THREADS=0] compile-time option then ** the entire mutexing subsystem is omitted from the build and hence calls to ** [sqlite3_config()] with the SQLITE_CONFIG_GETMUTEX configuration option will ** return [SQLITE_ERROR]. </dd> ** ** [[SQLITE_CONFIG_LOOKASIDE]] <dt>SQLITE_CONFIG_LOOKASIDE</dt> ** <dd> ^The SQLITE_CONFIG_LOOKASIDE option takes two arguments that determine ** the default size of lookaside memory on each [database connection]. ** The first argument is the ** size of each lookaside buffer slot and the second is the number of ** slots allocated to each database connection.)^ ((SQLITE_CONFIG_LOOKASIDE ** sets the <i>default</i> lookaside size. The [SQLITE_DBCONFIG_LOOKASIDE] ** option to [sqlite3_db_config()] can be used to change the lookaside ** configuration on individual connections.)^ </dd> ** ** [[SQLITE_CONFIG_PCACHE2]] <dt>SQLITE_CONFIG_PCACHE2</dt> ** <dd> ^The SQLITE_CONFIG_PCACHE2 option takes a single argument which is ** a pointer to an [sqlite3_pcach_methods2] object. This object specifies ** the interface to a custom page cache implementation.)^ ** ^SQLite makes a copy of the [sqlite3_pcach_methods2] object. </dd> ** ** [[SQLITE_CONFIG_GETPCACHE2]] <dt>SQLITE_CONFIG_GETPCACHE2</dt> ** <dd> ^The SQLITE_CONFIG_GETPCACHE2 option takes a single argument which ** is a pointer to an [sqlite3_pcach_methods2] object. SQLite copies of ** the current page cache implementation into that object.)^ </dd> ** ** [[SQLITE_CONFIG_LOG]] <dt>SQLITE_CONFIG_LOG</dt> ** <dd> The SQLITE_CONFIG_LOG option is used to configure the SQLite ** global [error log]. ** (^The SQLITE_CONFIG_LOG option takes two arguments: a pointer to a ** function with a call signature of void(*)(void*,int,const char*), ** and a pointer to void. ^If the function pointer is not NULL, it is ** invoked by [sqlite3_log()] to process each logging event. ^If the ** function pointer is NULL, the [sqlite3_log()] interface becomes a no-op. ** ^The void pointer that is the second argument to SQLITE_CONFIG_LOG is ** passed through as the first parameter to the application-defined logger ** function whenever that function is invoked. ^The second parameter to ** the logger function is a copy of the first parameter to the corresponding ** [sqlite3_log()] call and is intended to be a [result code] or an ** [extended result code]. ^The third parameter passed to the logger is ** log message after formatting via [sqlite3_snprintf()]. ** The SQLite logging interface is not reentrant; the logger function ** supplied by the application must not invoke any SQLite interface. ** In a multi-threaded application, the application-defined logger ** function must be threadsafe. </dd> ** ** [[SQLITE_CONFIG_URI]] <dt>SQLITE_CONFIG_URI ** <dd> ^The SQLITE_CONFIG_URI option takes a single argument of type int. ** If non-zero, then URI handling is globally enabled. If the parameter is zero, ** then URI handling is globally disabled.)^ ^If URI handling is globally ** enabled, all filenames passed to [sqlite3_open()], [sqlite3_open_v2()], ** [sqlite3_open16()] or ** specified as part of [ATTACH] commands are interpreted as URIs, regardless ** of whether or not the [SQLITE_OPEN_URI] flag is set when the database ** connection is opened. ^If it is globally disabled, filenames are ** only interpreted as URIs if the SQLITE_OPEN_URI flag is set when the ** database connection is opened. ^((By default, URI handling is globally ** disabled. The default value may be changed by compiling with the ** [SQLITE_USE_URI] symbol defined.)^ ** ** [[SQLITE_CONFIG_COVERING_INDEX_SCAN]] <dt>SQLITE_CONFIG_COVERING_INDEX_SCAN ** <dd> ^The SQLITE_CONFIG_COVERING_INDEX_SCAN option takes a single integer ** argument which is interpreted as a boolean in order to enable or disable ** the use of covering indices for full table scans in the query optimizer. ** ^The default setting is determined ** by the [SQLITE_ALLOW_COVERING_INDEX_SCAN] compile-time option, or is "on" ** if that compile-time option is omitted. ** The ability to disable the use of covering indices for full table scans ** is because some incorrectly coded legacy applications might malfunction ** when the optimization is enabled. Providing the ability to ** disable the optimization allows the older, buggy application code to work ** without change even with newer versions of SQLite. ** ** [[SQLITE_CONFIG_PCACHE]] [[SQLITE_CONFIG_GETPCACHE]] ** <dt>SQLITE_CONFIG_PCACHE and SQLITE_CONFIG_GETPCACHE ** <dd> These options are obsolete and should not be used by new code. ** They are retained for backwards compatibility but are now no-ops. ** </dd> ** ** [[SQLITE_CONFIG_SQLLOG]] ** <dt>SQLITE_CONFIG_SQLLOG ** <dd> This option is only available if sqlite is compiled with the ** [SQLITE_ENABLE_SQLLOG] pre-processor macro defined. The first argument should ** be a pointer to a function of type void(*)(void*,sqlite3*,const char*, int). ** The second should be of type (void*). The callback is invoked by the library ** in three separate circumstances, identified by the value passed as the ** fourth parameter. If the fourth parameter is 0, then the database connection ** passed as the second argument has just been opened. The third argument ** points to a buffer containing the name of the main database file. If the ** fourth parameter is 1, then the SQL statement that the third parameter ** points to has just been executed. Or, if the fourth parameter is 2, then ** the connection being passed as the second parameter is being closed. The ** third parameter is passed NULL in this case. An example of using this ** configuration option can be seen in the "test_sqllog.c" source file in ** the canonical SQLite source tree. </dd> ** ** [[SQLITE_CONFIG_MMAP_SIZE]] ** <dt>SQLITE_CONFIG_MMAP_SIZE ** <dd> ^SQLITE_CONFIG_MMAP_SIZE takes two 64-bit integer (sqlite3_int64) values ** that are the default mmap size limit (the default setting for ** [PRAGMA mmap_size]) and the maximum allowed mmap size limit. ** ^The default setting can be overridden by each database connection using ** either the [PRAGMA mmap_size] command, or by using the ** [SQLITE_FCNTL_MMAP_SIZE] file control. ^((The maximum allowed mmap size ** will be silently truncated if necessary so that it does not exceed the ** compile-time maximum mmap size set by the ** [SQLITE_MAX_MMAP_SIZE] compile-time option.)^ ** ^If either argument to this option is negative, then that argument is ** changed to its compile-time default. ** ** [[SQLITE_CONFIG_WIN32_HEAPSIZE]] ** <dt>SQLITE_CONFIG_WIN32_HEAPSIZE ** <dd> ^The SQLITE_CONFIG_WIN32_HEAPSIZE option is only available if SQLite is ** compiled for Windows with the [SQLITE_WIN32_MALLOC] pre-processor macro ** defined. ^SQLITE_CONFIG_WIN32_HEAPSIZE takes a 32-bit unsigned integer value ** that specifies the maximum size of the created heap. ** ** [[SQLITE_CONFIG_PCACHE_HDRSZ]] ** <dt>SQLITE_CONFIG_PCACHE_HDRSZ ** <dd> ^The SQLITE_CONFIG_PCACHE_HDRSZ option takes a single parameter which ** is a pointer to an integer and writes into that integer the number of extra ** bytes per page required for each page in [SQLITE_CONFIG_PAGECACHE]. ** The amount of extra space required can change depending on the compiler, ** target platform, and SQLite version. ** ** [[SQLITE_CONFIG_PMASZ]] ** <dt>SQLITE_CONFIG_PMASZ ** <dd> ^The SQLITE_CONFIG_PMASZ option takes a single parameter which ** is an unsigned integer and sets the "Minimum PMA Size" for the multithreaded ** sorter to that integer. The default minimum PMA Size is set by the ** [SQLITE_SORTER_PMASZ] compile-time option. New threads are launched ** to help with sort operations when multithreaded sorting ** is enabled (using the [PRAGMA threads] command) and the amount of content ** to be sorted exceeds the page size times the minimum of the ** [PRAGMA cache_size] setting and this value. ** ** [[SQLITE_CONFIG_STMTJRN_SPILL]] ** <dt>SQLITE_CONFIG_STMTJRN_SPILL ** <dd> ^The SQLITE_CONFIG_STMTJRN_SPILL option takes a single parameter which ** becomes the [statement journal] spill-to-disk threshold. ** [Statement journals] are held in memory until their size (in bytes) ** exceeds this threshold, at which point they are written to disk. ** Or if the threshold is -1, statement journals are always held ** exclusively in memory. ** Since many statement journals never become large, setting the spill ** threshold to a value such as 64KiB can greatly reduce the amount of ** I/O required to support statement rollback. ** The default value for this setting is controlled by the ** [SQLITE_STMTJRN_SPILL] compile-time option. ** </dd>

label: code-design

61154. One of the uses of pBt->pTmpSpace is to format cells before ** inserting them into a leaf page (function fillInCell()). If ** a cell is less than 4 bytes in size, it is rounded up to 4 bytes ** by the various routines that manipulate binary cells. Which ** can mean that fillInCell() only initializes the first 2 or 3 ** bytes of pTmpSpace, but that the first 4 bytes are copied from ** it into a database page. This is not actually a problem, but it ** does cause a valgrind error when the 1 or 2 bytes of uninitialized ** data is passed to system call write(). So to avoid this error, ** zero the first 4 bytes of temp space here. ** ** Also: Provide four bytes of initialized space before the ** beginning of pTmpSpace as an area available to prepend the ** left-child pointer to the beginning of a cell.

61155. ** Trigger the alarm

61156. ** Invoke a virtual table constructor (either xCreate or xConnect). The ** pointer to the function to invoke is passed as the fourth parameter ** to this procedure.

61157. !defined(SQLITE_CORE) && !defined(SQLITE OMIT_LOAD_EXTENSION)

61158. Ticket #2804: When we open a database in the newer file format, ** clear the legacy_file_format pragma flag so that a VACUUM will ** not downgrade the database and thus invalidate any descending ** indices that the user might have created.

61159. MATCH => ID

61160. ** Insert a record into the %_segdir table.

61161. 310

61162. Knuth multiplicative hashing. (Sorting & Searching, p. 510). ** 0x9e3779b1 is 2654435761 which is the closest prime number to ** (2**32)*golden_ratio, where golden_ratio = (sqrt(5) - 1)/2.

61163. **comment:** TODO: Check if all of these are really required.
label: requirement

61164. The journal file needs to be opened. Higher level routines have already ** obtained the necessary locks to begin the write-transaction, but the ** rollback journal might not yet be open. Open it now if this is the case. ** ** This is done before calling sqlite3PcacheMakeDirty() on the page. ** Otherwise, if it were done after calling sqlite3PcacheMakeDirty(), then ** an error might occur and the pager would end up in WRITER_LOCKED state ** with pages marked as dirty in the cache.

61165. Location of new mapping

61166. Size information about the cell

61167. ** Register a collation sequence factory callback with the database handle ** db. Replace any previously installed collation sequence factory.

61168. Write the number of columns here

61169. 38..3f 89:;<=>?

61170. (288) nm ::= JOIN_KW

61171. True if range start uses ==, >= or <=

61172. **comment:** ** The Pager.eLock variable is almost always set to one of the ** following locking-states, according to the lock currently held on ** the database file: NO_LOCK, SHARED_LOCK, RESERVED_LOCK or EXCLUSIVE_LOCK. ** This variable is kept up to date as locks are taken and released by ** the pagerLockDb() and pagerUnlockDb() wrappers. ** ** If the VFS xLock() or xUnlock() returns an error other than SQLITE_BUSY ** (i.e. one of the SQLITE_IOERR subtypes), it is not clear whether or not ** the operation was successful. In these circumstances pagerLockDb() and ** pagerUnlockDb() take a conservative approach - eLock is always updated ** when unlocking the file, and only updated when locking the file if the ** VFS call is successful. This way, the Pager.eLock variable may be set ** to a less exclusive (lower) value than the lock that is actually held ** at the system level, but it is never set to a more exclusive value. ** ** This is usually safe. If an xUnlock fails or appears to fail, there may ** be a few redundant xLock() calls or a lock may be held for longer than ** required, but nothing really goes wrong. ** ** The exception is when the database file is unlocked as the pager moves ** from ERROR to OPEN state. At this point there may be a hot-journal file ** in the file-system that needs to be rolled back (as part of an OPEN->SHARED ** transition, by the same pager or any other). If the call to xUnlock() ** fails at this point and the pager is left holding an EXCLUSIVE lock, this ** can confuse the call to xCheckReservedLock() call made later as part ** of hot-journal detection. ** ** xCheckReservedLock() is defined as returning true "if there is a RESERVED ** lock held by this process or any others". So xCheckReservedLock may ** return true because the caller itself is holding an EXCLUSIVE lock (but ** doesn't know it because of a previous error in xUnlock). If this happens ** a hot-journal may be mistaken for a journal being created by an active ** transaction in another process, causing SQLite to read from the database ** without rolling it back. ** ** To work around this, if a call to xUnlock() fails when unlocking the ** database in the ERROR state, Pager.eLock is set to UNKNOWN_LOCK. It ** is only changed back to a real locking state after a successful call ** to xLock(EXCLUSIVE). Also, the code to do the OPEN->SHARED state transition ** omits the check for a hot-journal if Pager.eLock is set to UNKNOWN_LOCK ** lock. Instead, it assumes a hot-journal exists and obtains an EXCLUSIVE ** lock on the database file before attempting to roll it back. See function ** PagerSharedLock() for more detail. ** ** Pager.eLock may only be set to UNKNOWN_LOCK when the pager is in ** PAGER_OPEN state.
label: code-design

61173. Begin the database scan.

61174. This routine has been called to create an automatic index as a ** result of a PRIMARY KEY or UNIQUE clause on a column definition, or ** a PRIMARY KEY or UNIQUE clause following the column definitions. ** i.e. one of: ** ** CREATE TABLE t(x PRIMARY KEY, y); ** CREATE TABLE t(x, y, UNIQUE(x, y)); ** ** Either way, check to see if the table already has such an index. If ** so, don't bother creating this one. This only applies to ** automatically created indices. Users can do as they wish with ** explicit indices. ** ** Two UNIQUE or PRIMARY KEY constraints are considered equivalent ** (and thus suppressing the second one) even if they have different ** sort orders. ** ** If there are different collating sequences or if the columns of ** the constraint occur in different orders, then the constraints are ** considered distinct and both result in separate indices.

61175. Message from strerror() or equivalent

61176. ** Increase the reference count of a supplied page by 1.

61177. If the file is a temp-file has not yet been opened, open it now. It ** is not possible for rc to be other than SQLITE_OK if this branch ** is taken, as pager_wait_on_lock() is a no-op for temp-files.

61178. 10, 9, 8, 7, 6

61179. Dequoted name of collation sequence

61180. ** Implementation of the total_changes() SQL function. The return value is ** the same as the sqlite3_total_changes() API function.

61181. Number of bytes in a page

61182. List of tables in the join

61183. xConnect

61184. Values between 0 and 127

61185. ** 2009 Nov 12 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

61186. id that may be used to recursive triggers

61187. **comment:** A malloc() may have failed within the call to sqlite3_value_text16() ** above. If this is the case, then the db->mallocFailed flag needs to ** be cleared before returning. Do this directly, instead of via ** sqlite3ApiExit(), to avoid setting the database handle error message.
label: code-design

61188. Number of winShm objects pointing to this

61189. ** Change the default pages size and the number of reserved bytes per page. ** Or, if the page size has already been fixed, return SQLITE_READONLY ** without changing anything. ** ** The page size must be a power of 2 between 512 and 65536. If the page ** size supplied does not meet this constraint then the page size is not ** changed. ** ** Page sizes are constrained to be a power of two so that the region ** of the database file used for locking (beginning at PENDING_BYTE, ** the first byte past the 1GB boundary, 0x40000000) needs to occur ** at the beginning of a page. ** ** If parameter nReserve is less than zero, then the number of reserved ** bytes per page is left unchanged. ** ** If the iFix!=0 then the BTS_PAGESIZE_FIXED flag is set so that the page size ** and autovacuum mode can no longer be changed.

61190. Output variables

61191. At this point, if the literal was an integer, the parse is ** finished. Or, if it is a floating point value, it may continue ** with either a decimal point or an 'E' character.

61192. ***** End of fts5.h *****

61193. Set the foreign key value to its default

61194. **comment:** ** SUMMARY ** ** Writing a transaction containing a large number of operations on ** b-tree indexes that are collectively larger than the available cache ** memory can be very inefficient. ** ** The problem is that in order to update a b-tree, the leaf page (at least) ** containing the entry being inserted or deleted must be modified. If the ** working set of leaves is larger than the available cache memory, then a ** single leaf that is modified more than once as part of the transaction ** may be loaded from or written to the persistent media multiple times. ** Additionally, because the index updates are likely to be applied in ** random order, access to pages within the database is also likely to be in ** random order, which is itself quite inefficient. ** ** One way to improve the situation is to sort the operations on each index ** by index key before applying them to the b-tree. This leads to an IO ** pattern that resembles a single linear scan through the index b-tree, ** and all but guarantees each modified leaf page is loaded and stored ** exactly once. SQLite uses this trick to improve the performance of ** CREATE INDEX commands. This extension allows it to be used to improve ** the performance of large transactions on existing databases. ** ** Additionally, this extension allows the work involved in writing the ** large transaction to be broken down into sub-transactions performed ** sequentially by separate processes. This is useful if the system cannot ** guarantee that a single update process will run for long enough to apply ** the entire update, for example because the update is being applied on a ** mobile device that is frequently rebooted. Even after the writer process ** has committed one or more sub-transactions, other

database clients continue ** to read from the original database snapshot. In other words, partially ** applied transactions are not visible to other clients. *** *** "RBU" stands for "Resumable Bulk Update". As in a large database update ** transmitted via a wireless network to a mobile device. A transaction ** applied using this extension is hence referred to as an "RBU update". *** *** LIMITATIONS *** An "RBU update" transaction is subject to the following limitations: *** *** The transaction must consist of INSERT, UPDATE and DELETE operations ** only. *** *** INSERT statements may not use any default values. *** *** * UPDATE and DELETE statements must identify their target rows by ** non-NULL PRIMARY KEY values. Rows with NULL values stored in PRIMARY KEY fields may not be updated or deleted. If the table being written ** has no PRIMARY KEY, affected rows must be identified by rowid. *** *** * UPDATE statements may not modify PRIMARY KEY columns. *** *** No triggers will be fired. *** *** No foreign key violations are detected or reported. *** *** CHECK constraints are not enforced. *** *** No constraint handling mode except for "OR ROLLBACK" is supported. *** *** PREPARATION *** An "RBU update" is stored as a separate SQLite database. A database ** containing an RBU update is an "RBU database". For each table in the ** target database to be updated, the RBU database should contain a table ** named "data_<target name>" containing the same set of columns as the ** target table, and one more - "rbu_control". The data_% table should ** have no PRIMARY KEY or UNIQUE constraints, but each column should have ** the same type as the corresponding column in the target database. ** The "rbu_control" column should have no type at all. For example, if ** the target database contains: *** *** CREATE TABLE t1(a INTEGER PRIMARY KEY, b TEXT, c UNIQUE); *** Then the RBU database should contain: *** *** CREATE TABLE data_t1(a INTEGER, b TEXT, c, rbu_control); *** The order of the columns in the data_% table does not matter. *** Instead of a regular table, the RBU database may also contain virtual ** tables or view named using the data_<target> naming scheme. *** Instead of the plain data_<target> naming scheme, RBU database tables ** may also be named data<integer>_<target>, where <integer> is any sequence ** of zero or more numeric characters (0-9). This can be significant because ** tables within the RBU database are always processed in order sorted by ** name. By judicious selection of the <integer> portion of the names ** of the RBU tables the user can therefore control the order in which they ** are processed. This can be useful, for example, to ensure that "external *** content" FTS4 tables are updated before their underlying content tables. *** If the target database table is a virtual table or a table that has no ** PRIMARY KEY declaration, the data_% table must also contain a column ** named "rbu_rowid". This column is mapped to the tables implicit primary ** key column - "rowid". Virtual tables for which the "rowid" column does ** not function like a primary key value cannot be updated using RBU. For ** example, if the target db contains either of the following: *** *** CREATE VIRTUAL TABLE x1 USING fts3(a, b); ** CREATE TABLE x1(a, b) ** ** then the RBU database should contain: *** *** CREATE TABLE data_x1(a, b, rbu_rowid, rbu_control); *** All non-hidden columns (i.e. all columns matched by "SELECT *") of the ** target table must be present in the input table. For virtual tables, ** hidden columns are optional - they are updated by RBU if present in ** the input table, or not otherwise. For example, to write to an fts4 ** table with a hidden languageid column such as: *** *** CREATE VIRTUAL TABLE ft1 USING fts4(a, b, languageid='langid'); *** Either of the following input table schemas may be used: *** *** CREATE TABLE data_ft1(a, b, langid, rbu_rowid, rbu_control); ** CREATE TABLE data_ft1(a, b, rbu_rowid, rbu_control); *** *** For each row to INSERT into the target database as part of the RBU ** update, the corresponding data_% table should contain a single record ** with the "rbu_control" column set to contain integer value 0. The ** other columns should be set to the values that make up the new record ** to insert. *** *** If the target database table has an INTEGER PRIMARY KEY, it is not ** possible to insert a NULL value into the IPK column. Attempting to ** do so results in an SQLITE_MISMATCH error. *** For each row to DELETE from the target database as part of the RBU ** update, the corresponding data_% table should contain a single record ** with the "rbu_control" column set to contain integer value 1. The ** real primary key values of the row to delete should be stored in the ** corresponding columns of the data_% table. The values stored in the ** other columns are not used. *** *** For each row to UPDATE from the target database as part of the RBU ** update, the corresponding data_% table should contain a single record ** with the "rbu_control" column set to contain a value of type text. ** The real primary key values identifying the row to update should be ** stored in the corresponding columns of the data_% table row, as should ** the new values of all columns being update. The text value in the ** "rbu_control" column must contain the same number of characters as ** there are columns in the target database table, and must consist entirely ** of 'x' and '.' characters (or in some special cases 'd' - see below). For ** each column that is being updated, the corresponding character is set to ** 'x'. For those that remain as they are, the corresponding character of the ** rbu_control value should be set to '..'. For example, given the tables ** above, the update statement: *** *** UPDATE t1 SET c = 'usa' WHERE a = 4; *** is represented by the data_t1 row created by: *** *** INSERT INTO data_t1(a, b, c, rbu_control) VALUES(4, NULL, 'usa', '.x'); *** Instead of an 'x' character, characters of the rbu_control value specified ** for UPDATES may also be set to 'd'. In this case, instead of updating the ** target table with the value stored in the corresponding data_% column, the ** user-defined SQL function "rbu_delta()" is invoked and the result stored in ** the target table column. rbu_delta() is invoked with two arguments - the ** original value currently stored in the target table column and the ** value specified in the data_xxx table. *** *** For example, this row: *** *** INSERT INTO data_t1(a, b, c, rbu_control) VALUES(4, NULL, 'usa', '..d'); *** is similar to an UPDATE statement such as: *** *** UPDATE t1 SET c = rbu_delta(c, 'usa') WHERE a = 4; *** *** Finally, if an 'f' character appears in place of a 'd' or 's' in an ** ota_control string, the contents of the data_xxx table column is assumed ** to be a "fossil delta" - a patch to be applied to a blob value in the ** format used by the fossil source-code management system. In this case ** the existing value within the target database table must be of type BLOB. ** It is replaced by the result of applying the specified fossil delta to ** itself. *** *** If the target database table is a virtual table or a table with no PRIMARY ** KEY, the rbu_control value should not include a character corresponding ** to the rbu_rowid value. For example, this: *** *** INSERT INTO data_ft1(a, b, rbu_rowid, rbu_control) ** VALUES(NULL, 'usa', 12, '.x'); *** causes a result similar to: *** *** UPDATE ft1 SET b = 'usa' WHERE rowid = 12; *** *** The data_xxx tables themselves should have no PRIMARY KEY declarations. ** However, RBU is more efficient if reading the rows in from each data_xxx ** table in "rowid" order is roughly the same as reading them sorted by ** the PRIMARY KEY of the corresponding target database table. In other ** words, rows should be sorted using the destination table PRIMARY KEY ** fields before they are inserted into the data_xxx tables. *** *** USAGE *** The API declared below allows an application to apply an RBU update ** stored on disk to an existing target database. Essentially, the ** application: *** *** 1) Opens an RBU handle using the sqlite3rbu_open() function. *** *** 2) Registers any required virtual table modules with the database ** handle returned by sqlite3rbu_db(). Also, if required, register ** the rbu_delta() implementation. *** *** 3) Calls the sqlite3rbu_step() function one or more times on ** the new handle. Each call to sqlite3rbu_step() performs a single ** b-tree operation, so thousands of calls may be required to apply ** a complete update. *** *** 4) Calls sqlite3rbu_close() to close the RBU update handle. If ** sqlite3rbu_step() has been called enough times to completely ** apply the update to the target database, then the RBU database ** is marked as fully applied. Otherwise, the state of the RBU ** update application is saved in the RBU database for later ** resumption. *** *** See comments below for more detail on APIs. *** *** If an update is only partially applied to the target database by the ** time sqlite3rbu_close() is called, various state information is saved ** within the RBU database. This allows subsequent processes to automatically ** resume the RBU update from where it left off. *** *** To remove all RBU extension state information, returning an RBU database ** to its original contents, it is sufficient to drop all tables that begin ** with the prefix "rbu_" *** *** DATABASE LOCKING *** *** An RBU update may not be applied to a database in WAL mode. Attempting ** to do so is an error (SQLITE_ERROR). *** *** While an RBU handle is open, a SHARED lock may be held on the target ** database file. This means it is possible for other clients to read the ** database, but not to write it. *** *** If an RBU update is started and then suspended before it is completed, ** then an external client writes to the database, then attempting to resume ** the suspended RBU update is also an error (SQLITE_BUSY).

label: code-design

61195. ** An instance of the following object is used to read records out of a ** PMA, in sorted order. The next key to be read is cached in nKey/aKey. ** aKey might point into aMap or into aBuffer. If neither of those locations ** contain a contiguous representation of the key, then aAlloc is allocated ** and the key is copied into aAlloc and aKey is made to point to aAlloc. *** *** pFd==0 at EOF.
61196. ANALYZE => ID
61197. ** Return meta information about a specific column of a database table. ** See comment in sqlite3.h (sqlite.h.in) for details.
61198. Naming context
61199. Offset within poslist
61200. Do not attempt to process this query if there are any WITH clauses ** attached to it. Proceeding may generate a false "no such table: xxx" ** error if pSelect reads from a CTE named "xxx".
61201. Generate code that runs whenever the GROUP BY changes. ** Changes in the GROUP BY are detected by the previous code ** block. If there were no changes, this block is skipped. *** *** This code copies current group by terms in b0,b1,b2,... ** over to a0,a1,a2. It then calls the output subroutine ** and resets the aggregate accumulator registers in preparation ** for the next GROUP BY batch.
61202. Number of elements in a bitmap array.
61203. Configured "cache_size" value
61204. **comment:** Comment to improve readability
- label:** code-design
61205. Jump to here if a malloc() fails.
61206. ** CAPI3REF: Custom Page Cache Object ** ** The sqlite3_pcache_page object represents a single page in the ** page cache. The page cache will allocate instances of this ** object. Various methods of the page cache use pointers to instances ** of this object as parameters or as their return value. *** *** See [sqlite3_pcache_methods2] for additional information.
61207. Check the integrity of the freelist
61208. **comment:** Bytes of opcode memory used
- label:** code-design

61209. Column into which text is being inserted
61210. We have already reached the end of this doclist. EOF.
61211. LIKE
61212. Information used during initialization
61213. The WHERE clause. May be null
61214. Configuration object to update
61215. Find the root of the NEAR expression
61216. Number of instructions in the program
61217. Usable bytes per page
61218. Check that zKey/nKey is larger than the largest key the candidate
61219. SQLITE_EMPTY
61220. ** This routine is a Walker callback for "expanding" a SELECT statement. ** "Expanding" means to do the following: *** (1) Make sure Vdbe cursor numbers have been assigned to every ** element of the FROM clause. *** (2) Fill in the pTabList->a[],pTab fields in the SrcList that ** defines FROM clause. When views appear in the FROM clause, ** fill pTabList->a[],pSelect with a copy of the SELECT statement ** that implements the view. A copy is made of the view's SELECT ** statement so that we can freely modify or delete that statement ** without worrying about messing up the persistent representation ** of the view. ** *** (3) Add terms to the WHERE clause to accommodate the NATURAL keyword ** on joins and the ON and USING clause of joins. *** (4) Scan the list of columns in the result set (pEList) looking ** for instances of the "*" operator or the TABLE.* operator. ** If found, expand each "*" to be every column in every table ** and TABLE.* to be every column in TABLE. **
61221. The table from which rows are deleted
61222. If the write version is set to 2, this database should be accessed ** in WAL mode. If the log is not already open, open it now. Then ** return SQLITE_OK and return without populating BtShared.pPage1. ** The caller detects this and calls this function again. This is ** required as the version of page 1 currently in the page1 buffer ** may not be the latest version - there may be a newer one in the log ** file.
61223. ** Invoke the destructor function associated with FuncDef p, if any. Except, ** if this is not the last copy of the function, do not invoke it. Multiple ** copies of a single function are created when create_function() is called ** with SQLITE_ANY as the encoding.
61224. This is an optimization only. The call to sqlite3Stat4ProbeSetValue() ** below would return the same value.
61225. A-overwrites-B
61226. ** This function is called from within a pre-update callback to retrieve ** the number of columns in the row being updated, deleted or inserted.
61227. Handle to return
61228. xRandomness
61229. Context pointer for xCreate/xConnect
61230. Delete the index and table entries. Skip this step if pTab is really ** a view (in which case the only effect of the DELETE statement is to ** fire the INSTEAD OF triggers). *** If variable 'count' is non-zero, then this OP_Delete instruction should ** invoke the update-hook. The pre-update-hook, on the other hand should ** be invoked unless table pTab is a system table. The difference is that ** the update-hook is not invoked for rows removed by REPLACE, but the ** pre-update-hook is.
61231. ** This function is only called from with a pre-update-hook reporting a ** change on table pTab (attached to session pSession). The type of change ** (UPDATE, INSERT, DELETE) is specified by the first argument. ** ** Unless one is already present or an error occurs, an entry is added ** to the changed-rows hash table associated with table pTab.
61232. ** This function is called as part of initializing or reinitializing an ** incremental checkpoint. *** It populates the sqlite3rbu.aFrame[] array with the set of ** (wal frame -> db page) copy operations required to checkpoint the ** current wal file, and obtains the set of shm locks required to safely ** perform the copy operations directly on the file-system. *** If argument pState is not NULL, then the incremental checkpoint is ** being resumed. In this case, if the checksum of the wal-index-header ** following recovery is not the same as the checksum saved in the RbuState ** object, then the rru handle is set to DONE state. This occurs if some ** other client appends a transaction to the wal file in the middle of ** an incremental checkpoint.
61233. Value to return if (lhs > rhs)
61234. ** This routine generates the code for the inside of the inner loop ** of a SELECT. *** If srcTab is negative, then the pEList expressions ** are evaluated in order to get the data for this row. If srcTab is ** zero or more, then data is pulled from srcTab and pEList is used only ** to get the number of columns and the collation sequence for each column.
61235. A-overwrites-Y
61236. stmt_vtab is a subclass of sqlite3_vtab which will ** serve as the underlying representation of a stmt virtual table
61237. Generate a coroutine to evaluate the SELECT statement to the ** left of the compound operator - the "A" select.
61238. ** The xUpdate method for rtree module virtual tables.
61239. **comment:** Space for aMem[0] even if not used
 label: code-design
61240. ifndef NDEBUG
61241. ***** Begin file rowset.c *****
61242. 3
61243. 0x40 .. 0x4F
61244. ** Make sure the cell sizes at idx, idx+1, ..., idx+N-1 have been ** computed.
61245. If the file should be created now, create it and write the new data ** into the file on disk.
61246. One or more aggregate functions seen
61247. **comment:** If there is no IDLIST term but the table has an integer primary ** key, the set the ipkColumn variable to the integer primary key ** column index in the original table definition.
 label: code-design
61248. 1560
61249. vfs module to open wal and wal-index
61250. Number of bytes still to come
61251. OUT: Array of column names for table
61252. All information about this parse
61253. ** Load pointers to all cells on sibling pages and the divider cells ** into the local b.apCell[] array. Make copies of the divider cells ** into space obtained from aSpace1[]. The divider cells have already ** been removed from pParent. *** If the siblings are on leaf pages, then the child pointers of the ** divider cells are stripped from the cells before they are copied ** into aSpace1[]. In this way, all cells in b.apCell[] are without ** child pointers. If siblings are not leaves, then all cell in ** b.apCell[] include child pointers. Either way, all cells in b.apCell[] ** are alike. *** leafCorrection: 4 if pPage is a leaf. 0 if pPage is not a leaf. ** leafData: 1 if pPage holds key+data and pParent holds only keys.
61254. ***** Begin file attach.c *****
61255. Return address register for output subroutine
61256. Deep, not shallow copies
61257. ** PRAGMA [schema.]max_page_count ** PRAGMA [schema.]max_page_count=N ** ** The first form reports the current setting for the ** maximum number of pages in the database file. The ** second form attempts to change this setting. Both ** forms return the current setting. *** The absolute value of N is used. This is undocumented and might ** change. The only purpose is to provide an easy way to test ** the sqlite3AbsInt32() function. *** PRAGMA [schema.]page_count ** ** Return the number of pages in the specified database.
61258. SQLITE_MIGHT_BE_SINGLE_CORE
61259. ** An instance of the following structure holds the context of a ** sum() or avg() aggregate computation.
61260. Name of column in zTo. If NULL use PRIMARY KEY
61261. Hash table of tokenizers
61262. ***** Begin file table.c *****
61263. Phrase to return doclist for
61264. ** Fts5SegIter.iLeafOffset currently points to the first byte of a ** position-list size field. Read the value of the field and store it ** in the following variables: ** ** Fts5SegIter.nPos ** Fts5SegIter.bDel ** ** Leave Fts5SegIter.iLeafOffset pointing to the first byte of the ** position list content (if any).
61265. ** The xRelease() method. *** This is a no-op.

61266. ** Let V[] be an array of unsigned characters sufficient to hold ** up to N bits. Let I be an integer between 0 and N. 0<=I<N. ** Then the following macros can be used to set, clear, or test ** individual bits within V.

61267. ** Attempt to set the maximum database page count if mxPage is positive. ** Make no changes if mxPage is zero or negative. And never reduce the ** maximum page count below the current size of the database. ** ** Regardless of mxPage, return the current maximum page count.

61268. ***** Continuing where we left off in os.h *****

61269. Used by: index_list

61270. ** Initialize PmaReader pReadr to scan through the PMA stored in file pFile ** starting at offset iStart and ending at offset iEof-1. This function ** leaves the PmaReader pointing to the first key in the PMA (or EOF if the ** PMA is empty). ** ** If the pnByte parameter is NULL, then it is assumed that the file ** contains a single PMA, and that that PMA omits the initial length varint.

61271. WhereLoop.u.vtab is valid

61272. If the inner loop is driven by an index such that values from ** the same iteration of the inner loop are in sorted order, then ** immediately jump to the next iteration of an inner loop if the ** entry from the current iteration does not fit into the top ** LIMIT+OFFSET entries of the sorter.

61273. 17

61274. The FROM clause

61275. Array of flags, set on indexed & PK cols

61276. ** This is a comparison function used as a qsort() callback when sorting ** an array of pending terms by term. This occurs as part of flushing ** the contents of the pending-terms hash table to the database.

61277. Ex

61278. ** All of the FuncDef structures in the aBuiltinFunc[] array above ** to the global function hash table. This occurs at start-time (as ** a consequence of calling sqlite3_initialize()). ** ** After this routine runs

61279. ** Table of methods for MemJournal sqlite3_file object.

61280. SAVEPOINT => ID

61281. The right pointer of the child page pOld becomes the left ** pointer of the divider cell

61282. OUT: Set to true if EOF

61283. Non-recursive mutex required to access this object

61284. Next token type

61285. Close all database connections

61286. Check that all non-HIDDEN columns in the destination table are also ** present in the input table. Populate the abTblPk[], azTblType[] and ** aiTblOrder[] arrays at the same time.

61287. SQLITE_NOTADB

61288. 2

61289. Failed to open the file for read/write access. Try read-only.

61290. get sign of exponent

61291. ** Return the value of a status counter for a prepared statement

61292. Pages journalled since last j-header written

61293. If the subquery is not correlated and if we are not inside of ** a trigger, then we only need to compute the value of the subquery ** once.

61294. Check to see if z[0..n-1] is a keyword. If it is, write the ** parser symbol code for that keyword into *pType. Always ** return the integer n (the length of the token).

61295. ** If the expression list pEList contains more than iLimit elements, ** leave an error message in pParse.

61296. ** 2009 Oct 23 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

61297. SQLITE_INTERNAL

61298. Maximum nColumn is BMS-2, not BMS-1, so that we can compute ** BITMASK(nExpr) without overflowing

61299. Unless the database is sharable and unlocked, then BtShared.db ** should already be set correctly.

61300. xBestIndex

61301. ** Class derived from sqlite3_tokenizer_cursor

61302. The binary operation

61303. Parser state when parsing URI

61304. SQLITE_BUSY? proxyTakeConch called during locking

61305. ** 2017-05-31 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file demonstrates an eponymous virtual table that returns information ** about all prepared statements for the database connection. ** ** Usage example: ** ** .load ./stmt ** .mode line ** .header on ** SELECT * FROM stmt;

61306. Initialize the count of updated rows

61307. Used by: wal_checkpoint

61308. Used to iterate through ATC[] array.

61309. Finally, sync the database file.

61310. ** Possible values for FuncDef.flags. Note that the _LENGTH and _TYPEOF ** values must correspond to OPFLAG_LENGTHHARG and OPFLAG_TYPEOFARG. And ** SQLITE_FUNC_CONSTANT must be the same as SQLITE_DETERMINISTIC. There ** are assert() statements in the code to verify this. ** ** Value constraints (enforced via assert()): ** SQLITE_FUNC_MINMAX == NC_MinMaxAgg == SF_MinMaxAgg ** SQLITE_FUNC_LENGTH == OPFLAG_LENGTHHARG ** SQLITE_FUNC_TYPEOF == OPFLAG_TYPEOFARG ** SQLITE_FUNC_CONSTANT == SQLITE_DETERMINISTIC from the API ** SQLITE_FUNC_ENCMASK depends on SQLITE_UTF* macros in the API

61311. ** The following macro converts a relative address in the p2 field ** of a VdbeOp structure into a negative number so that ** sqlite3VdbeAddOpList() knows that the address is relative. Calling ** the macro again restores the address.

61312. **comment:** we can skip this cause it was (effectively) done above in calc'ing s
label: code-design

61313. * Each trigger present in the database schema is stored as an instance of * struct Trigger. ** Pointers to instances of struct Trigger are stored in two ways. * 1. In the "trigHash" hash table (part of the sqlite3* that represents the * database). This allows Trigger structures to be retrieved by name. * 2. All triggers associated with a single table form a linked list, using the * pNext member of struct Trigger. A pointer to the first element of the * linked list is stored as the "pTrigger" member of the associated * struct Table. * * The "step_list" member points to the first element of a linked list * containing the SQL statements specified as the trigger program.

61314. Checksum of last frame in log

61315. ** Delete any previous value and set the value to be a BLOB of length ** n containing all zeros.

61316. Stat4Accum.aBest[], a[]

61317. True if p is a compound select

61318. ** Control and query of the open file handle.

61319. Bitmask of FROM clause terms referenced by pExpr

61320. Token with unqualified schema object name

61321. ** Increment the reference count of node p.

61322. The common case of ?N for a single digit N

61323. x<EXPR or x<=EXPR constraint

61324. List of VTable objects.

61325. Pointer to data for string (char array) types

61326. Use for looping over pSrcList items

61327. 0

61328. Cursor number of pseudo-table

61329. If changeCountDone is set, a RESERVED lock or greater must be held ** on the file.
61330. End of the 'extern "C"' block
61331. the table into which we are inserting
61332. Two salt values copied from WAL header
61333. A value was pulled from the index
61334. ** The implementation of the sqlite_record() function. This function accepts ** a single argument of any type. The return value is a formatted database ** record (a blob) containing the argument value. ** ** This is used to convert the value stored in the 'sample' column of the ** sqlite_stat3 table to the record format SQLite uses internally.
61335. ** These are definitions of bits in the WhereLoop.wsFlags field. ** The particular combination of bits in each WhereLoop help to ** determine the algorithm that WhereLoop represents.
61336. ** Convert a 64-bit IEEE double into a 64-bit signed integer. ** If the double is out of range of a 64-bit signed integer then ** return the closest available 64-bit signed integer.
61337. If this database is not shareable, or if the client is reading ** and has the read-uncommitted flag set, then no lock is required. ** Return true immediately.
61338. One of the PAGER_JOURNALMODE_XXX symbols
61339. Cursors that are not yet positioned
61340. unicode characters usable in IDs
61341. ** Lock the file with the lock specified by parameter eFileLock - one ** of the following: ** ** (1) SHARED_LOCK ** (2) RESERVED_LOCK ** (3) PENDING_LOCK ** (4) EXCLUSIVE_LOCK ** ** Sometimes when requesting one lock state, additional lock states ** are inserted in between. The locking might fail on one of the later ** transitions leaving the lock state different from what it started but ** still short of its goal. The following chart shows the allowed ** transitions and the inserted intermediate states: ** ** UNLOCKED -> SHARED ** SHARED -> RESERVED ** SHARED -> (PENDING) -> EXCLUSIVE ** RESERVED -> (PENDING) -> EXCLUSIVE ** PENDING -> EXCLUSIVE ** ** Semaphores locks only really support EXCLUSIVE locks. We track intermediate ** lock states in the sqlite3_file structure, but all locks SHARED or ** above are really EXCLUSIVE locks and exclude all other processes from ** access the file. ** ** This routine will only increase a lock. Use the sqlite3OsUnlock() ** routine to lower a locking level.
61342. Head of in-memory chunk-list
61343. Bytes of runtime space required for sub-program
61344. Next available unixShm.id value
61345. Allocate the mutex and register the new VFS (not as the default)
61346. Begin with this column of pList
61347. One character in z[]
61348. ** Structure used internally by this VFS to record the state of an ** open shared memory connection. ** ** The following fields are initialized when this object is created and ** are read-only thereafter: ** ** unixShm.pFile ** unixShm.id ** ** All other fields are read/write. The unixShm.pFile->mutex must be held ** while accessing any read/write fields.
61349. Error message for circular references
61350. ** 2013 November 25 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file contains code that is specific to Windows.
61351. Filename in UTF-8 encoding
61352. ** Delete a P4 value if necessary.
61353. Trigger to code
61354. Which read lock is being held. -1 for none
61355. Set the abort flag and return
61356. #include "sqliteInt.h"
61357. Desired amount of reserved space per page
61358. DROP
61359. 297
61360. Iterator handle
61361. Name of the database file
61362. List of tables in FROM clause
61363. **comment:** The heap subsystem has now been shutdown and these values are supposed ** to be NULL or point to memory that was obtained from sqlite3_malloc(), ** which would rely on that heap subsystem; therefore, make sure these ** values cannot refer to heap memory that was just invalidated when the ** heap subsystem was shutdown. This is only done if the current call to ** this function resulted in the heap subsystem actually being shutdown.
label: code-design
61364. ** This routine is called after all of the trigger actions have been parsed ** in order to complete the process of building the trigger.
61365. This function contains the logic that determines if a statement or ** transaction will be committed or rolled back as a result of the ** execution of this virtual machine. ** ** If any of the following errors occur: ** ** SQLITE_NOMEM ** SQLITE_IOERR ** SQLITE_FULL ** SQLITE_INTERRUPT ** ** Then the internal cache might have been left in an inconsistent ** state. We need to rollback the statement transaction, if there is ** one, or the complete transaction if there is no statement transaction.
61366. Used for testing and analysis only
61367. Pointer to buffer containing token
61368. Stmt-journal spill-to-disk threshold
61369. Number of fields in test sample
61370. ** In this implementation, error checking is provided for testing ** and debugging purposes. The mutexes still do not provide any ** mutual exclusion.
61371. This routine is only called from the OP_JournalMode opcode, and ** the logic there will never allow a temporary file to be changed ** to WAL mode.
61372. ** Push an authorization context. After this routine is called, the ** zArg3 argument to authorization callbacks will be zContext until ** popped. Or if pParse==0, this routine is a no-op.
61373. Number of deferred imm fk.
61374. ** Analyze a term that consists of two or more OR-connected ** subterms. So in: ** ** ... WHERE (a=5) AND (b=7 OR c=9 OR d=13) AND (d=13) ** **** This routine analyzes terms such as the middle term in the above example. ** A WhereOrTerm object is computed and attached to the term under ** analysis, regardless of the outcome of the analysis. Hence: ** ** WhereTerm.wsFlags |= TERM_ORINFO ** WhereTerm.u.pOrInfo = a dynamically allocated WhereOrTerm object ** ** The term being analyzed must have two or more of OR-connected subterms. ** A single subterm might be a set of AND-connected sub-subterms. ** Examples of terms under analysis: ** ** (A) t1.x=2,y OR t1.x=t2.z OR t1.y=15 OR t1.z=t3.a+5 ** (B) x=expr1 OR expr2=x OR x=expr3 ** (C) t1.x=t2,y OR (t1.x=t2.z AND t1.y=15) ** (D) x=expr1 OR (y>11 AND y<22 AND z LIKE "hello*") ** (E) (p.a=1 AND q.b=2 AND r.c=3) OR (p.x=4 AND q.y=5 AND r.z=6) ** (F) x>A OR (x=A AND y>B) ** ** CASE 1: ** ** If all subterms are of the form T.C=expr for some single column of C and ** a single table T (as shown in example B above) then create a new virtual ** term that is an equivalent IN expression. In other words, if the term ** being analyzed is: ** ** x = expr1 OR expr2 = x OR x = expr3 ** ** then create a new virtual term like this: ** ** x IN (expr1,expr2,expr3) ** ** CASE 2: ** ** If there are exactly two disjuncts and one side has x>A and the other side ** has x=A (for the same x and A) then add a new virtual conjunct term to the ** WHERE clause of the form "x>=A". Example: ** ** x>A OR (x=A AND y>B) adds: x>=A ** ** The added conjunct can sometimes be helpful in query planning. ** ** CASE 3: ** ** If all subterms are indexable by a single table T, then set ** ** WhereTerm.eOperator = WO_OR ** WhereTerm.u.pOrInfo->indexable |= the cursor number for table T ** ** A subterm is "indexable" if it is of the form ** "T.C <op> <expr>" where C is any column of table T and ** <op> is one of "=", "<", "<=", ">", ">=", "IS NULL", or "IN". ** A subterm is also indexable if it is an AND of two or more ** subsubterms at least one of which is indexable. Indexable AND ** subterms have their eOperator set to WO_AND and they have ** u.pAndInfo set to a dynamically allocated WhereAndTerm object. ** ** From another point of view, "indexable" means that the subterm could ** potentially be used with an index if an appropriate index exists. ** This analysis does not consider whether or not the index exists; that ** is decided elsewhere. This analysis only looks at whether subterms ** appropriate for indexing exist. ** ** All examples A through E above satisfy case 3. But if a term ** also satisfies case 1 (such as B) we know that the optimizer will ** always prefer case 1, so in that case we pretend that case 3 is not ** satisfied. ** ** It might be the case that multiple tables are indexable. For example, ** (E) above is indexable on tables P, Q, and R. ** ** Terms that satisfy case 3 are candidates for lookup by using ** separate indices to find rowids for each subterm and composing ** the union of all rowids using a RowSet object. This is similar ** to "bitmap indices" in other database engines. ** ** OTHERWISE: **** If none of cases 1, 2, or 3 apply, then leave the eOperator set to ** zero. This term is not useful for search.

61375. tcons ::= CHECK LP expr RP onconf
61376. 84
61377. ** Clear the content of node p (set all bytes to 0x00).
61378. Index of column in first table
61379. The FROM clause term to process
61380. For every "*" that occurs in the column list, insert the names of ** all columns in all tables. And for every TABLE.* insert the names ** of all columns in TABLE.
The parser inserted a special expression ** with the TK_ASTERISK operator for each "*" that it found in the column ** list. The following code just has to locate the TK_ASTERISK ** expressions and expand each one to the list of all columns in ** all tables. *** The first loop just checks to see if there are any "*" operators ** that need expanding.
61381. Number of registers left to copy
61382. 930
61383. If this session is attached to a different database ("main", "temp" ** etc.), or if it is not currently enabled, there is nothing to do. Skip ** to the next session object attached to this database.
61384. **comment:** We can only reach this point when reading a corrupt database ** file. In that case we are not in any hurry. Use the (relatively ** slow) general-purpose sqlite3GetVarint() routine to extract the ** value.
label: code-design
61385. The expression to code when the Vdbe initializes
61386. **comment:** ** Finalize the statement passed as the second argument. *** If the sqlite3_finalize() call indicates that an error occurs, and the ** rdb handle error code is not already set, set the error code and error ** message accordingly.
label: code-design
61387. True once backup has been registered with pager
61388. ** Free all memory allocations associated with the tree pTree.
61389. ** CAPI3REF: Error Codes And Messages ** METHOD: sqlite3 ** ** ^If the most recent sqlite3_* API call associated with ** [database connection] D failed, then the sqlite3_errcode(D) interface ** returns the numeric [result code] or [extended result code] for that ** API call. ** If the most recent API call was successful, ** then the return value from sqlite3_errcode() is undefined. ** ^The sqlite3_extended_errcode() ** interface is the same except that it always returns the ** [extended result code] even when extended result codes are ** disabled. *** ^The sqlite3_errmsg() and sqlite3_errmsg16() return English-language ** text that describes the error, as either UTF-8 or UTF-16 respectively. ** ^Memory to hold the error message string is managed internally. ** The application does not need to worry about freeing the result. ** However, the error string might be overwritten or deallocated by ** subsequent calls to other SQLite interface functions.** ^The sqlite3_errstr() interface returns the English-language text ** that describes the [result code], as UTF-8. ** ^Memory to hold the error message string is managed internally ** and must not be freed by the application).** ^When the serialized [threading mode] is in use, it might be the ** case that a second error occurs on a separate thread in between ** the time of the first error and the call to these interfaces. ** When that happens, the second error will be reported since these ** interfaces always report the most recent result. To avoid ** this, each thread can obtain exclusive use of the [database connection] D ** by invoking [sqlite3_mutex_enter][[sqlite3_db_mutex](D)]) before beginning ** to use D and invoking [sqlite3_mutex_leave][[sqlite3_db_mutex](D)) after ** all calls to the interfaces listed here are completed. *** If an interface fails with SQLITE_MISUSE, that means the interface ** was invoked incorrectly by the application. In that case, the ** error code and message may or may not be set.
61390. If the opcode is TK_TRIGGER, then the expression is a reference ** to a column in the new.* or old.* pseudo-tables available to ** trigger programs. In this case Expr.iTable is set to 1 for the ** new.* pseudo-table, or 0 for the old.* pseudo-table. Expr.iColumn ** is set to the column of the pseudo-table to read, or to -1 to ** read the rowid field. *** The expression is implemented using an OP_Param opcode. The p1 ** parameter is set to 0 for an old.rowid reference, or to (i+1) ** to reference another column of the old.* pseudo-table, where ** i is the index of the column. For a new.rowid reference, p1 is ** set to (n+1), where n is the number of columns in each pseudo-table. ** For a reference to any other column in the new.* pseudo-table, p1 ** is set to (n+2+i), where n and i are as defined previously. For ** example, if the table on which triggers are being fired is ** declared as: *** CREATE TABLE t1(a, b); ** ** Then p1 is interpreted as follows: *** ** p1==0 -> old.rowid p1==3 -> new.rowid ** p1==1 -> old.a p1==4 -> new.a ** p1==2 -> old.b p1==5 -> new.b
61391. This constraint creates the same index as a previous ** constraint specified somewhere in the CREATE TABLE statement. ** However the ON CONFLICT clauses are different. If both this ** constraint and the previous equivalent constraint have explicit ** ON CONFLICT clauses this is an error. Otherwise, use the ** explicitly specified behavior for the index.
61392. Pointers to the body of overflow cells
61393. ** Generate code that will increment the schema cookie. *** The schema cookie is used to determine when the schema for the ** database changes. After each schema change, the cookie value ** changes. When a process first reads the schema it records the ** cookie. Thereafter, whenever it goes to access the database, ** it checks the cookie to make sure the schema has not changed ** since it was last read. *** This plan is not completely bullet-proof. It is possible for ** the schema to change multiple times and for the cookie to be ** set back to prior value. But schema changes are infrequent ** and the probability of hitting the same cookie value is only ** 1 chance in 2^32. So we're safe enough. *** IMPLEMENTATION-OF: R-34230-56049 SQLite automatically increments ** the schema-version whenever the schema changes.
61394. Because of isCandidateForInOpt(p)
61395. A copy of pPage->pBt
61396. FROM clause of subquery
61397. Cursor is for seek/delete only
61398. Name of automatically created rdb vfs
61399. Copy data from buffer to page (a write operation)
61400. Position list data following iHead
61401. (281) savepoint_opt ::= SAVEPOINT
61402. Argument to xInit() and xShutdown()
61403. Rowid is stored in this register, if not zero
61404. 1540
61405. What to do with the lock
61406. Length of string z
61407. x is the INTEGER PRIMARY KEY
61408. Total number of blocks read
61409. Column is part of the primary key
61410. OP_Compare: use the permutation
61411. The UTF-8 string to compare against
61412. Write the results into a priority queue that is ordered according to ** pDest->pOrderBy (in pSO). pDest->iSDParm (in iParm) is the cursor for an ** index with pSO->nExpr+2 columns. Build a key using pSO for the first ** pSO->nExpr columns, then make sure all keys are unique by adding a ** final OP_Sequence column. The last column is the record as a blob.
61413. ***** Unix Pthreads *****
61414. expr ::= expr OR expr
61415. **comment:** If this is the first term selected, copy the doclist to the output ** buffer using memcpy(). *** Add FTS3_VARINT_MAX bytes of unused space to the end of the ** allocation. This is so as to ensure that the buffer is big enough ** to hold the current doclist AND'd with any other doclist. If the ** doclists are stored in order=ASC order, this padding would not be ** required (since the size of [doclistA AND doclistB] is always less ** than or equal to the size of [doclistA] in that case). But this is ** not true for order=DESC. For example, a doclist containing (1, -1) ** may be smaller than (-1), as in the first example the -1 may be stored ** as a single-byte delta, whereas in the second it must be stored as a ** FTS3_VARINT_MAX byte varint. *** Similar padding is added in the fts3DoclistOrMerge() function.
label: code-design
61416. Current OR clause term
61417. ** CAPI3REF: Register A Callback To Handle SQLITE_BUSY Errors ** KEYWORDS: {busy-handler callback} {busy handler} ** METHOD: sqlite3 *** ^The sqlite3_busy_handler(D,X,P) routine sets a callback function X ** that might be invoked with argument P whenever ** an attempt is made to access a database table associated with ** [database connection] D when another thread ** or process has the table locked. ** The sqlite3_busy_handler() interface is used to implement ** [sqlite3_busy_timeout()] and [PRAGMA busy_timeout]. *** ^If the busy callback is NULL, then [SQLITE_BUSY] ** is returned immediately upon encountering the lock. ^If the busy callback ** is not NULL, then the callback might be invoked with two arguments. *** ^The first argument to the busy

handler is a copy of the void* pointer which ** is the third argument to sqlite3_busy_handler(). ^The second argument to ** the busy handler callback is the number of times that the busy handler has ** been invoked previously for the same locking event. ^If the ** busy callback returns 0, then no additional attempts are made to ** access the database and [SQLITE_BUSY] is returned ** to the application. ** ^If the callback returns non-zero, then another attempt ** is made to access the database and the cycle repeats. ** ** The presence of a busy handler does not guarantee that it will be invoked ** when there is lock contention. ^If SQLite determines that invoking the busy ** handler could result in a deadlock, it will go ahead and return [SQLITE_BUSY] ** to the application instead of invoking the ** busy handler. ** Consider a scenario where one process is holding a read lock that ** it is trying to promote to a reserved lock and ** a second process is holding a reserved lock that it is trying ** to promote to an exclusive lock. The first process cannot proceed ** because it is blocked by the second and the second process cannot ** proceed because it is blocked by the first. If both processes ** invoke the busy handlers, neither will make any progress. Therefore, ** SQLite returns [SQLITE_BUSY] for the first process, hoping that this ** will induce the first process to release its read lock and allow ** the second process to proceed. ** ** ^The default busy callback is NULL. ** ** ^There can only be a single busy handler defined for each ** [database connection]. Setting a new busy handler clears any ** previously set handler. ^A Note that calling [sqlite3_busy_timeout()] ** or evaluating [PRAGMA busy_timeout=N] will change the ** busy handler and thus clear any previously set busy handler. ** ** The busy callback should not take any actions which modify the ** database connection that invoked the busy handler. In other words, ** the busy handler is not reentrant. Any such actions ** result in undefined behavior. ** ** A busy handler must not close the database connection ** or [prepared statement] that invoked the busy handler.

- 61418. One or more subtasks
- 61419. Select from a "CREATE ... AS SELECT"
- 61420. Set the lower and upper bounds on docids to return
- 61421. Number of cells in page pPage
- 61422. ** Many compilers we encounter do not define constants for the ** minimum and maximum 64-bit integers, or they define them ** inconsistently. And many do not understand the "LL" notation. ** So we define our own static constants here using nothing ** larger than a 32-bit integer constant.
- 61423. ** If the PmaReader passed as the first argument is not an incremental-reader ** (if pReadr->pIncr==0), then this function is a no-op. Otherwise, it invokes ** the vdbePmaReaderIncrMergeInit() function with the parameters passed to ** this routine to initialize the incremental merge. ** ** If the IncrMerger object is multi-threaded (IncrMerger.bUseThread==1), ** then a background thread is launched to call vdbePmaReaderIncrMergeInit(). ** Or, if the IncrMerger is single threaded, the same function is called ** using the current thread.
- 61424. End of analysis
- 61425. If a languageid= option was specified, remove the language id ** column from the aCol[] array.
- 61426. **comment:** Not WITHOUT ROWID table. (FIXME: Why not?)
label: code-design
- 61427. Maximum file pathname length
- 61428. Number of errors seen
- 61429. Try to get the file to memory map
- 61430. 58
- 61431. If the lock failed (busy): * 1st try: get the mod time of the conch, wait 0.5s and try again. * 2nd try: fail if the mod time changed or host id is different, wait * 10 sec and try again * 3rd try: break the lock unless the mod time has changed.
- 61432. !defined(SQLITE_DISABLE_FTS3_UNICODE)
- 61433. No match found
- 61434. **comment:** ** Sort the list of pages in ascending order by pgno. Pages are ** connected by pDirty pointers. The pDirtyPrev pointers are ** corrupted by this sort. ** ** Since there cannot be more than 2^31 distinct pages in a database, ** there cannot be more than 31 buckets required by the merge sorter. ** One extra bucket is added to catch overflow in case something ** ever changes to make the previous sentence incorrect.
label: documentation
- 61435. **comment:** ** CAPI3REF: Apply A Changeset To A Database ** ** Apply a changeset to a database. This function attempts to update the ** "main" database attached to handle db with the changes found in the ** changeset passed via the second and third arguments. ** ** The fourth argument (xFilter) passed to this function is the "filter ** callback". If it is not NULL, then for each table affected by at least one ** change in the changeset, the filter callback is invoked with ** the table name as the second argument, and a copy of the context pointer ** passed as the sixth argument to this function as the first. If the "filter ** callback" returns zero, then no attempt is made to apply any changes to ** the table. Otherwise, if the return value is non-zero or the xFilter ** argument to this function is NULL, all changes related to the table are ** attempted. ** ** For each table that is not excluded by the filter callback, this function ** tests that the target database contains a compatible table. A table is ** considered compatible if all of the following are true: ** ** ** The table has the same name as the name recorded in the ** changeset, and ** The table has at least as many columns as recorded in the ** changeset, and ** The table has primary key columns in the same position as ** recorded in the changeset. ** ** ** If there is no compatible table, it is not an error, but none of the ** changes associated with the table are applied. A warning message is issued ** via the sqlite3_log() mechanism with the error code SQLITE_SCHEMA. At most ** one such warning is issued for each table in the changeset. ** ** For each change for which there is a compatible table, an attempt is made ** to modify the table contents according to the UPDATE, INSERT or DELETE ** change. If a change cannot be applied cleanly, the conflict handler ** function passed as the fifth argument to sqlite3changeset_apply() may be ** invoked. A description of exactly when the conflict handler is invoked for ** each type of change is below. ** ** Unlike the xFilter argument, xConflict may not be passed NULL. The results ** of passing anything other than a valid function pointer as the xConflict ** argument are undefined. ** ** Each time the conflict handler function is invoked, it must return one ** of [SQLITE_CHANGESET OMIT], [SQLITE_CHANGESET_ABORT] or ** [SQLITE_CHANGESET_REPLACE]. SQLITE_CHANGESET_REPLACE may only be returned ** if the second argument passed to the conflict handler is either ** SQLITE_CHANGESET_DATA or SQLITE_CHANGESET_CONFLICT. If the conflict-handler ** returns an illegal value, any changes already made are rolled back and ** the call to sqlite3changeset_apply() returns SQLITE_MISUSE. Different ** actions are taken by sqlite3changeset_apply() depending on the value ** returned by each invocation of the conflict-handler function. Refer to ** the documentation for the three ** [SQLITE_CHANGESET OMIT]available return values] for details. ** ** <dl> ** <dt>DELETE Changes<dd> ** For each DELETE change, this function checks if the target database ** contains a row with the same primary key value (or values) as the ** original row values stored in the changeset. If it does, and the values ** stored in all non-primary key columns also match the values stored in ** the changeset the row is deleted from the target database. ** ** If a row with matching primary key values is found, but one or more of ** the non-primary key fields contains a value different from the original ** row value stored in the changeset, the conflict-handler function is ** invoked with [SQLITE_CHANGESET DATA] as the second argument. If the ** database table has more columns than are recorded in the changeset, ** only the values of those non-primary key fields are compared against ** the current database contents - any trailing database table columns ** are ignored. ** ** If no row with matching primary key values is found in the database, ** the conflict-handler function is invoked with [SQLITE_CHANGESET NOTFOUND] ** passed as the second argument. ** ** If the DELETE operation is attempted, but SQLite returns SQLITE_CONSTRAINT ** (which can only happen if a foreign key constraint is violated), the ** conflict-handler function is invoked with [SQLITE_CHANGESET CONSTRAINT] ** passed as the second argument. This includes the case where the DELETE ** operation is attempted because an earlier call to the conflict handler ** function returned [SQLITE_CHANGESET_REPLACE]. ** ** <dt>INSERT Changes<dd> ** For each INSERT change, an attempt is made to insert the new row into ** the database. If the changeset row contains fewer fields than the ** database table, the trailing fields are populated with their default ** values. ** ** If the attempt to insert the row fails because the database already ** contains a row with the same primary key values, the conflict handler ** function is invoked with the second argument set to ** [SQLITE_CHANGESET CONFLICT]. ** ** If the attempt to insert the row fails because of some other constraint ** violation (e.g. NOT NULL or UNIQUE), the conflict handler function is ** invoked with the second argument set to [SQLITE_CHANGESET CONSTRAINT]. ** This includes the case where the INSERT operation is re-attempted because ** an earlier call to the conflict handler function returned ** [SQLITE_CHANGESET_REPLACE]. ** ** <dt>UPDATE Changes<dd> ** For each UPDATE change, this function checks if the target database ** contains a row with the same primary key value (or values) as the ** original row values stored in the changeset. If it does, and the values ** stored in all modified non-primary key columns also match the values ** stored in the changeset the row is updated within the target database. ** ** If a row with matching primary key values is found, but one or more of ** the modified non-primary key fields contains a value different from an ** original row value stored in the changeset, the conflict-handler function ** is invoked with [SQLITE_CHANGESET DATA] as the second argument. Since ** UPDATE changes only contain values for non-primary key fields that are ** to be modified, only those fields need to match the original values to ** avoid the SQLITE_CHANGESET DATA conflict-handler callback. ** ** If no row with matching primary key values is found in the database, ** the conflict-handler function is invoked with [SQLITE_CHANGESET NOTFOUND] ** passed as the second argument. ** ** If the UPDATE operation is attempted, but SQLite returns ** SQLITE_CONSTRAINT, the conflict-handler function is invoked with ** [SQLITE_CHANGESET CONSTRAINT] passed as the second argument. ** This includes the case where the UPDATE operation is attempted after ** an earlier call to the conflict handler function returned ** [SQLITE_CHANGESET_REPLACE]. ** </dl> ** ** It is safe to execute SQL statements, including those that write to the ** table that the callback related to, from within the xConflict callback. ** This can be used to further customize the applications conflict ** resolution strategy. ** ** All changes made by this

function are enclosed in a savepoint transaction. ** If any other error (aside from a constraint failure when attempting to ** write to the target database) occurs, then the savepoint transaction is ** rolled back, restoring the target database to its original state, and an ** SQLite error code returned.

label: code-design

61436. in3: P3 is an input

61437. Return the number of rows that were deleted. If this routine is ** generating code because of a call to sqlite3NestedParse(), do not ** invoke the callback function.

61438. explain ::= EXPLAIN QUERY PLAN

61439. New path replaces the prior worst to keep count below mxChoice

61440. Otherwise, fall thru into the TK_COLUMN case

61441. Default lookaside buffer count

61442. Number of equality constraints

61443. This can happen using journal_mode=off. Move the pager to the error ** state to indicate that the contents of the cache may not be trusted. ** Any active readers will get SQLITE_ABORT.

61444. ** Drop a trigger given a pointer to that trigger.

61445. 28

61446. Search an ephemeral b-tree

61447. localtime ** ** Assuming the current time value is UTC (a.k.a. GMT), shift it to ** show local time.

61448. Number of fragmented bytes on the page

61449. True if IPK can be equal to maxKey ** False if IPK must be strictly less than maxKey

61450. If there is not already a read-only (or read-write) transaction opened ** on the b-tree database, open one now. If a transaction is opened, it ** will be closed immediately after reading the meta-value.

61451. String in Mem.z is zero terminated

61452. ccons ::= DEFAULT LP expr RP

61453. **comment:** ** At least two bugs have slipped in because we changed the MEMORY_DEBUG ** macro to SQLITE_DEBUG and some older makefiles have not yet made the ** switch. The following code should catch this problem at compile-time.

label: code-design

61454. OUT: Block id of last entry written

61455. Filter used with cursor pCsr

61456. typetoken ::= typename LP signed RP

61457. Special case: Parsing the sqlite_master or sqlite_temp_master schema

61458. ** An instance of the following structure is used as a dynamic buffer ** to build up nodes or other blobs of data in. ** ** The function blobGrowBuffer() is used to extend the allocation.

61459. ** Write up to nBuf bytes of randomness into zBuf.

61460. OUT: End of parsed string

61461. Reference counted destructor function

61462. ** The cursor passed as the only argument must point to a valid entry ** when this function is called (i.e. have eState==CURSOR_VALID). This ** function saves the current cursor key in variables pCur->nKey and ** pCur->pKey. SQLITE_OK is returned if successful or an SQLite error ** code otherwise. ** ** If the cursor is open on an intkey table, then the integer key ** (the rowid) is stored in pCur->nKey and pCur->pKey is left set to ** NULL. If the cursor is open on a non-intkey table, then pCur->pKey is ** set to point to a mallocoed buffer pCur->nKey bytes in size containing ** the key.

61463. ** Transfer all bindings from the first statement over to the second.

61464. No. of reg prior to regData available for use

61465. zero columns if RHS argument is present

61466. Operands of the opcode used to ends the loop

61467. Reset the cursor to the same state as rtreeOpen() leaves it in.

61468. path for the new unixFile

61469. limit_opt ::=

61470. A block of WALINDEX_LOCK_RESERVED bytes beginning at ** WALINDEX_LOCK_OFFSET is reserved for locks. Since some systems ** only support mandatory file-locks, we do not read or write data ** from the region of the file on which locks are applied.

61471. Transform this into an alias to the result set

61472. True to fail EXCLUSIVE locks

61473. Opcode: Le P1 P2 P3 P4 P5 ** Synopsis: IF r[P3]<=r[P1] ** ** This works just like the Lt opcode except that the jump is taken if ** the content of register P3 is less than or equal to the content of ** register P1. See the Lt opcode for additional information.

61474. RENAME

61475. Size of source db in pages

61476. Foreign key to get action for

61477. Rowid of row about to be deleted/updated

61478. Root page of scanned b-tree

61479. Scratch memory

61480. Expression we are trying to match

61481. token.z was originally in "..."

61482. ** Set the suggested cache-spill value. Make no changes if if the ** argument is zero. Return the effective cache-spill size, which will ** be the larger of the szSpill and szCache.

61483. True for write lock. False for a read lock

61484. Construct a query to find the rowid or primary key for every row ** to be deleted, based on the WHERE clause. Set variable eOnePass ** to indicate the strategy used to implement this delete: ** ** ONEPASS_OFF: Two-pass approach - use a FIFO for rowids/PK values. ** ONEPASS_SINGLE: One-pass approach - at most one row deleted. ** ONEPASS_MULTI: One-pass approach - any number of rows may be deleted.

61485. SQLITE_OMIT_COMPILEOPTION_DIAGS

61486. ***** Include os_common.h in the middle of os_unix.c *****

61487. !defined(SQLITE_CORE) || defined(SQLITE_ENABLE_FTS3)

61488. The group by clause

61489. **comment:** ** The sqlite3_mutex_try() routine is very rarely used, and when it ** is used it is merely an optimization. So it is OK for it to always ** fail. ** ** The TryEnterCriticalSection() interface is only available on WinNT. ** And some windows compilers complain if you try to use it without ** first doing some #defines that prevent SQLite from building on Win98. ** For that reason, we will omit this optimization for now. See ** ticket #2685.

label: code-design

61490. ** Process statements of the form: ** ** INSERT INTO table(table) VALUES('merge=A,B'); ** ** A and B are integers that decode to be the number of leaf pages ** written for the merge, and the minimum number of segments on a level ** before it will be selected for a merge, respectively.

61491. Result of a successful parse

61492. ** Given the name of a database file, compute the name of its conch file. ** Store the conch filename in memory obtained from sqlite3_malloc64(). ** Make *pConchPath point to the new name. Return SQLITE_OK on success ** or SQLITE_NOMEM if unable to obtain memory. ** ** The caller is responsible for ensuring that the allocated memory ** space is eventually freed. ** ** *pConchPath is set to NULL if a memory allocation error occurs.

61493. ** This routine is callback for sqlite3WalkExpr(). ** ** Resolve symbolic names into TK_COLUMN operators for the current ** node in the expression tree. Return 0 to continue the search down ** the tree or 2 to abort the tree walk. ** ** This routine also does error checking and name resolution for ** function names. The operator for aggregate functions is changed ** to TK_AGG_FUNCTION.

61494. String to return

61495. Zero values greater than this

61496. 207

61497. The parser to be deleted

61498. SQLITE_AFF_REAL

61499. Store the current value of the database handles deferred constraint ** counter. If the statement transaction needs to be rolled back, ** the value of this counter needs to be restored too.

61500. 317

61501. Bitmap of columns used for indexing

61502. DETACH

61503. *** End of interface to code in fts5_varint.c. *****

61504. Index iterator

61505. ** Maximum size of any allocation is ((1<<LOGMAX)*mem5.szAtom). Since ** mem5.szAtom is always at least 8 and 32-bit integers are used, ** it is not actually possible to reach this limit.

61506. ** Windows will only let you create file view mappings ** on allocation size granularity boundaries. ** During sqlite3_os_init() we do a GetSystemInfo() ** to get the granularity size.

61507. Register a trace callback using the version-2 interface.

61508. ** Attempt to start a new transaction. A write-transaction ** is started if the second argument is nonzero, otherwise a read- ** transaction. If the second argument is 2 or more and exclusive ** transaction is started, meaning that no other process is allowed ** to access the database. A preexisting transaction may not be ** upgraded to exclusive by calling this routine a second time - the ** exclusivity flag only works for a new transaction. ** ** A write-transaction must be started before attempting any ** changes to the database. None of the following routines ** will work unless a transaction is started first: ** ** sqlite3BtreeCreateTable() ** sqlite3BtreeCreateIndex() ** sqlite3BtreeClearTable() ** sqlite3BtreeDropTable() ** sqlite3BtreeInsert() ** sqlite3BtreeDelete() ** sqlite3BtreeUpdateMeta() ** ** If an initial attempt to acquire the lock fails because of lock contention ** and the database was previously unlocked, then invoke the busy handler ** if there is one. But if there was previously a read-lock, do not ** invoke the busy handler - just return SQLITE_BUSY. SQLITE_BUSY is ** returned when there is already a read-lock in order to avoid a deadlock. ** ** Suppose there are two processes A and B. A has a read lock and B has ** a reserved lock. B tries to promote to exclusive but is blocked because ** of A's read lock. A tries to promote to reserved but is blocked by B. ** One or the other of the two processes must give way or there can be ** no progress. By returning SQLITE_BUSY and not invoking the busy callback ** when A already has a read lock, we encourage A to give up and let B ** proceed.

61509. ***** Interface to code in fts5_hash.c.

61510. ** GCC does not define the offsetof() macro so we'll have to do it ** ourselves.

61511. Column number of matching column on the right

61512. ** Constants for the largest and smallest possible 64-bit signed integers. ** These macros are designed to work correctly on both 32-bit and 64-bit ** compilers.

61513. Offset to the cell pointer array

61514. Bitmask of tables referenced by pExpr

61515. If we reach this point, it means the two subterms can be combined

61516. Open and close a connection to a write-ahead log.

61517. Size of one page

61518. Number of the page to get

61519. ** Used as an fts3ExprIterate() context when loading phrase doclists to ** Fts3Expr.aDoclist[]/nDoclist.

61520. Number of bytes of URI args at *zUri

61521. Boolean properties. See COLFLAG_ defines below

61522. Take a shared-cache advisory read-lock on the parent table. Allocate ** a cursor to use to search the unique index on the parent key columns ** in the parent table.

61523. plidx is a covering index. No need to access the main table.

61524. Overwrite deleted information with zeros when the secure_delete ** option is enabled

61525. Opcode: OpenDup P1 P2 * * * ** Open a new cursor P1 that points to the same ephemeral table as ** cursor P2. The P2 cursor must have been opened by a prior OP_OpenEphemeral ** opcode. Only ephemeral cursors may be duplicated. ** ** Duplicate ephemeral cursors are used for self-joins of materialized views.

61526. If true, store 0 for segment size

61527. Minimum amount of payload held locally

61528. Samples of the left-most key

61529. Find any inequality constraint terms for the start and end ** of the range.

61530. #include "os_win.h"

61531. Number of rows in the entire table

61532. List of triggers on pTab, if required

61533. ** True if we are evaluating an out-of-memory callback.

61534. Check for index coverage

61535. Store the value in this register

61536. EXCLUSIVE => ID

61537. Number of entries in aVar[]

61538. ** Add an entry to the array of counters managed by sqlite3_stmt_scanstatus().

61539. Size of term in bytes

61540. The content of the whole page

61541. ** PRAGMA table_info(<table>) ** ** Return a single row for each column of the named table. The columns of ** the returned data set are: ** ** cid: Column id (numbered from left to right, starting at 0) ** name: Column name ** type: Column declaration type. ** notnull: True if 'NOT NULL' is part of column declaration ** dflt_value: The default value for the column, if any.

61542. ***** Include vdbe.h in the middle of sqliteInt.h *****

61543. Initialize all iterators

61544. If there is anything other than a rowset object in memory cell P1, ** delete it now and initialize P1 with an empty rowset

61545. idx

61546. ** FTS4 virtual tables may maintain multiple indexes - one index of all terms ** in the document set and zero or more prefix indexes. All indexes are stored ** as one or more b+-trees in the %_segments and %_segdir tables. ** ** It is possible to determine which index a b+-tree belongs to based on the ** value stored in the "%_segdir.level" column. Given this value L, the index ** that the b+-tree belongs to is (L<<10). In other words, all b+-trees with ** level values between 0 and 1023 (inclusive) belong to index 0, all levels ** between 1024 and 2047 to index 1, and so on. ** ** It is considered impossible for an index to use more than 1024 levels. In ** theory though this may happen, but only after at least ** (FTS3_MERGE_COUNT^1024) separate flushes of the pending-terms tables.

61547. If pToRelease is not zero than pPrior points into the data area ** of pToRelease. Make sure pToRelease is still writeable.

61548. The virtual table to be modified

61549. ** Return the declared type of a column. Or return zDflt if the column ** has no declared type. ** ** The column type is an extra string stored after the zero-terminator on ** the column name if and only if the COLFLAG_HASTYPE flag is set.

61550. For each term in the FROM clause, do two things: ** (1) Authorized unreferenced tables ** (2) Generate code for all sub-queries

61551. sortlist ::= sortlist COMMA expr sortorder

61552. Page not on the PCache.pDirty list

61553. ***** General Implementation Functions *****

61554. Number of SHARED locks held

61555. Find out what action to take in case there is a uniqueness conflict

61556. Length of position list

61557. Generate code to remove the index and from the master table

61558. Opcode: Jump P1 P2 P3 * * * ** Jump to the instruction at address P1, P2, or P3 depending on whether ** in the most recent OP_Compare instruction the P1 vector was less than ** equal to, or greater than the P2 vector, respectively.

61559. The thread routine

61560. **comment:** By default use a full table scan. This is an expensive option, ** so search through the constraints to see if a more efficient ** strategy is possible.
label: code-design

61561. **comment:** Save the current term so that it can be used to prefix-compress the next. ** If the isCopyTerm parameter is true, then the buffer pointed to by ** zTerm is transient, so take a copy of the term data. Otherwise, just ** store a copy of the pointer.
label: code-design

61562. ** Mark term iChild as being a child of term iParent
61563. Enlarge the allocation
61564. Leaf pages in a "unit" of work
61565. The token codes above must all fit in 8 bits
61566. Allocated array
61567. Number of input segments on best level
61568. 3x
61569. ** 2008 May 27 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains inline asm code for retrieving "high-performance" ** counters for x86 class CPUs.
61570. Values for xColumnSize()
61571. 103
61572. same as TK_EQ, jump, in1, in3
61573. SELECT may not be DISTINCT
61574. Value is an integer
61575. SQLite error code
61576. **comment:** ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** The code in this file implements the function that runs the ** bytecode of a prepared statement. *** ** Various scripts scan this source file in order to generate HTML ** documentation, headers files, or other derived files. The formatting ** of the code in this file is, therefore, important. See other comments ** in this file for details. If in doubt, do not deviate from existing ** commenting and indentation practices when changing or adding code.
label: code-design
61577. pPager->errCode = 0;
61578. If the RHS is a vector, then we can immediately check to see that ** the size of the RHS and LHS match. But if the RHS is a SELECT, ** wildcards ("*") in the result set of the SELECT must be expanded before ** we can do the size check, so defer the size check until code generation.
61579. Remove either the OP_OpenWrite or OpenRead. Set the P2 ** parameter of the other to pTab->tnum.
61580. ***** Begin file vdbletrace.c *****
61581. Start of subroutine that outputs a result row
61582. 48..4f HIJKLMNO
61583. ** Return a KeyInfo structure that is appropriate for the given Index. *** ** The caller should invoke sqlite3KeyInfoUnref() on the returned object ** when it has finished using it.
61584. Put the new key into this register if not 0
61585. Chain of simple selects to delete
61586. Compute the complete text of the CREATE VIRTUAL TABLE statement
61587. ** Return TRUE if database connection db has unfinished prepared ** statements or unfinished sqlite3_backup objects.
61588. ** Walk an expression tree. Invoke the callback once for each node ** of the expression, while descending. (In other words, the callback ** is invoked before visiting children.) *** ** The return value from the callback should be one of the WRC_* ** constants to specify how to proceed with the walk. *** **
WRC_Continue Continue descending down the tree. *** ** WRC_Prune Do not descend into child nodes, but allow ** the walk to continue with sibling nodes. **
** WRC_Abort Do no more callbacks. Unwind the stack and ** return from the top-level walk call. *** ** The return value from this routine is WRC_Abort to abandon the tree walk ** and WRC_Continue to continue.
61589. Add "0" or "0x"
61590. NOT REACHED
61591. Ideal number of tokens to shift forward
61592. Opcode: Cast P1 P2 * * * Synopsis: affinity(r[P1]) *** ** Force the value in register P1 to be the type defined by P2. *** ** ** P2=='A' → BLOB
** P2=='B' → TEXT ** P2=='C' → NUMERIC ** P2=='D' → INTEGER ** P2=='E' → REAL *** *** ** A NULL value is not changed by this routine. It remains NULL.
61593. selectNodeIsConstant will disallow
61594. Unread parts of the buffer must be zero-filled
61595. ** Implementation of the sqlite_compileoption_get() function. ** The result is a string that identifies the compiler options ** used to build SQLite.
61596. The new table name.
61597. ** Generate code that will *** (1) acquire a lock for table pTab then ** (2) open pTab as cursor iCur. *** ** If pTab is a WITHOUT ROWID table, then it is the PRIMARY KEY index ** for that table that is actually opened.
61598. seltablist ::= stl_prefix nm dbnm as indexed_opt on_opt using_opt
61599. Allocated size of aFrame[] array
61600. Index of containing database.
61601. If the B-Tree was successfully opened, set the pager-cache size to the ** default value. Except, when opening on an existing shared pager-cache, ** do not change the pager-cache size.
61602. Twice the number of dimensions
61603. For columns use the column name name
61604. C
61605. OUT: Bytes actually loaded
61606. Remove connection p from the set of connections associated ** with pShmNode
61607. a &= (0x7f<<14)|(0x7b);
61608. Versions of SQLite prior to 3.5.8 set the page-size field of the ** journal header to zero. In this case, assume that the Pager.pageSize ** variable is already set to the correct page size.
61609. SQL_DELETE_SEGDIR_ENTRY ** Delete the %_segdir entry on absolute level :1 with index :2.
61610. Populate new node buffer
61611. Total internal fragmentation
61612. someone else has the lock when we are in NO_LOCK
61613. Prior journalmode
61614. current position in pInput
61615. If we are doing a normal write to a database file (as opposed to ** doing a hot-journal rollback or a write to some file other than a ** normal database file) and we truncate the file to zero length, ** that effectively updates the change counter. This might happen ** when restoring a database using the backup API from a zero-length ** source.
61616. Used by reverse iterators
61617. Result code stored here
61618. Characters to treat as exceptions
61619. Next node in this hash collision chain
61620. Left subnode
61621. Stat4Accum.nMaxEqZero is set to the maximum number of leading 0 ** values in the anEq[] array of any sample in Stat4Accum.a[]. In ** other words, if nMaxEqZero is n, then it is guaranteed that there ** are no samples with Stat4Sample.anEq[m]==0 for (m>=n).
61622. Number of parameters to the SQL function
61623. Remove the cursor from the Fts5Global.pCsr list
61624. cmd ::= with UPDATE orconf fullname indexed_opt SET setlist where_opt
61625. OUT: Merged change
61626. Free any old pages that were not reused as new pages.

61627. ** Set the name of the idx'th column to be returned by the SQL statement. ** zName must be a pointer to a nul terminated string. ** ** This call must be made after a call to sqlite3VdbeSetNumCols(). ** ** The final parameter, xDel, must be one of SQLITE_DYNAMIC, SQLITE_STATIC ** or SQLITE_TRANSIENT. If it is SQLITE_DYNAMIC, then the buffer pointed ** to by zName will be freed by sqlite3DbFree() when the vdbe is destroyed.

61628. If SQLITE_CASE_SENSITIVE_LIKE is defined, then the LIKE operator ** is case sensitive causing 'a' LIKE 'A' to be false

61629. A column name: ID ** Or table name and column name: ID.ID ** Or a database, table and column: ID.ID.ID ** ** The TK_ID and TK_OUT cases are combined so that there will only ** be one call to lookupName(). Then the compiler will in-line ** lookupName() for a size reduction and performance increase.

61630. Linked list of main db files

61631. The entire segment is stored in the root node.

61632. createkw ::= CREATE

61633. joinop ::= JOIN_KW nm JOIN

61634. int int*

61635. Table of WHERE clause constraints

61636. Figure out if we have any triggers and if the table being ** updated is a view.

61637. EVIDENCE-OF: R-64536-51728 The values for each column in the record ** immediately follow the header.

61638. Configured by FCNTL_CHUNK_SIZE

61639. ** Structure version. Should always be set to 0 or 1.

61640. Current docid

61641. Language id

61642. CASCADE => ID

61643. **comment:** ** All of the static variables used by this module are collected ** into a single structure named "mem". This is to keep the ** static variables organized and to reduce namespace pollution ** when this module is combined with other in the amalgamation.

label: code-design

61644. Allocate registers for holding the rowid of the new row, ** the content of the new row, and the assembled row record.

61645. If the auto-commit flag is set to true, then any locks that were held ** by connection db have now been released. Call sqlite3ConnectionUnlocked() ** to invoke any required unlock-notify callbacks.

61646. ** Argument apSegment is an array of nSegment elements. It is known that ** the final (nSegment-nSuspect) members are already in sorted order ** (according to the comparison function provided). This function shuffles ** the array around until all entries are in sorted order.

61647. Cursor containing term and doclist

61648. Loop through all the foreign key constraints that refer to this table. ** (the "child" constraints)

61649. Tokenizer module methods object

61650. Pages from root to current page

61651. ** For LIKE and GLOB matching on EBCDIC machines, assume that every ** character is exactly one byte in size. Also, provide the Utf8Read() ** macro for fast reading of the next character in the common case where ** the next character is ASCII.

61652. **comment:** ** The page getter methods each try to acquire a reference to a ** page with page number pgno. If the requested reference is ** successfully obtained, it is copied to *ppPage and SQLITE_OK returned. ** ** There are different implementations of the getter method depending ** on the current state of the pager. *** ** getPageNormal() -- The normal getter ** getPageError() -- Used if the pager is in an error state ** getPageMmap() -- Used if memory-mapped I/O is enabled *** ** If the requested page is already in the cache, it is returned. ** Otherwise, a new page object is allocated and populated with data ** read from the database file. In some cases, the pcache module may ** choose not to allocate a new page object and may reuse an existing ** object with no outstanding references. ** ** The extra data appended to a page is always initialized to zeros the ** first time a page is loaded into memory. If the page requested is ** already in the cache when this function is called, then the extra ** data is left as it was when the page object was last used. ** ** If the database image is smaller than the requested page or if ** the flags parameter contains the PAGER_GET_NOCONTENT bit and the ** requested page is not already stored in the cache, then no ** actual disk read occurs. In this case the memory image of the ** page is initialized to all zeros. ** ** If PAGER_GET_NOCONTENT is true, it means that we do not care about ** the contents of the page. This occurs in two scenarios: ** ** a) When reading a free-list leaf page from the database, and ** ** b) When a savepoint is being rolled back and we need to load ** a new page into the cache to be filled with the data read ** from the savepoint journal. ** ** If PAGER_GET_NOCONTENT is true, then the data returned is zeroed instead ** of being read from the database. Additionally, the bits corresponding ** to pgno in Pager.pInJournal (bitvec of pages already written to the ** journal file) and the PagerSavepoint.pInSavepoint bitvecs of any open ** savepoints are set. This means if the page is made writable at any ** point in the future, using a call to sqlite3PagerWrite(), its contents ** will not be journaled. This saves IO. ** ** The acquisition might fail for several reasons. In all cases, ** an appropriate error code is returned and *ppPage is set to NULL. ** ** See also sqlite3PagerLookup(). Both this routine and Lookup() attempt ** to find a page in the in-memory cache first. If the page is not already ** in memory, this routine goes to disk to read it in whereas Lookup() ** just returns 0. This routine acquires a read-lock the first time it ** has to go to disk, and could also playback an old journal if necessary. ** Since Lookup() never goes to disk, it never has to deal with locks ** or journal files.

label: code-design

61653. ***** End of threads.c *****

61654. ***** Begin file fts3_tokenizer.h *****

61655. Named semaphore locking uses the file path so it needs to be ** included in the semLockingContext

61656. Specify an alternative mutex implementation

61657. Next offset to read from log file

61658. Bytes of storage available in zBuf[]

61659. ** Values for RbuObjIter.eType *** 0: Table does not exist (error) ** 1: Table has an implicit rowid. ** 2: Table has an explicit IPK column. ** 3: Table has an external PK index. ** 4: Table is WITHOUT ROWID. ** 5: Table is a virtual table.

61660. EP_Static if space not obtained from malloc

61661. ** Set *ppData to point to the Fts5Bm25Data object for the current query. ** If the object has not already been allocated, allocate and populate it ** now.

61662. ** Checkpoint database zDb. If zDb is NULL, or if the buffer zDb points ** to contains a zero-length string, all attached databases are ** checkpointed.

61663. Load automatic extensions - extensions that have been registered ** using the sqlite3_automatic_extension() API.

61664. ***** Page Allocation/SQLITE_CONFIG_PCACHE Related Functions *****

61665. ** These are the initializer values used when declaring a "static" mutex ** on Win32. It should be noted that all mutexes require initialization ** on the Win32 platform.

61666. 23

61667. ***** Begin file mem2.c *****

61668. Make nMaxCells a multiple of 4 in order to preserve 8-byte ** alignment

61669. 18

61670. EVIDENCE-OF: R-12976-22893 Value is the integer 0.

61671. ** Tokenization callback used when inserting tokens into the FTS index.

61672. Current value of 'col' column

61673. Calculate an IDF for each phrase in the query

61674. ** Generate code that will push the record in registers regData ** through regData+nData-1 onto the sorter.

61675. anLt[] + anEq[] of largest sample pRec is >

61676. ** Return true if we can prove the pE2 will always be true if pE1 is ** true. Return false if we cannot complete the proof or if pE2 might ** be false. Examples: ** ** pE1: x==5 pE2: x>=5 Result: true ** pE1: x>0 pE2: x==5 Result: false ** pE1: x=21 pE2: x=21 OR y=43 Result: true ** pE1: x!=123 pE2: x IS NOT NULL Result: true ** pE1: x!=?1 pE2: x IS NOT NULL Result: true ** pE1: x IS NULL pE2: x IS NOT NULL Result: false ** pE1: x IS ?2 pE2: x IS NOT NULL Result: false *** ** When comparing TK_COLUMN nodes between pE1 and pE2, if pE2 has ** Expr.iTable<0 then assume a table number given by iTab. ** ** If pParse is not NULL, then the values of bound variables in pE1 are ** compared against literal values in pE2 and pParse->pVdbe->expmask is ** modified to record which bound variables are referenced. If pParse ** is NULL, then false will be returned if pE1 contains any bound variables. ** ** When in doubt, return false. Returning true might give a performance ** improvement. Returning false might cause a performance reduction, but ** it will always give the correct answer and is hence always safe.

61677. ** Routines to implement min() and max() aggregate functions.

61678. If the trigger name was unqualified, and the table is a temp table, ** then set iDb to 1 to create the trigger in the temporary database. ** If sqlite3SrcListLookup() returns 0, indicating the table does not ** exist, the error is caught by the block below.

61679. Set to ", " later on
61680. ** 2004 May 26 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code use to manipulate "Mem" structure. A "Mem" ** stores a single value in the VDBE. Mem is an opaque structure visible ** only within the VDBE. Interface routines refer to a Mem using the ** name sqlite_value
61681. ***** End of hash.h *****
61682. ** SQLITE_ENABLE_EXPLAIN_COMMENTS is incompatible with SQLITE OMIT_EXPLAIN
61683. Add column type information to this table
61684. If bCloseTrans is true, then this function opened a read transaction ** on the source database. Close the read transaction here. There is ** no need to check the return values of the btree methods here, as ** "committing" a read-only transaction cannot fail.
61685. Add the freeblocks to the min-heap *** EVIDENCE-OF: R-20690-50594 The second field of the b-tree page header ** is the offset of the first freeblock, or zero if there are no ** freeblocks on the page.
61686. **comment:** Number of bytes of extra space to alloc
label: code-design
61687. If the database supports auto-vacuum, and the second or subsequent ** overflow page is being allocated, add an entry to the pointer-map ** for that page now. **
** If this is the first overflow page, then write a partial entry ** to the pointer-map. If we write nothing to this pointer-map slot, ** then the optimistic overflow chain processing in clearCell() ** may misinterpret the uninitialized values and delete the ** wrong pages from the database.
61688. When playing back page 1, restore the nReserve setting
61689. Jump destination
61690. Compose and prepare an SQL statement to loop through the content table
61691. Table has only 64-bit signed integer keys
61692. 2^32 - to avoid use of LL and warnings in gcc
61693. Use pread() and pwrite() if they are available
61694. Where clause context
61695. Magic number for detect library misuse
61696. Previous content. 0 for INSERTS
61697. ***** End of vxworks.h *****
61698. The index for which to generate a key
61699. ** Hash and comparison functions when the mode is FTS3_HASH_STRING
61700. This branch runs if the record-size field of the cell is a ** single byte varint and the record fits entirely on the main ** b-tree page.
61701. Number of Records in the journal
61702. Subset of btreeMask that requires a lock
61703. End of the loop
61704. ***** Begin file vdbemem.c *****
61705. ** Compute a hash on a page number. The resulting hash value must land ** between 0 and (HASHTABLE_NSLOT-1). The walHashNext() function advances ** the hash to the next value in the event of a collision.
61706. Open file descriptor of file being opened
61707. **comment:** ** Return true if the DISTINCT expression-list passed as the third argument ** is redundant. ** ** A DISTINCT list is redundant if any subset of the columns in the ** DISTINCT list are collectively unique and individually non-null.
label: code-design
61708. Rows are coming out in undetermined order. We have to push ** each row into a sorting index, terminate the first loop, ** then loop over the sorting index in order to get the output ** in sorted order
61709. Local cache of jsl
61710. If one has been configured, invoke the rollback-hook callback
61711. It is possible that if journal_mode=wal here that neither the ** journal file nor the WAL file are open. This happens during ** a rollback transaction that switches from journal_mode=off ** to journal_mode=wal.
61712. ***** End Unix Pthreads *****
61713. Check any rowid-less pages that occur before the current leaf.
61714. For looping over pList
61715. True if iDocid is valid
61716. Changset operation (SQLITE_UPDATE etc.)
61717. At EOF already
61718. Size of buffer pChangeset in bytes
61719. Table column this handle is open on
61720. Expression pPhrase belongs to
61721. Password used to authenticate
61722. This cell was just inserted
61723. OUT: Write the cell contents here
61724. DEFERRED
61725. colset ::= MINUS STRING
61726. 2-byte signed integer
61727. A parameter used by the eDest disposal method
61728. ** CAPI3REF: A Handle To An Open BLOB ** KEYWORDS: {BLOB handle} {BLOB handles} ** ** An instance of this object represents an open BLOB on which ** [sqlite3_blob_open | incremental BLOB I/O] can be performed. ** ^Objects of this type are created by [sqlite3_blob_open()] ** and destroyed by [sqlite3_blob_close()]. ** ^The [sqlite3_blob_read()] and [sqlite3_blob_write()] interfaces ** can be used to read or write small subsections of the BLOB. ** ^The [sqlite3_blob_bytes()] interface returns the size of the BLOB in bytes.
61729. UTF-16 encoded SQL statement.
61730. ** Try to insert a new prerequisite/cost entry into the WhereOrSet pSet. ** ** The new entry might overwrite an existing entry, or it might be ** appended, or it might be discarded. Do whatever is the right thing ** so that pSet keeps the N_OR_COST best entries seen so far.
61731. ** Conversion types fall into various categories as defined by the ** following enumeration.
61732. This happens when parsing a token or quoted phrase that contains ** no token characters at all. (e.g ... MATCH "")".
61733. 2 or greater. A position.
61734. This term does not restrict search space
61735. The statement to bind against
61736. Fall through into OP_AggStep
61737. For each CTE in the WITH clause....
61738. Open a cursor to iterate through the contents of the oldest nSeg ** indexes of absolute level iAbsLevel. If this cursor is opened using ** the 'hint' parameters, it is possible that there are less than nSeg ** segments available in level iAbsLevel. In this case, no work is ** done on iAbsLevel - fall through to the next iteration of the loop ** to start work on some other level.
61739. AS
61740. ** The main routine for background threads that populate aFile[1] of ** multi-threaded IncrMerger objects.
61741. Token to query for
61742. P4 is a 64-bit signed integer
61743. Size of the SHM filename in bytes
61744. ** A tree of these objects forms the RHS of a MATCH operator. ** ** If Fts3Expr.eType is FTSQUERY_PHRASE and isLoaded is true, then aDoclist ** points to a malloced buffer, size nDoclist bytes, containing the results ** of this phrase query in FTS3 doclist format. As usual, the initial ** "Length" field found in doclists stored on disk is omitted from this ** buffer. ** ** Variable aMI is used only for FTSQUERY_NEAR nodes to store the global ** matchinfo data. If it is not

NULL, it points to an array of size nCol*3, ** where nCol is the number of columns in the queried FTS table. The array ** is populated as follows: ***
aMII[iCol*3 + 0] = Undefined ** aMII[iCol*3 + 1] = Number of occurrences ** aMII[iCol*3 + 2] = Number of rows containing at least one instance *** The aMI array is allocated using sqlite3_malloc(). It should be freed ** when the expression node is.

61745. Before calling sqlite3ExprDelete(), set the EP_Static flag. This ** prevents ExprDelete() from deleting the Expr structure itself, ** allowing it to be repopulated by the memcpy() on the following line. ** The pExpr->u.zToken might point into memory that will be freed by the ** sqlite3DbFree(db, pDup) on the last line of this block, so be sure to ** make a copy of the token before doing the sqlite3DbFree().

61746. Results of the query

61747. Newly allocated VFS

61748. Open a read-only cursor, execute the OP_Count, close the cursor.

61749. Try to flatten subqueries in the FROM clause up into the main query

61750. The double-precision datatype used by RTree depends on the ** SQLITE_RTREE_INT_ONLY compile-time option.

61751. EVIDENCE-OF: R-64852-21591 The sqlite3_backup object is created by a ** call to sqlite3_backup_init() and is destroyed by a call to ** sqlite3_backup_finish().

61752. ** This function is called to select the tokens (if any) that will be ** deferred. The array aTC[] has already been populated when this is ** called. *** This function is called once for each AND/NEAR cluster in the ** expression. Each invocation determines which tokens to defer within ** the cluster with root node pRoot. See comments above the definition ** of struct Fts3TokenAndCost for more details. *** If no error occurs, SQLITE_OK is returned and sqlite3Fts3DeferToken() ** called on each token to defer. Otherwise, an SQLite error code is ** returned.

61753. A slot for the record has already been allocated in the ** SQLITE_MASTER table. We just need to update that slot with all ** the information we've collected.

61754. A single node of parsed JSON

61755. total space to allocate

61756. Offset to start writing file at

61757. **comment:** We have to initialize zBuf to prevent valgrind from reporting ** errors. The reports issued by valgrind are incorrect - we would ** prefer that the randomness be increased by making use of the ** uninitialized space in zBuf - but valgrind errors tend to worry ** some users. Rather than argue, it seems easier just to initialize ** the whole array and silence valgrind, even if that means less randomness ** in the random seed. *** When testing, initializing zBuf[] to zero is all we do. That means ** that we always use the same random number sequence. This makes the ** tests repeatable.

label: code-design

61758. Size of the pCell cell in bytes

61759. Number of subqueries that the label is moving

61760. Number of bytes to copy

61761. Need to sqlite3DbFree() Expr.zToken

61762. don't break the lock if the host id doesn't match

61763. Counter for IN constraints

61764. Value is a BLOB

61765. ** Execute the statement pStmt, either until a row of data is ready, the ** statement is completely executed or an error occurs. *** This routine implements the bulk of the logic behind the sqlite_step() ** API. The only thing omitted is the automatic recompile if a ** schema change has occurred. That detail is handled by the ** outer sqlite3_step() wrapper procedure.

61766. xRollbackTo

61767. This call is to open a *-wal file. Instead, open the *-oal. This ** code ensures that the string passed to xOpen() is terminated by a ** pair of '\0' bytes in case the VFS attempts to extract a URI ** parameter from it.

61768. If isIgnoreErrors is true, then a table is being dropped. In this ** case SQLite runs a "DELETE FROM xxx" on the table being dropped ** before actually dropping it in order to check FK constraints. ** If the parent table of an FK constraint on the current table is ** missing, behave as if it is empty. i.e. decrement the relevant ** FK counter for each row of the current table with non-NULL keys.

61769. ** This function is called to configure the RtreeConstraint object passed ** as the second argument for a MATCH constraint. The value passed as the ** first argument to this function is the right-hand operand to the MATCH ** operator.

61770. Name of the mmapped file

61771. IMP: R-37434-19929 Abs(X) returns NULL if X is NULL.

61772. Determine the number of phrases in the query

61773. Suppress errors if the trigger already exists

61774. Pointer to HighlightContext object

61775. ** Implementation of the SQL scalar function for accessing the underlying ** hash table. This function may be called as follows: *** SELECT <function-name>(<key-name>); ** SELECT <function-name>(<key-name>, <pointer>); *** where <function-name> is the name passed as the second argument ** to the sqlite3Fts3InitHashTable() function (e.g. 'fts3_tokenizer'). ** If the <pointer> argument is specified, it must be a blob value ** containing a pointer to be stored as the hash data corresponding ** to the string <key-name>. If <pointer> is not specified, then ** the string <key-name> must already exist in the has table. Otherwise, ** an error is returned. *** Whether or not the <pointer> argument is specified, the value returned ** is a blob containing the pointer stored as the hash data corresponding ** to string <key-name> (after the hash-table is updated, if applicable).

61776. ** NOTE: We are dealing with a relative path name and the data ** directory has been set. Therefore, use it as the basis ** for converting the relative path name to an absolute ** one by prepending the data directory and a backslash.

61777. Largest serialized key seen so far

61778. FROM clause element to resolve

61779. ** Free all entries in the pCsr->pDeferred list. Entries are added to ** this list using sqlite3Fts3DeferToken().

61780. This INSERT command is part of a VACUUM operation, which guarantees ** that the destination table is empty. If all indexed columns use ** collation sequence BINARY, then it can also be assumed that the ** index will be populated by inserting keys in strictly sorted ** order. In this case, instead of seeking within the b-tree as part ** of every OP_IdxInsert opcode, an OP_Last is added before the ** OP_IdxInsert to seek to the point within the b-tree where each key ** should be inserted. This is faster. *** If any of the indexed columns use a collation sequence other than ** BINARY, this optimization is disabled. This is because the user ** might change the definition of a collation sequence and then run ** a VACUUM command. In that case keys may not be written in strictly ** sorted order.

61781. OUT: True to remove PK row and retry

61782. The record to verify

61783. ** The unicode() function. Return the integer unicode code-point value ** for the first character of the input string.

61784. 310

61785. Version 3.10.0 and later

61786. This is a new term. Append a term to the output segment.

61787. If running in direct mode, write the contents of page 1 to the file.

61788. If the above search did not find a BtLock struct associating Btree p ** with table iTable, allocate one and link it into the list.

61789. ** PRAGMA [schema.]schema_version ** PRAGMA [schema.]schema_version = <integer> *** PRAGMA [schema.]user_version ** PRAGMA [schema.]user_version = <integer> *** PRAGMA [schema.]freelist_count ** PRAGMA [schema.]data_version ** PRAGMA [schema.]application_id ** PRAGMA [schema.]application_id = <integer> ** The pragma's schema_version and user_version are used to set or get ** the value of the schema-version and user-version, respectively. Both ** the schema-version and the user-version are 32-bit signed integers ** stored in the database header. *** The schema-cookie is usually only manipulated internally by SQLite. It ** is incremented by SQLite whenever the database schema is modified (by ** creating or dropping a table or index). The schema version is used by ** SQLite each time a query is executed to ensure that the internal cache ** of the schema used when compiling the SQL query matches the schema of ** the database against which the compiled query is actually executed. ** Subverting this mechanism by using "PRAGMA schema_version" to modify ** the schema-version is potentially dangerous and may lead to program ** crashes or database corruption. Use with caution! *** The user-version is not used internally by SQLite. It may be used by ** applications for any purpose.

61790. ** Append a doclist to buffer pBuf. *** This function assumes that space within the buffer has already been ** allocated.

61791. ** Return the amount of memory currently checked out.

61792. pPager->pBusyHandlerArg = 0;

61793. RHS is a string

61794. Database connection for this cursor

61795. Initialize iLast, the "lastest" rowid any iterator points to. If the ** iterator skips through rowids in the default ascending order, this means ** the maximum rowid. Or, if the iterator is "ORDER BY rowid DESC", then it ** means the minimum rowid.

61796. Update the assumed sector-size to match the value used by ** the process that created this journal. If this journal was ** created by a process other than this one, then this routine ** is being called from within pager_playback(). The local value ** of Pager.sectorSize is restored at the end of that routine.

61797. Index of changed index field

61798. ** Compare key1 (buffer pKey1, size nKey1 bytes) with key2 (buffer pKey2, ** size nKey2 bytes). Use (pTask->pKeyInfo) for the collation sequences ** used by the comparison. Return the result of the comparison. ** ** If IN/OUT parameter *pbKey2Cached is true when this function is called, ** it is assumed that (pTask->pUnpacked) contains the unpacked version ** of key2. If it is false, (pTask->pUnpacked) is populated with the unpacked ** version of key2 and *pbKey2Cached set to true before returning. ** ** If an OOM error is encountered, (pTask->pUnpacked->error_rc) is set ** to SQLITE_NOMEM.

61799. Read this many bytes

61800. OUT: UPDATE statement handle

61801. **comment:** Database connection, for malloc()
label: code-design

61802. Sequence number of leaf within tree

61803. 690

61804. Boolean array - true if column is in PK

61805. List terminator

61806. A table is not writable under the following circumstances: ** ** 1) It is a virtual table and no implementation of the xUpdate method ** has been provided, or ** 2) It is a system table (i.e. sqlite_master), this call is not ** part of a nested parse and writable_schema pragma has not ** been specified. ** ** In either case leave an error message in pParse and return non-zero.

61807. Next position

61808. ** Allocate nBytes of memory.

61809. **comment:** ** sqlite3_snprintf() works like sprintf() except that it ignores the ** current locale settings. This is important for SQLite because we ** are not able to use a "," as the decimal point in place of "." as ** specified by some locales. ** ** Oops: The first two arguments of sqlite3_snprintf() are backwards ** from the snprintf() standard. Unfortunately, it is too late to change ** this without breaking compatibility, so we just have to live with the ** mistake. ** ** sqlite3_vsnprintf() is the varargs version.
label: code-design

61810. OUT: Number of entries in output array

61811. Honor DESC

61812. Use full fsync for checkpoint

61813. ** Return the index of a column in a table. Return -1 if the column ** is not contained in the table.

61814. ** Return a static string that describes the kind of error specified in the ** argument.

61815. ** If a prior call to sqlite3GenerateIndexKey() generated a jump-over label ** because it was a partial index, then this routine should be called to ** resolve that label.

61816. A named index with an explicit CREATE INDEX statement

61817. Or here, if bUseThreads==0

61818. First slot in %_segments written

61819. The Source list to check and modify

61820. If the SET_FULLSYNC macro is not defined above, then make it ** a no-op

61821. ** Values for Fts5Cursor.csflags

61822. True if this call is part of a bulk load

61823. ** An instance of the following object is used to record information about ** the ORDER BY (or GROUP BY) clause of query is being coded.

61824. First term on new page

61825. ** isConsonant() and isVowel() determine if their first character in ** the string they point to is a consonant or a vowel, according ** to Porter rules. ** ** A consonant is any letter other than 'a', 'e', 'i', 'o', or 'u'. ** 'Y' is a consonant unless it follows another consonant, ** in which case it is a vowel. ** ** In these routine, the letters are in reverse order. So the 'y' rule ** is that 'y' is a consonant unless it is followed by another ** consonant.

61826. Opcode: NotExists P1 P2 P3 * * * Synopsis: intkey=r[P3] ** ** P1 is the index of a cursor open on an SQL table btree (with integer ** keys). P3 is an integer rowid. If P1 does not contain a record with ** rowid P3 then jump immediately to P2. Or, if P2 is 0, raise an ** SQLITE_CORRUPT error. If P1 does contain a record with rowid P3 then ** leave the cursor pointing at that record and fall through to the next ** instruction. ** ** The OP_SeekRowid opcode performs the same operation but also allows the ** P3 register to contain a non-integer value, in which case the jump is ** always taken. This opcode requires that P3 always contain an integer. ** ** The OP_NotFound opcode performs the same operation on index btrees ** (with arbitrary multi-value keys). ** ** This opcode leaves the cursor in a state where it cannot be advanced ** in either direction. In other words, the Next and Prev opcodes will ** not work following this opcode. ** ** See also: Found, NotFound, NoConflict, SeekRowid

61827. ** zIn is a UTF-16 encoded unicode string at least nChar characters long. ** Return the number of bytes in the first nChar unicode characters ** in pZ. nChar must be non-negative.

61828. TK_UPDATE, TK_INSERT or TK_DELETE

61829. There are now 4 possibilities: ** ** 1. uPattern is an unescaped match-all character "%", ** 2. uPattern is an unescaped match-one character "_", ** 3. uPattern is an unescaped escape character, or ** 4. uPattern is to be handled as an ordinary character

61830. PRAGMA synchronous=NORMAL

61831. Like optimization range constraints always occur in pairs

61832. ** Implementation of a special SQL scalar function for testing tokenizers ** designed to be used in concert with the Tcl testing framework. This ** function must be called with two or more arguments: ** ** SELECT <function-name>(<key-name>, ..., <input-string>); ** ** where <function-name> is the name passed as the second argument ** to the sqlite3Fts3InitHashTable() function (e.g. 'fts3_tokenizer') ** concatenated with the string '_test' (e.g. 'fts3_tokenizer_test'). ** ** The return value is a string that may be interpreted as a Tcl ** list. For each token in the <input-string>, three elements are ** added to the returned list. The first is the token position, the ** second is the token text (folded, stemmed, etc.) and the third is the ** substring of <input-string> associated with the token. For example, ** using the built-in "simple" tokenizer: ** ** SELECT fts_tokenizer_test('simple', 'I don't see how'); ** ** will return the string: ** ** "{0 i 1 dont don't 2 see see 3 how how}" **

61833. True if resizeIndexObject() has been called

61834. Store result in a memory cell

61835. Size of buffer at zNode

61836. State: ** SEMI WS OTHER EXPLAIN CREATE TEMP TRIGGER END

61837. ** Set the pIter->bEof variable based on the state of the sub-iterators.

61838. ORDER BY processing for min() func

61839. ** The testcase() macro is only used by the amalgamation. If undefined, ** make it a no-op.

61840. number of sizeof(int) objects needed for zName

61841. The entire tree fits on the root node. Write it to the segdir table.

61842. Number of pages in apNew[]

61843. ** sqlite3_test_control(BENIGN_MALLOC_HOOKS, xBegin, xEnd) ** ** Register hooks to call to indicate which malloc() failures ** are benign.

61844. Create a new tokenizer

61845. FTS5_VOCAB_COL or ROW

61846. ** This function is called to rollback a transaction on a WAL database.

61847. Forward declarations of structures.

61848. Selects no more than one row

61849. **comment:** If a malloc() failure occurred in sqlite3HashInsert(), it will ** return the pColl pointer to be deleted (because it wasn't added ** to the hash table).
label: code-design

61850. open file object for the database file

61851. Define the fts5yytestcase() macro to be a no-op if is not already defined ** otherwise. ** ** Applications can choose to define fts5yytestcase() in the %include section ** to a macro that can assist in verifying code coverage. For production ** code the fts5yytestcase() macro should be turned off. But it is useful ** for

testing.

61852. Number of pages in database after autovacuuming

61853. Check that the new column is not specified as PRIMARY KEY or UNIQUE. ** If there is a NOT NULL constraint, then the default value for the ** column must not be NULL.

61854. ** Set the collation function of the most recently parsed table column ** to the CollSeq given.

61855. Existence of VDBE checked by caller

61856. Objects

61857. Chomp all segments opened by this cursor

61858. If there are dirty pages in the page cache with page numbers greater ** than Pager.dbSize, this means sqlite3PagerTruncateImage() was called to ** make the file smaller (presumably by auto-vacuum code). Do not write ** any such pages to the file. ** ** Also, do not write out any page that has the PGHDR_DONT_WRITE flag ** set (set by sqlite3PagerDontWrite()).

61859. ***** End of sqliteInt.h *****

61860. unreachable code

61861. How to dispose of results

61862. Default number of tokens in snippet

61863. Beginning of the body of this loop

61864. ** Return true if the prepared statement is in need of being reset.

61865. Pointer to doclist

61866. Schema name of attached database

61867. Fallback token

61868. **comment:** This index implies that the DISTINCT qualifier is redundant.

label: code-design

61869. These are used by 'col' tables only

61870. ** pFile is a file that has been opened by a prior xOpen call. dbPath ** is a string buffer at least MAXPATHLEN+1 characters in size. ** ** This routine find the filename associated with pFile and writes it ** int dbPath.

61871. **comment:** ** Overview: ** ** The %_data table contains all the FTS indexes for an FTS5 virtual table. ** As well as the main term index, there may be up to 31 prefix indexes. ** The format is similar to FTS3/4, except that: ** ** * all segment b-tree leaf data is stored in fixed size page records ** (e.g. 1000 bytes). A single doclist may span multiple pages. Care is ** taken to ensure it is possible to iterate in either direction through ** the entries in a doclist, or to seek to a specific entry within a ** doclist, without loading it into memory. ** ** * large doclists that span many pages have associated "doclist index" ** records that contain a copy of the first rowid on each page spanned by ** the doclist. This is used to speed up seek operations, and merges of ** large doclists with very small doclists. ** ** * extra fields in the "structure record" record the state of ongoing ** incremental merge operations. **

label: code-design

61872. **comment:** The PGHDR_DIRTY bit is set above when the page was added to the dirty-list ** and before writing the page into the rollback journal. Wait until now, ** after the page has been successfully journaled, before setting the ** PGHDR_WRITEABLE bit that indicates that the page can be safely modified.

label: code-design

61873. Default lookaside buffer size

61874. Initialize the contents of sCtx.aTerm[] for column iCol. There is ** no way that this operation can fail, so the return code from ** fts3ExprIterate() can be discarded.

61875. The table being referenced

61876. sqlite3_finalize() return code

61877. If the calls to xBestIndex() in the above loop did not find a plan ** that requires no source tables at all (i.e. one guaranteed to be ** usable), make a call here with all source tables disabled

61878. The input string A

61879. Number of pages written to journal

61880. xOpen

61881. defined(SQLITE_TEST) || defined(SQLITE_DEBUG)

61882. Iterate through all tokens in this AND/NEAR cluster, in ascending order ** of the number of overflow pages that will be loaded by the pager layer ** to retrieve the entire doclist for the token from the full-text index. ** Load the doclists for tokens that are either: ** ** a. The cheapest token in the entire query (i.e. the one visited by the ** first iteration of this loop), or ** ** b. Part of a multi-token phrase. ** ** After each token doclist is loaded, merge it with the others from the ** same phrase and count the number of documents that the merged doclist ** contains. Set variable "nMinEst" to the smallest number of documents in ** any phrase doclist for which 1 or more token doclists have been loaded. ** Let nOther be the number of other phrases for which it is certain that ** one or more tokens will not be deferred. ** ** Then, for each token, defer it if loading the doclist would result in ** loading N or more overflow pages into memory, where N is computed as: ** ** (nMinEst + 4^nOther - 1) / (4^nOther)

61883. ** Move the cursor down to the left-most leaf entry beneath the ** entry to which it is currently pointing. ** ** The left-most leaf is the one with the smallest key - the first ** in ascending order.

61884. Internal logic error in SQLite

61885. Parse context

61886. ** The macro USEFETCH is true if we are allowed to use the xFetch and xUnfetch ** interfaces to access the database using memory-mapped I/O.

61887. Cannot optimize overly large ORDER BYs

61888. Opcode: Init P1 P2 * P4 * ** Synopsis: Start at P2 ** ** Programs contain a single instance of this opcode as the very first ** opcode. ** ** If tracing is enabled (by the sqlite3_trace()) interface, then ** the UTF-8 string contained in P4 is emitted on the trace callback. ** Or if P4 is blank, use the string returned by sqlite3_sql(). ** ** If P2 is not zero, jump to instruction P2. ** ** Increment the value of P1 so that OP_Once opcodes will jump the ** first time they are evaluated for this run.

61889. Access to the unixShmNode object is serialized by the caller

61890. Index of next free chunk

61891. ** Register built-in functions used to help read ANALYZE data.

61892. Number of indices that need updating

61893. **comment:** ** This function is used to allocate an Fts3SegReader that iterates through ** a subset of the terms stored in the Fts3Table.pendingTerms array. ** ** If the isPrefixIter parameter is zero, then the returned SegReader iterates ** through each term in the pending-terms table. Or, if isPrefixIter is ** non-zero, it iterates through each term and its prefixes. For example, if ** the pending terms hash table contains the terms "sqlite", "mysql" and ** "firebird", then the iterator visits the following 'terms' (in the order ** shown): ** ** f f1 fir fire firebi firebir firebird ** m my mys mysq mysql ** s sq sql sqli sqlite ** Whereas if isPrefixIter is zero, the terms visited are: ** ** firebird mysql sqlite

label: code-design

61894. turn off proxy locking - not supported. If support is added for ** switching proxy locking mode off then it will need to fail if ** the journal mode is WAL mode.

61895. ** CAPI3REF: Testing Interface Operation Codes ** ** These constants are the valid operation code parameters used ** as the first argument to [sqlite3_test_control()]. ** ** These parameters and their meanings are subject to change ** without notice. These values are for testing purposes only. ** Applications should not use any of these parameters or the ** [sqlite3_test_control()] interface.

61896. 90

61897. Delete this when closing file

61898. The hash function

61899. ** Locate a pragma in the aPragmaName[] array.

61900. cmd ::= DROP TRIGGER ifexists fullname

61901. True to hash the new.* PK

61902. ** The number of different kinds of things that can be limited ** using the sqlite3_limit() interface.

61903. **comment:** The "xxx" in the name "xxx.yyy" or "xxx"

label: code-design

61904. Size of buffer pToken in bytes

61905. Do not omit these constraints

61906. Search through the cursors list of Fts5Auxdata objects for one that ** corresponds to the currently executing auxiliary function.
61907. Mark all dirty list pages as clean
61908. True for SQLITE_PRINTF_SQLFUNC
61909. Invoke the virtual table xBestIndex() method
61910. True for native byte-order checksums
61911. Hash table iterator
61912. xSavepoint
61913. 29
61914. Optional database for lookaside. Can be NULL
61915. Set the OPFLAG_USESEEKRESULT flag if either (a) there are no REPLACE ** constraints or (b) there are no triggers and this table is not a ** parent table in a foreign key constraint. It is safe to set the ** flag in the second case as if any REPLACE constraint is hit, an ** OP_Delete or OP_IdxDelete instruction will be executed on each ** cursor that is disturbed. And these instructions both clear the ** VdbeCursor.seekResult variable, disabling the OPFLAG_USESEEKRESULT ** functionality.
61916. ***** Begin %include sections from the grammar *****
61917. **comment:** ** Use a macro to replace memcpy() if compiled with SQLITE_INLINE_MEMCPY. ** This allows better measurements of where memcpy() is used when running ** cachegrind. But this macro version of memcpy() is very slow so it ** should not be used in production. This is a performance measurement ** hack only.
label: code-design
61918. TK_COLUMN: column index. -1 for rowid. ** TK_VARIABLE: variable number (always >= 1). ** TK_SELECT_COLUMN: column of the result vector
61919. Address of this instruction
61920. Name of table or index
61921. ***** Begin file loadext.c *****
61922. **comment:** ** CAPI3REF: OS Interface File Virtual Methods Object *** Every file opened by the [sqlite3_vfs.xOpen] method populates an ** [sqlite3_file] object (or, more commonly, a subclass of the ** [sqlite3_file] object) with a pointer to an instance of this object. ** This object defines the methods used to perform various operations ** against the open file represented by the [sqlite3_file] object. *** If the [sqlite3_vfs.xOpen] method sets the sqlite3_file.pMethods element ** to a non-NULL pointer, then the sqlite3_io_methods.xClose method ** may be invoked even if the [sqlite3_vfs.xOpen] reported that it failed. The ** only way to prevent a call to xClose following a failed [sqlite3_vfs.xOpen] ** is for the [sqlite3_vfs.xOpen] to set the sqlite3_file.pMethods element ** to NULL. *** The flags argument to xSync may be one of [SQLITE_SYNC_NORMAL] or ** [SQLITE_SYNC_FULL]. The first choice is the normal fsync(). ** The second choice is a Mac OS X style fullsync. The [SQLITE_SYNC_DATAONLY] ** flag may be ORed in to indicate that only the data of the file ** and not its inode needs to be synced. *** The integer values to xLock() and xUnlock() are one of ** ** [SQLITE_LOCK_NONE], ** [SQLITE_LOCK_SHARED], ** [SQLITE_LOCK_RESERVED], ** [SQLITE_LOCK_PENDING], or ** [SQLITE_LOCK_EXCLUSIVE]. ** ** xLock() increases the lock. xUnlock() decreases the lock. ** The xCheckReservedLock() method checks whether any database connection, ** either in this process or in some other process, is holding a RESERVED, ** PENDING, or EXCLUSIVE lock on the file. It returns true ** if such a lock exists and false otherwise. *** ** The xFileControl() method is a generic interface that allows custom ** VFS implementations to directly control an open file using the ** [sqlite3_file_control()] interface. The second "op" argument is an ** integer opcode. The third argument is a generic pointer intended to ** point to a structure that may contain arguments or space in which to ** write return values. Potential uses for xFileControl() might be ** functions to enable blocking locks with timeouts, to change the ** locking strategy (for example to use dot-file locks), to inquire ** about the status of a lock, or to break stale locks. The SQLite ** core reserves all opcodes less than 100 for its own use. ** A [file control opcodes | list of opcodes] less than 100 is available. ** Applications that define a custom xFileControl method should use opcodes ** greater than 100 to avoid conflicts. VFS implementations should ** return [SQLITE_NOTFOUND] for file control opcodes that they do not ** recognize. *** ** The xSectorSize() method returns the sector size of the ** device that underlies the file. The sector size is the ** minimum write that can be performed without disturbing ** other bytes in the file. The xDeviceCharacteristics() ** method returns a bit vector describing behaviors of the ** underlying device: *** ** [SQLITE_IOCAP_ATOMIC] ** [SQLITE_IOCAP_ATOMIC512] ** [SQLITE_IOCAP_ATOMIC1K] ** [SQLITE_IOCAP_ATOMIC2K] ** [SQLITE_IOCAP_ATOMIC4K] ** [SQLITE_IOCAP_ATOMIC8K] ** [SQLITE_IOCAP_ATOMIC16K] ** [SQLITE_IOCAP_ATOMIC32K] ** [SQLITE_IOCAP_ATOMIC64K] ** [SQLITE_IOCAP_SAFE_APPEND] ** [SQLITE_IOCAP_SEQUENTIAL] ** [SQLITE_IOCAP_UNDELETABLE_WHEN_OPEN] ** [SQLITE_IOCAP_POWERSAFE_OVERWRITE] ** [SQLITE_IOCAP_IMMUTABLE] ** ** ** The SQLITE_IOCAP_ATOMIC property means that all writes of ** any size are atomic. The SQLITE_IOCAP_ATOMICCnnn values ** mean that writes of blocks that are nnn bytes in size and ** are aligned to an address which is an integer multiple of ** nnn are atomic. The SQLITE_IOCAP_SAFE_APPEND value means ** that when data is appended to a file, the data is appended ** first then the size of the file is extended, never the other ** way around. The SQLITE_IOCAP_SEQUENTIAL property means that ** information is written to disk in the same order as calls ** to xWrite(). ** ** If xRead() returns SQLITE_IOERR_SHORT_READ it must also fill ** in the unread portions of the buffer with zeros. A VFS that ** fails to zero-fill short reads might seem to work. However, ** failure to zero-fill short reads will eventually lead to ** database corruption.
label: code-design
61923. Obtain all b-tree mutexes before making any calls to BtreeRollback(). ** This is important in case the transaction being rolled back has ** modified the database schema. If the b-tree mutexes are not taken ** here, then another shared-cache connection might sneak in between ** the database rollback and schema reset, which can cause false ** corruption reports in some cases.
61924. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains C code routines that are called by the SQLite parser ** when syntax rules are reduced. The routines in this file handle the ** following kinds of SQL syntax: *** CREATE TABLE ** DROP TABLE ** CREATE INDEX ** DROP INDEX ** creating ID lists ** BEGIN TRANSACTION ** COMMIT ** ROLLBACK
61925. Cursor number for the table
61926. **comment:** ** A macro used for invoking the codec if there is one
label: code-design
61927. Mask of all ORDER BY terms
61928. All the allocations succeeded. Put the expression object together.
61929. ** Different Unix systems declare open() in different ways. Same use ** open(const char*,int,mode_t). Others use open(const char*,int,...). ** The difference is important when using a pointer to the function. *** ** The safest way to deal with the problem is to always use this wrapper ** which always has the same well-defined interface.
61930. The P4 operand as an integer
61931. In a NATURAL join, omit the join columns from the ** table to the right of the join
61932. Add the entries to the stat3 or stat4 table.
61933. Information about each nested IN operator
61934. Replacement values
61935. True for rowid tables. False for indexes
61936. ** The number of rowset entries per allocation chunk.
61937. New chunk is required to extend the file.
61938. **comment:** The VFS pointer. Not used
label: code-design
61939. Return code from sqlite3_prepare()
61940. Set or get the suggested spill-size for the specified pager-cache. ** ** The spill-size is the minimum number of pages in cache before the cache ** will attempt to spill dirty pages by calling xStress.
61941. Type name follows column name
61942. ** Advance iterator pIter until it points to a value equal to or laster ** than the initial value of *pLast. If this means the iterator points ** to a value laster than ** *pLast, update *pLast to the new lastest value. *** If the iterator reaches EOF, set *pbEof to true before returning. If ** an error occurs, set *pRc to an error code. If either *pbEof or *pRc ** are set, return a non-zero value. Otherwise, return zero.
61943. **comment:** TERM_xxx bit flags. See below
label: code-design
61944. Same as READ_UTF8() above but without the zTerm parameter. ** For this routine, we assume the UTF8 string is always zero-terminated.

61945. ** The type used to represent a page number. The first page in a file ** is called page 1. 0 is used to represent "not a page".
61946. Page number is 1 or more
61947. True if journal file exists
61948. Number of elements in argv array
61949. ** Evaluate a view and store its result in an ephemeral table. The ** pWhere argument is an optional WHERE clause that restricts the ** set of rows in the view that are to be added to the ephemeral table.
61950. Cache configuration parameters. Page size (szPage) and the purgeable ** flag (bPurgeable) are set when the cache is created. nMax may be ** modified at any time by a call to the pcache1Cachesize() method. ** The PGroup mutex must be held when accessing nMax.
61951. ** Copy nPage pages from the source b-tree to the destination.
61952. If the output is destined for a temporary table, open that table.
61953. Size of apSegment array
61954. This is the only way out of this procedure. We have to ** release the mutexes on btrees that were acquired at the ** top.
61955. This routine is only called by btree immediately after creating ** the Pager object. There has not been an opportunity to transition ** to WAL mode yet.
61956. Allocate a temporary buffer to store the fully qualified file ** name for the temporary file. If this fails, we cannot continue.
61957. Opcode: BitOr P1 P2 P3 * * *** Synopsis: r[P3]=r[P1]|r[P2] *** Take the bit-wise OR of the values in register P1 and P2 and ** store the result in register P3. ** If either input is NULL, the result is NULL.
61958. (2b)
61959. Mask of locks to take or release
61960. ** This routine runs at the end of parsing a CREATE TABLE statement that ** has a WITHOUT ROWID clause. The job of this routine is to convert both ** internal schema data structures and the generated VDBE code so that they ** are appropriate for a WITHOUT ROWID table instead of a rowid table. ** Changes include: ** ** (1) Set all columns of the PRIMARY KEY schema object to be NOT NULL. ** (2) Convert the OP_CreateTable into an OP_CreateIndex. There is ** no rowid btree for a WITHOUT ROWID. Instead, the canonical ** data storage is a covering index btree. ** (3) Bypass the creation of the sqlite_master table entry ** for the PRIMARY KEY as the primary key index is now ** identified by the sqlite_master table entry of the table itself. ** (4) Set the Index.num of the PRIMARY KEY Index object in the ** schema to the rootpage from the main table. ** (5) Add all table columns to the PRIMARY KEY Index object ** so that the PRIMARY KEY is a covering index. The surplus ** columns are part of KeyInfo.nXField and are not used for ** sorting or lookup or uniqueness checks. ** (6) Replace the rowid tail on all automatically generated UNIQUE ** indices with the PRIMARY KEY columns. *** For virtual tables, only (1) is performed.
61961. Store the name and dectype
61962. ** The following function extracts information on the current change ** from a changeset iterator. It may only be called after changeset_next() ** has returned SQLITE_ROW.
61963. ** Select the fragment of text consisting of nFragment contiguous tokens ** from column iCol that represent the "best" snippet. The best snippet ** is the snippet with the highest score, where scores are calculated ** by adding: ** ** (a) +1 point for each occurrence of a matchable phrase in the snippet. ** ** (b) +100 points for the first occurrence of each matchable phrase in ** the snippet for which the corresponding mCovered bit is not set. ** ** The selected snippet parameters are stored in structure *pFragment before ** returning. The score of the selected snippet is stored in *piScore ** before returning.
61964. Rh input changeset
61965. Return code
61966. **comment:** ** SQLITE_MAX_U32 is a u64 constant that is the maximum u64 value ** that can be stored in a u32 without loss of data. The value ** is 0x00000000ffffffffff. But because of quirks of some compilers, we ** have to specify the value in the less intuitive manner shown:
label: code-design
61967. Num of col names. 0 means use pragma name
61968. ** Set the 32-bit cookie value stored at the start of all structure ** records to the value passed as the second argument. ** ** Return SQLITE_OK if successful, or an SQLite error code if an error ** occurs.
61969. ***** Start of porter stemmer implementation.
61970. 18..1f,
61971. Next buffer in the list of free buffers
61972. Step 5. If we do not care about the difference between NULL and ** FALSE, then just return false.
61973. Number of valid bytes of data in a[]
61974. if i==0 and iCol==0, then record pRec is smaller than all samples ** in the aSample[] array. Otherwise, if (iCol>0) then pRec must ** be greater than or equal to the (iCol) field prefix of sample i. ** If (i>0), then pRec must also be greater than sample (i-1).
61975. ** The block of page numbers associated with the first hash-table in a ** wal-index is smaller than usual. This is so that there is a complete ** hash-table on each aligned 32KB page of the wal-index.
61976. **comment:** ** This function is used by the implementation of the IN (...) operator. ** The pX parameter is the expression on the RHS of the IN operator, which ** might be either a list of expressions or a subquery. ** ** The job of this routine is to find or create a b-tree object that can ** be used either to test for membership in the RHS set or to iterate through ** all members of the RHS set, skipping duplicates. ** ** A cursor is opened on the b-tree object that is the RHS of the IN operator ** and pX->iTable is set to the index of that cursor. ** ** The returned value of this function indicates the b-tree type, as follows: ** **
IN_INDEX_ROWID - The cursor was opened on a database table. ** IN_INDEX_INDEX_ASC - The cursor was opened on an ascending index. **
IN_INDEX_INDEX_DESC - The cursor was opened on a descending index. ** IN_INDEX_EPH - The cursor was opened on a specially created and ** populated ephemeral table. ** IN_INDEX_NOOP - No cursor was allocated. The IN operator must be ** implemented as a sequence of comparisons. ** ** An existing b-tree might be used if the RHS expression pX is a simple ** subquery such as: ** ** SELECT <column1>, <column2>... FROM <table> ** ** If the RHS of the IN operator is a list or a more complex subquery, then ** an ephemeral table might need to be generated from the RHS and then ** pX->iTable made to point to the ephemeral table instead of an ** existing table. ** ** The inFlags parameter must contain exactly one of the bits ** IN_INDEX_MEMBERSHIP or IN_INDEX_LOOP. If inFlags contains ** IN_INDEX_MEMBERSHIP, then the generated table will be used for a ** fast membership test. When the IN_INDEX_LOOP bit is set, the ** IN index will be used to loop over all values of the RHS of the ** IN operator. ** ** When IN_INDEX_LOOP is used (and the b-tree will be used to iterate ** through the set members) then the b-tree must not contain duplicates. ** An ephemeral table must be used unless the selected columns are guaranteed ** to be unique - either because it is an INTEGER PRIMARY KEY or due to ** a UNIQUE constraint or index. ** ** When IN_INDEX_MEMBERSHIP is used (and the b-tree will be used ** for fast set membership tests) then an ephemeral table must ** be used unless <columns> is a single INTEGER PRIMARY KEY column or an ** index can be found with the specified <columns> as its left-most. ** ** If the IN_INDEX_NOOP and IN_INDEX_MEMBERSHIP are both set and ** if the RHS of the IN operator is a list (not a subquery) then this ** routine might decide that creating an ephemeral b-tree for membership ** testing is too expensive and return IN_INDEX_NOOP. In that case, the ** calling routine should implement the IN operator using a sequence ** of Eq or Ne comparison operations. ** ** When the b-tree is being used for membership tests, the calling function ** might need to know whether or not the RHS side of the IN operator ** contains a NULL. If prRhsHasNull is not a NULL pointer and ** if there is any chance that the (...) might contain a NULL value at ** runtime, then a register is allocated and the register number written ** to *prRhsHasNull. If there is no chance that the (...) contains a ** NULL value, then *prRhsHasNull is left unchanged. ** ** If a register is allocated and its location stored in *prRhsHasNull, then ** the value in that register will be NULL if the b-tree contains one or more ** NULL values, and it will be some non-NUL value if the b-tree contains no ** NULL values. ** ** If the aiMap parameter is not NULL, it must point to an array containing ** one element for each column returned by the SELECT statement on the RHS ** of the IN(...) operator. The i'th entry of the array is populated with the ** offset of the index column that matches the i'th column returned by the ** SELECT. For example, if the expression and selected index are: ** ** (?, ?, ?) IN (SELECT a, b, c FROM t1) ** CREATE INDEX i1 ON t1(b, c, a); ** ** then aiMap[] is populated with {2, 0, 1}.
label: code-design
61977. Next in linked list (see above)
61978. Allocate space for the new unixShm object.
61979. ** Return TRUE if the given yDbMask object is empty - if it contains no ** 1 bits. This routine is used by the DbMaskAllZero() and DbMaskNotZero() ** macros when SQLITE_MAX_ATTACHED is greater than 30.
61980. ** An instance of the following structure is allocated for each open ** inode. Or, on LinuxThreads, there is one of these structures for ** each inode opened by each thread. ** ** A single inode can have multiple file descriptors, so each unixFile ** structure contains a pointer to an instance of this object and this ** object keeps a count of the number of unixFile pointing to it.
61981. trigger_cmd ::= insert_cmd INTO trnm idlist_opt select
61982. STEP 3: ** Loop through the entire memory pool. Coalesce adjacent free ** chunks. Recompute the master chunk as the largest free chunk. ** Then try again to satisfy the allocation by carving a piece off ** of the end of the master chunk. This step happens very ** rarely (we hope!)

61983. #include <stdarg.h>
61984. Unpacked version of key
61985. True if the output segment is the oldest
61986. True if IDLIST is in table order
61987. assert(pPager->journalMode==PAGER_JOURNALMODE_MEMORY);
61988. ** Each call to sqlite3_rtree_geometry_callback() or ** sqlite3_rtree_query_callback() creates an ordinary SQLite ** scalar function that is implemented by this routine. *** All this function does is construct an RtreeMatchArg object that ** contains the geometry-checking callback routines and a list of ** parameters to this function, then return that RtreeMatchArg object ** as a BLOB. *** The R-Tree MATCH operator will read the returned BLOB, deserialize ** the RtreeMatchArg object, and use the RtreeMatchArg object to figure ** out which elements of the R-Tree should be returned by the query.
61989. Opcode: VRename P1 * * P4 * * * P4 is a pointer to a virtual table object, an sqlite3_vtab structure. ** This opcode invokes the corresponding xRename method. The value ** in register P1 is passed as the zName argument to the xRename method.
61990. Opcode: Sort P1 P2 * * * * * This opcode does exactly the same thing as OP_Rewind except that ** it increments an undocumented global variable used for testing. *** Sorting is accomplished by writing records into a sorting index, ** then rewinding that index and playing it back from beginning to ** end. We use the OP_Sort opcode instead of OP_Rewind to do the ** rewinding so that the global variable will be incremented and ** regression tests can determine whether or not the optimizer is ** correctly optimizing out sorts.
61991. ** CAPI3REF: Determine If An SQL Statement Writes The Database ** METHOD: sqlite3_stmt ** * ^The sqlite3_stmt_readonly(X) interface returns true (non-zero) if ** and only if the [prepared statement] X makes no direct changes to ** the content of the database file. *** Note that [application-defined SQL functions] or ** [virtual tables] might change the database indirectly as a side effect. ** ^For example, if an application defines a function "eval()" that ** calls [sqlite3_exec()], then the following SQL statement would ** change the database file through side-effects: *** <blockquote><pre> ** SELECT eval('DELETE FROM t1') FROM t2; </pre></blockquote> *** But because the [SELECT] statement does not change the database file ** directly, sqlite3_stmt_readonly() would still return true.)^ *** ^Transaction control statements such as [BEGIN], [COMMIT], [ROLLBACK], ** [SAVEPOINT], and [RELEASE] cause sqlite3_stmt_readonly() to return true, ** since the statements themselves do not actually modify the database but ** rather they control the timing of when other statements modify the ** database. ^The [ATTACH] and [DETACH] statements also cause ** sqlite3_stmt_readonly() to return true since, while those statements ** change the configuration of a database connection, they do not make ** changes to the content of the database files on disk. ** ^The sqlite3_stmt_readonly() interface returns true for [BEGIN] since ** [BEGIN] merely sets internal flags, but the [BEGIN|BEGIN IMMEDIATE] and ** [BEGIN|BEGIN EXCLUSIVE] commands do touch the database and so ** sqlite3_stmt_readonly() returns false for those commands.
61992. ** pTab is a pointer to a Table structure representing a virtual-table. ** Return a pointer to the VTable object used by connection db to access ** this virtual-table, if one has been created, or NULL otherwise.
61993. If condition (a) is not met, assume (b) is true. StructurePromoteTo() ** is a no-op if it is not.
61994. Number of samples
61995. assume positive
61996. 161
61997. ** Once the sorter has been populated by calls to sqlite3VdbeSorterWrite, ** this function is called to prepare for iterating through the records ** in sorted order.
61998. **comment:** ** CAPI3REF: Declare The Schema Of A Virtual Table ** * ^The [xCreate] and [xConnect] methods of a ** [virtual table module] call this interface ** to declare the format (the names and datatypes of the columns) of ** the virtual tables they implement.
label: code-design
61999. IN/OUT: Iterator pointer
62000. ** *pCurrent gets an accurate estimate of the amount of memory used ** to store all prepared statements. ** *pHighwater is set to zero.
62001. Number of phrases in query
62002. If the sqlite_sequence table exists in this database, then update ** it with the new table name.
62003. Value of %_segdir.end_block
62004. SQLITE_STATUS_PARSER_STACK
62005. A 32-bit varint is used to store size information in btrees. ** Objects are rarely larger than 2MiB limit of a 3-byte varint. ** A 3-byte varint is sufficient, for example, to record the size ** of a 1048569-byte BLOB or string. *** We only unroll the first 1-, 2-, and 3- byte cases. The very ** rare larger cases can be handled by the slower 64-bit varint ** routine.
62006. Saved value of db->suppressErr
62007. ** This function is called to obtain a pointer to region iRegion of the ** shared-memory associated with the database file fd. Shared-memory regions ** are numbered starting from zero. Each shared-memory region is szRegion ** bytes in size. *** If an error occurs, an error code is returned and *pp is set to NULL. *** Otherwise, if the isWrite parameter is 0 and the requested shared-memory ** region has not been allocated (by any client, including one running in a ** separate process), then *pp is set to NULL and SQLITE_OK returned. If ** isWrite is non-zero and the requested shared-memory region has not yet ** been allocated, it is allocated by this function. *** If the shared-memory region has already been allocated or is allocated by ** this call as described above, then it is mapped into this processes ** address space (if it is not already), *pp is set to point to the mapped ** memory and SQLITE_OK returned.
62008. Template for VFS
62009. IN/OUT: Docid pointer
62010. ** The xSavepoint method. *** This module does not need to do anything to support savepoints. However, ** it uses this hook to close any open blob handle. This is done because a ** DROP TABLE command - which fortunately always opens a savepoint - cannot ** succeed if there are any open blob handles. i.e. if the blob handle were ** not closed here, the following would fail: ** * BEGIN; ** INSERT INTO rtree... ** DROP TABLE <tablename>; -- Would fail with SQLITE_LOCKED ** COMMIT;
62011. Check if the query is of one of the following forms: *** SELECT min(x) FROM ... ** SELECT max(x) FROM ... ** * If it is, then ask the code in where.c to attempt to sort results ** as if there was an "ORDER ON x" or "ORDER ON x DESC" clause. ** If where.c is able to produce results sorted in this order, then ** add vdbe code to break out of the processing loop after the ** first iteration (since the first iteration of the loop is ** guaranteed to operate on the row with the minimum or maximum ** value of x, the only row required). *** A special flag must be passed to sqlite3WhereBegin() to slightly ** modify behavior as follows: ** * + If the query is a "SELECT min(x)", then the loop coded by ** where.c should not iterate over any values with a NULL value ** for x. ** * + The optimizer code in where.c (the thing that decides which ** index or indices to use) should place a different priority on ** satisfying the 'ORDER BY' clause than it does in other cases. ** Refer to code and comments in where.c for details.
62012. ** Return the sector-size in bytes for an rbuVfs-file.
62013. ** Record the set of tables that satisfy case 3. The set might be ** empty.
62014. Byte offset of end of token within input text
62015. Parent node
62016. Next entry is not on the current page
62017. True once iDelta has been written
62018. ** Each auxiliary function registered with the FTS5 module is represented ** by an object of the following type. All such objects are stored as part ** of the Fts5Global.pAux list.
62019. Write the new value here
62020. Cursor to retrieve value from
62021. OUT: Write the score here
62022. **comment:** ** Move to the next document *** Return SQLITE_OK if successful, or an SQLite error code otherwise. It ** is not considered an error if the query does not match any documents.
label: documentation
62023. ** Rowids for the averages and structure records in the %_data table.
62024. First varint to store in hint
62025. Mask of all non-compound WO_* values
62026. Use the two-part index name to determine the database ** to search for the table. 'Fix' the table name to this db ** before looking up the table.
62027. 1330
62028. ** Store the current contents of the p->nTotalRow and p->aTotalSize[] ** variables in the "averages" record on disk. *** Return SQLITE_OK if successful, or an SQLite error code if an error ** occurs.
62029. ** Generate code for a BETWEEN operator. *** x BETWEEN y AND z ** * The above is equivalent to *** x>=y AND x<=z ** ** Code it as such, taking care to do the common subexpression ** elimination of x. *** The xJumpIf parameter determines details: *** NULL: Store the boolean result in reg[dest] **

sqlite3ExprIfTrue: Jump to dest if true ** sqlite3ExprIfFalse: Jump to dest if false *** The jumpIfNull parameter is ignored if xJumpIf is NULL.
62030. Delete any foreign keys attached to this table.
62031. Index of the database containing the table to lock
62032. Backup is not possible if the page size of the destination is changing ** and a codec is in use.
62033. ** Argument is a pointer to an Fts5Data structure that contains a ** leaf page.
62034. Size of apVal[] array
62035. ** Resolve label "x" to be the address of the next instruction to ** be inserted. The parameter "x" must have been obtained from ** a prior call to sqlite3VdbeMakeLabel().
62036. Check that the page exists
62037. Forward references to the various page getters
62038. ** Indicate that subsequent calls to sqlite3Fts5IndexWrite() pertain to ** document iDocid.
62039. Number of entries in aPhrase[] array
62040. Flags that can be added to a token code when it is not ** being stored in a u8:
62041. Justification of ALWAYS(); The index must be on the list of ** indices.
62042. This block serves to assert() that the cursor really does point ** to the last entry in the b-tree.
62043. Pointer to a prior allocation
62044. Base register where results are written
62045. Handle pointing to row containing value
62046. Grow the hash table if required
62047. Table associated with change
62048. ** Maximum error message length (in chars) for WinRT.
62049. zNum is empty or contains non-numeric text or is longer ** than 19 digits (thus guaranteeing that it is too large)
62050. Value to return if (rhs < lhs)
62051. Opcode: Divide P1 P2 P3 * * * Synopsis: r[P3]=r[P2]/r[P1] ** ** Divide the value in register P1 by the value in register P2 ** and store the result in register P3 (P3=P2/P1). If the value in ** register P1 is zero, then the result is NULL. If either input is ** NULL, the result is NULL.
62052. 326
62053. ** Obtain permyriadage (permyriadage is to 10000 as percentage is to 100) ** progress indications for the two stages of an RBU update. This API may ** be useful for driving GUI progress indicators and similar. ** ** An RBU update is divided into two stages: ** ** Stage 1, in which changes are accumulated in an oal/wal file, and ** ** Stage 2, in which the contents of the wal file are copied into the ** main database. ** ** The update is visible to non-RBU clients during stage 2. During stage 1 ** non-RBU reader clients may see the original database. ** ** If this API is called during stage 2 of the update, output variable ** (*pnOne) is set to 10000 to indicate that stage 1 has finished and (*pnTwo) ** to a value between 0 and 10000 to indicate the permyriadage progress of ** stage 2. A value of 5000 indicates that stage 2 is half finished, ** 9000 indicates that it is 90% finished, and so on. ** ** If this API is called during stage 1 of the update, output variable ** (*pnTwo) is set to 0 to indicate that stage 2 has not yet started. The ** value to which (*pnOne) is set depends on whether or not the RBU ** database contains an "rbu_count" table. The rbu_count table, if it ** exists, must contain the same columns as the following: ** ** CREATE TABLE rbu_count(tbl TEXT PRIMARY KEY, cnt INTEGER) WITHOUT ROWID; ** ** There must be one row in the table for each source (data_xxx) table within ** the RBU database. The 'tbl' column should contain the name of the source ** table. The 'cnt' column should contain the number of rows within the ** source table. ** ** If the rbu_count table is present and populated correctly and this ** API is called during stage 1, the *pnOne output variable is set to the ** permyriadage progress of the same stage. If the rbu_count table does ** not exist, then (*pnOne) is set to -1 during stage 1. If the rbu_count ** table exists but is not correctly populated, the value of the *pnOne ** output variable during stage 1 is undefined.
62054. Maximum number of entries in hash table before ** sub-dividing and re-hashing.
62055. IMPLEMENTATION-OF: R-20790-14025 The sqlite3_threadsafe() function returns ** zero if and only if SQLite was compiled with mutexing code omitted due to ** the SQLITE_THREADSAFE compile-time option being set to 0.
62056. **comment:** If the file was just truncated to a size smaller than the currently ** mapped region, reduce the effective mapping size as well. SQLite will ** use read() and write() to access data beyond this point from now on.
label: code-design
62057. Best index found so far
62058. Number of bytes in zPathname
62059. An ORDER BY or GROUP BY clause to resolve
62060. Page number of current leaf page
62061. xDestroy
62062. Bytes of content in pCell
62063. Opcode: BitNot P1 P2 * * * Synopsis: r[P1]= ~r[P1] ** ** Interpret the content of register P1 as an integer. Store the ** ones-complement of the P1 value into register P2. If P1 holds ** a NULL then store a NULL in P2.
62064. True to unlink wal and wal-index files
62065. The OFFSET expression. NULL if there is none
62066. ** CAPI3REF: Low-level system error code ** ** ^Attempt to return the underlying operating system error code or error ** number that caused the most recent I/O error or failure to open a file. ** The return value is OS-dependent. For example, on unix systems, after ** [sqlite3_open_v2()] returns [SQLITE_CANTOPEN], this interface could be ** called to get back the underlying "errno" that caused the problem, such ** as ENOSPC, EAUTH, EISDIR, and so forth.
62067. Used only by system that substitute their own storage engine
62068. trigger_time ::=
62069. rowid >= ? expression (or NULL)
62070. Number of outstanding xFetch refs
62071. adjust exponent by d, and update sign
62072. 57
62073. If Y==0 and X will fit in a 64-bit int, ** handle the rounding directly, ** otherwise use printf.
62074. ** Deallocate and destroy a parser. Destructors are called for ** all stack elements before shutting the parser down. ** ** If the YYPARSEFREE NEVERNULL macro exists (for example because it ** is defined in a %include section of the input grammar) then it is ** assumed that the input pointer is never NULL.
62075. An open database
62076. Parent table of FK constraint pFKey
62077. Pointer to the end of the file
62078. Generate code to implement the subquery ** ** The subquery is implemented as a co-routine if all of these are true: ** (1) The subquery is guaranteed to be the outer loop (so that it ** does not need to be computed more than once) ** (2) The ALL keyword after SELECT is omitted. (Applications are ** allowed to say "SELECT ALL" instead of just "SELECT" to disable ** the use of co-routines.) ** (3) Co-routines are not disabled using sqlite3_test_control() ** with SQLITE_TESTCTRL_OPTIMIZATIONS. ** ** TODO: Are there other reasons beside (1) to use a co-routine ** implementation?
62079. VDBE cursor number of the pSub result set temp table
62080. ***** Win32 Threads *****
62081. True if range end uses ==, >= or <=
62082. Fill in the zName and zDb fields of the vtab structure.
62083. Index in p->pSrc->a[] of the inner subquery
62084. LHS of LIKE/GLOB operator
62085. ** Construct one or more SQL statements from the format string given ** and then evaluate those statements. The success code is written ** into *pRc. ** ** If *pRc is initially non-zero then this routine is a no-op.
62086. **comment:** ** This function is called by unixOpen() to determine the unix permissions ** to create new files with. If no error occurs, then SQLITE_OK is returned ** and a value suitable for passing as the third argument to open(2) is ** written to *pMode. If an IO error occurs, an SQLite error code is ** returned and the value of *pMode is not modified. ** ** In most cases, this routine sets *pMode to 0, which will become ** an indication to robust_open() to create the file using ** SQLITE_DEFAULT_FILE_PERMISSIONS adjusted by the umask. ** But if the file being opened is a WAL or regular journal file, then ** this function queries the file-system for the permissions on the ** corresponding database file and sets *pMode to this value. Whenever ** possible, WAL and journal files are

created using the same permissions ** as the associated database file. *** If the SQLITE_ENABLE_8_3_NAMES option is enabled, then the ** original filename is unavailable. But 8_3_NAMES is only used for ** FAT filesystems and permissions do not matter there, so just use ** the default permissions.

label: code-design

62087. True if input contains UTF16 with high byte non-zero

62088. SQLITE_AFF_BLOB

62089. 254

62090. 193

62091. Symbol on the left-hand side of the rule

62092. Assuming no errors have occurred, set up a merger structure to ** incrementally read and merge all remaining PMAs.

62093. True if the doclist is desc

62094. 287

62095. 311

62096. ***** End of keywordhash.h *****

62097. Page header size. 0 or 100

62098. Work around a sign-extension bug in the HP compiler for HP/UX

62099. Cursor for table associated with pIdx

62100. Jump to the this point in order to terminate the query.

62101. Cursor returned by simpleOpen

62102. c0..c7

62103. ** If the expression p codes a constant integer that is small enough ** to fit in a 32-bit integer, return 1 and put the value of the integer ** in *pValue. If the expression is not an integer or if it is too big ** to fit in a signed 32-bit integer, return 0 and leave *pValue unchanged.

62104. ** Allocate an appendable output segment on absolute level iAbsLevel+1 ** with idx value iIdx. *** In the %_segdir table, a segment is defined by the values in three ** columns: ** start_block ** leaves_end_block ** end_block *** When an appendable segment is allocated, it is estimated that the ** maximum number of leaf blocks that may be required is the sum of the ** number of leaf blocks consumed by the input segments, plus the number ** of input segments, multiplied by two. This value is stored in stack ** variable nLeafEst. *** A total of 16*nLeafEst blocks are allocated when an appendable segment ** is created ((1 + end_block - start_block)=16*nLeafEst). The contiguous ** array of leaf nodes starts at the first block allocated. The array ** of interior nodes that are parents of the leaf nodes start at block ** (start_block + (1 + end_block - start_block) / 16). And so on. *** In the actual code below, the value "16" is replaced with the ** pre-processor macro FTS_MAX_APPENDABLE_HEIGHT.

62105. if SQLITE_THREADSAFE

62106. Stop the operation but leave all prior changes

62107. Pointer to the rightmost select in sub-query

62108. Read the %_segdir entry for index iIdx absolute level (iAbsLevel+1)

62109. Client data for xCreate/xConnect

62110. Used to iterate through samples

62111. Write new entries to *(ppTC)++

62112. '-'. Minus or SQL-style comment

62113. ** Valid sqlite3_blob* handles point to Incrblob structures.

62114. ICU break-iterator object

62115. Read the first integer value

62116. Iterator for WHERE terms

62117. **comment:** Attempt a direct implementation of the built-in COALESCE() and ** IFNULL() functions. This avoids unnecessary evaluation of ** arguments past the first non-NULL argument.

label: code-design

62118. The output

62119. Possibly new term to test

62120. Total cost of this path

62121. Tokenizer hash table

62122. Final block of segment

62123. 215

62124. P2 to OP_Open** is a register number

62125. attempt to handle extremely small/large numbers better

62126. jump, in1, out3

62127. Write the new old.* record. Consists of the PK columns from the ** original old.* record, and the other values from the original ** new.* record.

62128. efts cluster size writes are atomic

62129. **comment:** ** CAPI3REF: Convenience Routines For Running Queries ** METHOD: sqlite3 *** This is a legacy interface that is preserved for backwards compatibility. ** Use of this interface is not recommended. *** Definition: A result table is memory data structure created by the ** [sqlite3_get_table()] interface. A result table records the ** complete query results from one or more queries. *** The table conceptually has a number of rows and columns. But ** these numbers are not part of the result table itself. These ** numbers are obtained separately. Let N be the number of rows ** and M be the number of columns. *** A result table is an array of pointers to zero-terminated UTF-8 strings. ** There are (N+1)*M elements in the array. The first M pointers point ** to zero-terminated strings that contain the names of the columns. ** The remaining entries all point to query results. NULL values result ** in NULL pointers. All other values are in their UTF-8 zero-terminated ** string representation as returned by [sqlite3_column_text()]. *** A result table might consist of one or more memory allocations. ** It is not safe to pass a result table directly to [sqlite3_free()]. ** A result table should be deallocated using [sqlite3_free_table()]. *** ^As an example of the result table format, suppose a query result ** is as follows: *** <blockquote><pre> ** Name | Age ** ----- ** Alice | 43 ** Bob | 28 ** Cindy | 21 ** </pre></blockquote> *** There are two column (M==2) and three rows (N==3). Thus the ** result table has 8 entries. Suppose the result table is stored ** in an array names azResult. Then azResult holds this content: *** <blockquote><pre> ** azResult[91;0] = "Name"; ** azResult[91;1] = "Age"; ** azResult[91;2] = "Alice"; ** azResult[91;3] = "43"; ** azResult[91;4] = "Bob"; ** azResult[91;5] = "28"; ** azResult[91;6] = "Cindy"; ** azResult[91;7] = "21"; ** </pre></blockquote> *** ^The sqlite3_get_table() function evaluates one or more ** semicolon-separated SQL statements in the zero-terminated UTF-8 ** string of its 2nd parameter and returns a result table to the ** pointer given in its 3rd parameter. *** After the application has finished with the result from sqlite3_get_table(), ** it must pass the result table pointer to sqlite3_free_table() in order to ** release the memory that was malloced. Because of the way the ** [sqlite3_malloc()] happens within sqlite3_get_table(), the calling ** function must not try to call [sqlite3_free()] directly. Only ** [sqlite3_free_table()] is able to release the memory properly and safely. *** The sqlite3_get_table() interface is implemented as a wrapper around ** [sqlite3_exec()]. The sqlite3_get_table() routine does not have access ** to any internal data structures of SQLite. It uses only the public ** interface defined here. As a consequence, errors that occur in the ** wrapper layer outside of the internal [sqlite3_exec()] call are not ** reflected in subsequent calls to [sqlite3_errcode()] or ** [sqlite3_errmsg()].

label: code-design

62130. Convert the OP_CreateTable opcode that would normally create the ** root-page for the table into an OP_CreateIndex opcode. The index ** created will become the PRIMARY KEY index.

62131. Take care here not to generate a TK_VECTOR containing only a ** single value. Since the parser never creates such a vector, some ** of the subroutines do not handle this case.

62132. ** For table-valued-functions, transform the function arguments into ** new WHERE clause terms. *** Each function argument translates into an equality constraint against ** a HIDDEN column in the table.

62133. IMP: R-17817-00630

62134. ** Write code to erase the table with root-page iTable from database iDb. ** Also write code to modify the sqlite_master table and internal schema ** if a root-page of another table is moved by the btree-layer whilst ** erasing iTable (this can happen with an auto-vacuum database).

62135. Object being parsed at the point of error

62136. init_deferred_pred_opt ::= INITIALLY IMMEDIATE

62137. Restriction (3)

62138. Generation counter. Incremented with each change

62139. What version of MSVC is being used. 0 means MSVC is not being used
62140. Original schema cookie in destination
62141. Try to pull the page from the write-ahead log.
62142. If more than one frame was recovered from the log file, report an ** event via sqlite3_log(). This is to help with identifying performance ** problems caused by applications routinely shutting down without ** checkpointing the log file.
62143. **comment:** ** THREAD SAFETY NOTES: ** ** Once it has been created using backup_init(), a single sqlite3_backup ** structure may be accessed via two groups of thread-safe entry points: ** ** * Via the sqlite3_backup_XXX() API function backup_step() and ** backup_finish(). Both these functions obtain the source database ** handle mutex and the mutex associated with the source BtShared ** structure, in that order. ** ** * Via the BackupUpdate() and BackupRestart() functions, which are ** invoked by the pager layer to report various state changes in ** the page cache associated with the source database. The mutex ** associated with the source database BtShared structure will always ** be held when either of these functions are invoked. ** ** The other sqlite3_backup_XXX() API functions, backup_remaining() and ** backup_pagecount() are not thread-safe functions. If they are called ** while some other thread is calling backup_step() or backup_finish(), ** the values returned may be invalid. There is no way for a call to ** BackupUpdate() or BackupRestart() to interfere with backup_remaining() ** or backup_pagecount(). ** ** Depending on the SQLite configuration, the database handles and/or ** the Btree objects may have their own mutexes that require locking. ** Non-shareable Btrees (in-memory databases for example), do not have ** associated mutexes.
label: requirement
62144. Magic number to detect structure corruption.
62145. Mask of ORDER BY terms satisfied so far
62146. Append to this buffer
62147. typetoken ::= typename LP signed COMMA signed RP
62148. ERROR: copy extends past end of input
62149. Jump here to break out of UPDATE loop
62150. Modify the .nOut and maybe .rRun fields
62151. ***** Utility routines for dealing with JsonString objects

62152. ** END OF REGISTRATION API *****
62153. An unconstrained column that might be NULL means that this ** WhereLoop is not well-ordered
62154. Extra table options. Usually 0.
62155. True if records are patchset records
62156. Make sure the locking sequence is correct ** (1) We never move from unlocked to anything higher than shared lock. ** (2) SQLite never explicitly requests a pendig lock. ** (3) A shared lock is always held when a reserve lock is requested.
62157. At one point the following block copied a single frame from the ** wal file to the database file. So that one call to sqlite3rbu_step() ** checkpointed a single frame. ** ** However, if the sector-size is larger than the page-size, and the ** application calls sqlite3rbu_savestate() or close() immediately ** after this step, then rbu_step() again, then a power failure occurs, ** then the database page written here may be damaged. Work around ** this by checkpointing frames until the next page in the aFrame[] ** lies on a different disk sector to the current one.
62158. Operation Savepoint Name
62159. If this frame set completes a transaction, then nTruncate>0. If ** nTruncate==0 then this frame set does not complete the transaction.
62160. ***** Include hwtim.h in the middle of vdbe.c *****
62161. HAVE_LOCALTIME_R || HAVE_LOCALTIME_S
62162. ** If SQLITE_RTREE_INT_ONLY is defined, then this virtual table will ** only deal with integer coordinates. No floating point operations ** will be done.
62163. Space used in zNewRecord[] header
62164. ON DELETE action
62165. Unable to edit this page. Rebuild it from scratch instead.
62166. SQLITE_SCHEMA
62167. ** xEof - Return true if the cursor is at EOF, or false otherwise.
62168. *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the implementation for TRIGGERS
62169. ** CAPI3REF: Run-Time Library Version Numbers ** KEYWORDS: sqlite3_version sqlite3_sourceid ** ** These interfaces provide the same information as the [SQLITE_VERSION], ** [SQLITE_VERSION_NUMBER], and [SQLITE_SOURCE_ID] C preprocessor macros ** but are associated with the library instead of the header file. ^Cautious ** programmers might include assert() statements in their application to ** verify that values returned by these interfaces match the macros in ** the header, and thus ensure that the application is ** compiled with matching library and header files. ** ** <blockquote><pre> ** assert(sqlite3_libversion_number()==SQLITE_VERSION_NUMBER); ** assert(strcmp(sqlite3_sourceid(),SQLITE_SOURCE_ID)==0); ** assert(strcmp(sqlite3_libversion(),SQLITE_VERSION)==0); ** </pre></blockquote>** ** ^The sqlite3_version[] string constant contains the text of [SQLITE_VERSION] ** macro. ^The sqlite3_libversion() function returns a pointer to the ** to the sqlite3_version[] string constant. The sqlite3_libversion() ** function is provided for use in DLLs since DLL users usually do not have ** direct access to string constants within the DLL. ^The ** sqlite3_libversion_number() function returns an integer equal to ** [SQLITE_VERSION_NUMBER]. ^The sqlite3_sourceid() function returns ** a pointer to a string constant whose value is the same as the ** [SQLITE_SOURCE_ID] C preprocessor macro. ** ** See also: [sqlite_version()] and [sqlite_source_id()].
62170. ** This function is called by the parser to process a string token. The ** string may or may not be quoted. In any case it is tokenized and a ** phrase object consisting of all tokens returned.
62171. ** Return the cost of sorting nRow rows, assuming that the keys have ** nOrderby columns and that the first nSorted columns are already in ** order.
62172. ** CAPI3REF: Load An Extension ** METHOD: sqlite3 *** ^This interface loads an SQLite extension library from the named file. ** ** ^The sqlite3_load_extension() interface attempts to load an ** [SQLite extension] library contained in the file zFile. If ** the file cannot be loaded directly, attempts are made to load ** with various operating-system specific extensions added. ** So for example, if "samplelib" cannot be loaded, then names like ** "samplelib.so" or "samplelib.dylib" or "samplelib.dll" might ** be tried also. ** ** ^The entry point is zProc. ** ^zProc may be 0, in which case SQLite will try to come up with an ** entry point name on its own. It first tries "sqlite3_extension_init". ** If that does not work, it constructs a name "sqlite3_X_init" where the ** X is consists of the lower-case equivalent of all ASCII alphabetic ** characters in the filename from the last "/" to the first following ** ". " and omitting any initial "lib.". ^** ^The sqlite3_load_extension() interface returns ** [SQLITE_OK] on success and [SQLITE_ERROR] if something goes wrong. ** ^If an error occurs and pzErrMsg is not 0, then the ** [sqlite3_load_extension()] interface shall attempt to ** fill *pzErrMsg with error message text stored in memory ** obtained from [sqlite3_malloc()]. The calling function ** should free this memory by calling [sqlite3_free()]. ** ** ^Extension loading must be enabled using ** [sqlite3_enable_load_extension()] or ** [sqlite3_db_config](db,[SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION],1,NULL) ** prior to calling this API, ** otherwise an error will be returned. ** ** Security warning: It is recommended that the ** [SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION] method be used to enable only this ** interface. The use of the [sqlite3_enable_load_extension()] interface ** should be avoided. This will keep the SQL function [load_extension()] ** disabled and prevent SQL injections from giving attackers ** access to extension loading capabilities. ** ** See also the [load_extension()] SQL function.
62173. Opcode: ResetCount **** The value of the change counter is copied to the database handle ** change counter (returned by subsequent calls to sqlite3_changes()). ** Then the VMs internal change counter resets to 0. ** This is used by trigger programs.
62174. 1: Open a cursor
62175. If there is already a lock of this type or more restrictive on the ** OsFile, do nothing. Don't use the end_lock: exit path, as ** sqlite3OsEnterMutex() hasn't been called yet.
62176. Entry point. Derived from zFile if 0
62177. sortlist
62178. ** Cast the datatype of the value in pMem according to the affinity ** "aff". Casting is different from applying affinity in that a cast ** is forced. In other words, the value is converted into the desired ** affinity even if that results in loss of data. This routine is ** used (for example) to implement the SQL "cast()" operator.
62179. ** Append the hash of the blob passed via the second and third arguments to ** the hash-key value passed as the first. Return the new hash-key value.
62180. Handle to the database.
62181. **comment:** ** 2013-11-12 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains structure and macro definitions for the query
** planner logic in "where.c". These definitions are broken out into ** a separate source file for easier editing.

label: code-design

62182. ** Format and write a message to the log if logging is enabled.

62183. The input VList. Might be NULL

62184. Free any SQLite statements used while processing the previous object

62185. OUT: Error message

62186. Allowed values for WhereLoopBuider.bldFlags

62187. Backup is not possible if the number of bytes of reserve space differ ** between source and destination. If there is a difference, try to ** fix the destination to agree with the source. If that is not possible, ** then the backup cannot proceed.

62188. ** The RBU handle is currently in RBU_STAGE_OAL state, with a SHARED lock ** on the database file. This proc moves the *-oal file to the *-wal path, ** then reopens the database file (this time in vanilla, non-oal, WAL mode). ** If an error occurs, leave an error code and error message in the rbu ** handle.

62189. The requested region is not mapped into this processes address space. ** Check to see if it has been allocated (i.e. if the wal-index file is ** large enough to contain the requested region).

62190. ***** Begin file rtree.c *****

62191. Cell to insert into rtree

62192. Allocate temporary space used by the merge-sort routine. This block ** of memory will be freed before this function returns.

62193. True if pTask->pUnpacked is pKey2

62194. Not a correlated subquery

62195. Bitmask of tables used by pExpr->pRight

62196. Net change in number of documents

62197. Maximum column in pSrc->colUsed

62198. (2)

62199. ** Construct and install a Module object for a virtual table. When this ** routine is called, it is guaranteed that all appropriate locks are held ** and the module is not already part of the connection.

62200. Expressions that are indexed

62201. The element to be removed from the pH

62202. ** If it is currently memory mapped, unmap file pFd.

62203. The recursive part of a recursive CTE

62204. Callback routine requested an abort

62205. ** Initialize the MergeEngine object passed as the second argument. Once this ** function returns, the first key of merged data may be read from the ** MergeEngine object in the usual fashion. ** ** If argument eMode is INCRINIT_ROOT, then it is assumed that any IncrMerge ** objects attached to the PmaReader objects that the merger reads from have ** already been populated, but that they have not yet populated aFile[0] and ** set the PmaReader objects up to read from it. In this case all that is ** required is to call vdbePmaReaderNext() on each PmaReader to point it at ** its first key. ** ** Otherwise, if eMode is any value other than INCRINIT_ROOT, then use ** vdbePmaReaderIncrMergeInit() to initialize each PmaReader that feeds data ** to pMerger. ** ** SQLITE_OK is returned if successful, or an SQLite error code otherwise.

62206. No need to use cachedCellSize() here. The sizes of all cells that ** are to be freed have already been computing while deciding which ** cells need freeing

62207. Only needed by assert() statements

62208. Content is raw, not JSON encoded

62209. Decode the three function arguments

62210. 170

62211. **comment:** ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the sqlite3_get_table() and sqlite3_free_table() ** interface routines. These are just wrappers around the main ** interface routine of sqlite3_exec(). ** ** These routines are in a separate files so that they will not be linked ** if they are not used.

label: code-design

62212. Buffer to merge doclists in

62213. raisetype ::= FAIL

62214. ** This function is called when a transaction opened by the database ** handle associated with the VM passed as an argument is about to be ** committed. If there are outstanding deferred foreign key constraint ** violations, return SQLITE_ERROR. Otherwise, SQLITE_OK. ** ** If there are outstanding FK violations and this function returns ** SQLITE_ERROR, set the result of the VM to SQLITE_CONSTRAINT_FOREIGNKEY ** and write an error message to it. Then return SQLITE_ERROR.

62215. ** The sqlite3_mutex.id, sqlite3_mutex.nRef, and sqlite3_mutex.owner fields ** are necessary under two conditions: (1) Debug builds and (2) using ** home-grown mutexes. Encapsulate these conditions into a single #define.

62216. ** Return an Expr object that refers to column iCol of table pTab which ** has cursor iCur.

62217. Colset to filter on

62218. ** Sort the linked list of records headed at pTask->pList. Return ** SQLITE_OK if successful, or an SQLite error code (i.e. SQLITE_NOMEM) if ** an error occurs.

62219. Store the token that zCsr points to in tname.

62220. We explicitly don't support UTF-8 delimiters for now.

62221. using_opt ::= USING LP idlist RP

62222. ** If parameter iOp is less than zero, then invoke the destructor for ** all auxiliary data pointers currently cached by the VM passed as ** the first argument. ** ** Or, if iOp is greater than or equal to zero, then the destructor is ** only invoked for those auxiliary data pointers created by the user ** function invoked by the OP_Function opcode at instruction iOp of ** VM pVdbe, and only then if: ** ** * the associated function parameter is the 32nd or later (counting ** from left to right), or ** ** * the corresponding bit in argument mask is clear (where the first ** function parameter corresponds to bit 0 etc.).

62223. ** Set or retrieve the codec for this pager

62224. Forward Declaration

62225. Shared memory segment information

62226. EVIDENCE-OF: R-15465-20813 The maximum and minimum embedded payload ** fractions and the leaf payload fraction values must be 64, 32, and 32. ** ** The original design allowed these amounts to vary, but as of ** version 3.6.0, we require them to be fixed.

62227. When the NATURAL keyword is present, add WHERE clause terms for ** every column that the two tables have in common.

62228. Test to make sure the linear array exactly matches the ** Bitvec object. Start with the assumption that they do ** match (rc==0). Change rc to non-zero if a discrepancy ** is found.

62229. ** A memory allocation (also called a "chunk") consists of two or ** more blocks where each block is 8 bytes. The first 8 bytes are ** a header that is not returned to the user. ** ** A chunk is two or more blocks that is either checked out or ** free. The first block has format u.hdr. u.hdr.size4x is 4 times the ** size of the allocation in blocks if the allocation is free. ** The u.hdr.size4x&1 bit is true if the chunk is checked out and ** false if the chunk is on the freelist. The u.hdr.size4x&2 bit ** is true if the previous chunk is checked out and false if the ** previous chunk is free. The u.hdr.prevSize field is the size of ** the previous chunk in blocks if the previous chunk is on the ** freelist. If the previous chunk is checked out, then ** u.hdr.prevSize can be part of the data for that chunk and should ** not be read or written. ** ** We often identify a chunk by its index in mem3.aPool[]. When ** this is done, the chunk index refers to the second block of ** the chunk. In this way, the first chunk has an index of 1. ** A chunk index of 0 means "no such chunk" and is the equivalent ** of a NULL pointer. ** ** The second block of free chunks is of the form u.list. The ** two fields form a double-linked list of chunks of related sizes. ** Pointers to the head of the list are stored in mem3.aiSmall[] ** for smaller chunks and mem3.aiHash[] for larger chunks. ** ** The second block of a chunk is user data if the chunk is checked ** out. If a chunk is checked out, the user data may extend into ** the u.hdr.prevSize value of the following chunk.

62230. ** This is a copy of the sqlite3GetVarint32() routine from the SQLite core. ** Except, this version does handle the single byte case that the core ** version depends on being handled before its function is called.

62231. ** Change the value of the P4 operand for a specific instruction. ** This routine is useful when a large program is loaded from a ** static array using sqlite3VdbeAddOpList but we want to make a ** few minor changes to the program. ** ** If n>=0 then the P4 operand is dynamic, meaning that a copy of ** the

string is made into memory obtained from sqlite3_malloc(). ** A value of n==0 means copy bytes of zP4 up to and including the ** first null byte. If n>0 then copy n+1 bytes of zP4. *** Other values of n (P4_STATIC, P4_COLLSEQ etc.) indicate that zP4 points ** to a string or structure that is guaranteed to exist for the lifetime of ** the Vdbe. In these cases we can just copy the pointer. *** If addr<0 then change P4 on the most recently inserted instruction.

62232. sqlite3PageMalloc()

62233. Memory location that acts as accumulator

62234. pPage might not be a btree page; it might be an overflow page ** or ptrmap page or a free page. In those cases, the following ** call to btreeInitPage() will likely return SQLITE_CORRUPT. ** But no harm is done by this. And it is very important that ** btreeInitPage() be called on every btree page so we make ** the call for every page that comes in for re-initing.

62235. Size each output buffer in bytes

62236. Number of retries

62237. OUT: Score

62238. Allocate the iterator

62239. ** This function is an optimized version of sqlite3VdbeRecordCompare() ** that (a) the first field of pPKey2 is a string, that (b) the first field ** uses the collation sequence BINARY and (c) that the size-of-header varint ** at the start of (pKey1/nKey1) fits in a single byte.

62240. dbnm ::= DOT nm

62241. s: p0<<21 | p1<<14 | p2<<7 | p3 (masked)

62242. If an error occurred in findInodeInfo(), close the file descriptor ** immediately, before releasing the mutex. findInodeInfo() may fail ** in two scenarios: ** ** (a) A call to fstat() failed. ** (b) A malloc failed. *** Scenario (b) may only occur if the process is holding no other ** file descriptors open on the same file. If there were other file ** descriptors on this file, then no malloc would be required by ** findInodeInfo(). If this is the case, it is quite safe to close ** handle h - as it is guaranteed that no posix locks will be released ** by doing so. *** ** If scenario (a) caused the error then things are not so safe. The ** implicit assumption here is that if fstat() fails, things are in ** such bad shape that dropping a lock or two doesn't matter much.

62243. 860

62244. If all the following are true: ** ** 1) this is a read operation, and ** 2) data is required from the start of this overflow page, and ** 3) there is no open write-transaction, and ** 4) the database is file-backed, and ** 5) the page is not in the WAL file ** 6) at least 4 bytes have already been read into the output buffer ** ** then data can be read directly from the database file into the ** output buffer, bypassing the page-cache altogether. This speeds ** up loading large records that span many overflow pages.

62245. ** Add the file descriptor used by file handle pFile to the corresponding ** pUnused list.

62246. New cursor object

62247. Filename might have query parameters

62248. The minimum PMA size is set to this value multiplied by the database ** page size in bytes.

62249. idlist_opt

62250. **comment:** ** Copy the contents of sample *pNew into the p->a[] array. If necessary, ** remove the least desirable sample from p->a[] to make room.

label: code-design

62251. Db name for state ("stat" or "main")

62252. even though it is the conch

62253. This section of code's only "output" is via assert() statements.

62254. ** Information and control of an open file handle.

62255. Read position list size

62256. Min and max heap requests sizes

62257. To reset column cache

62258. The common case: Advance to the next row

62259. 4 CREATE:

62260. Information that depends on pWLoop->wsFlags

62261. The conflict algorithm. (OE_Abort, OE_Ignore, etc)

62262. Invalid key size: 0x80 0x80 0x01

62263. Search for an existing entry that matching the canonical name. ** If found, increment the reference count and return a pointer to ** the existing file ID.

62264. ** Allocate space for memory structures

62265. **comment:** Optimization note: Adding the "pgno<=1" term before "pgno==0" here ** allows the compiler optimizer to reuse the results of the "pgno>1" ** test in the previous statement, and avoid testing pgno==0 in the ** common case where pgno is large.

label: code-design

62266. Reattach the ORDER BY clause to the query.

62267. collate ::= COLLATE ID|STRING

62268. Left outer join

62269. An in-memory journal file should only ever be appended to. Random ** access writes are not required. The only exception to this is when ** the in-memory journal is being used by a connection using the ** atomic-write optimization. In this case the first 28 bytes of the ** journal file may be written as part of committing the transaction.

62270. ** Call from within the xCreate() or xConnect() methods to provide ** the SQLite core with additional information about the behavior ** of the virtual table being implemented.

62271. 55

62272. Number of children of returned node

62273. If this is a database file (not a journal, master-journal or temp ** file), the bytes in the locking range should never be read or written.

62274. ** Recompute all indices of pTab that use the collating sequence pColl. ** If pColl==0 then recompute all indices of pTab.

62275. True if a locking error has occurred

62276. END => ID

62277. ** Select a currently unused rowid for a new r-tree record.

62278. Add the change to the hash-table

62279. 224

62280. Store the index column value in this register

62281. First stack entry

62282. Combine Step 3 and Step 5 into a single opcode

62283. Name of the file to be locked or unlocked

62284. Rows less than the lower bound

62285. Block number of last allocated block

62286. ** pExpr is an operand of a comparison operator. aff2 is the ** type affinity of the other operand. This routine returns the ** type affinity that should be used for the comparison operator.

62287. Only used in legacy parse mode

62288. PLAN

62289. Expression defining the new record number

62290. Context object passed to callback

62291. Escape ' characters

62292. P4 operand type

62293. the ORDER BY clause

62294. List of expressions to scan

62295. Index of largest aReadMark[] value

62296. ** Print out information about all locking operations. ** ** This routine is used for troubleshooting locks on multithreaded ** platforms. Enable by compiling with the -DSQLITE_LOCK_TRACE ** command-line option on the compiler. This code is normally ** turned off.

62297. ** This function is called for every node of an expression that is a candidate ** for a cursor hint on an index cursor. For TK_COLUMN nodes that reference ** the table CCurHint.iTabCur, verify that the same column can be ** accessed through the index. If it cannot, then set pWalker->eCode to 1.

62298. TUNING: Each index lookup yields 20 rows in the table. This ** is more than the usual guess of 10 rows, since we have no way ** of knowing how selective the index will ultimately be. It would ** not be unreasonable to make this value much larger.

62299. nSelectRow set by a constant LIMIT

62300. **comment:** ** Return the number of bytes required to create a duplicate of the ** expression passed as the first argument. The second argument is a ** mask containing EXPRDUP_XXX flags. ** ** The value returned includes space to create a copy of the Expr struct ** itself and the buffer referred to by Expr.u.zToken, if any. ** ** If the EXPRDUP_REDUCE flag is set, then the return value includes ** space to duplicate all Expr nodes in the tree formed by Expr.pLeft ** and Expr.pRight variables (but not for any structures pointed to or ** descended from the Expr.x.pList or Expr.x.pSelect variables).

label: code-design

62301. Used by sqlite3fts3EvalPhrasePoslist() if this is a descendent of an ** OR condition.

62302. EVIDENCE-OF: R-06626-12911 The SQLITE_CONFIG_HEAP option is only ** available if SQLite is compiled with either SQLITE_ENABLE_MEMSYS3 or ** SQLITE_ENABLE_MEMSYS5 and returns SQLITE_ERROR if invoked otherwise.

62303. Conflict resolution algorithms.

62304. Number of elements in pTabList

62305. ** The following routine suspends the current thread for at least ms ** milliseconds. This is equivalent to the Win32 Sleep() interface.

62306. EVIDENCE-OF: R-01470-60482 The sqlite3_value_type(V) interface returns ** the datatype code for the initial datatype of the sqlite3_value object ** V. The returned value is one of SQLITE_INTEGER, SQLITE_FLOAT, ** SQLITE_TEXT, SQLITE_BLOB, or SQLITE_NULL.

62307. Array of arguments for Fts5Config

62308. ** Append the value passed as the second argument to the buffer passed ** as the first. ** ** This function is a no-op if *pRc is non-zero when it is called. ** Otherwise, if an error occurs, *pRc is set to an SQLite error code ** before returning.

62309. Index of child node in parent

62310. Address at which row is visited

62311. Index of the table column

62312. This is an aggregate query

62313. The parsing context

62314. If there are one or more deferred tokens, load the current row into ** memory and scan it to determine the position list for each deferred ** token. Then, see if this row is really a match, considering deferred ** tokens and NEAR operators (neither of which were taken into account ** earlier, by fts3EvalNextRow()).

62315. ** Change the page size for PCache object. The caller must ensure that there ** are no outstanding page references when this function is called.

62316. If the index or temporary table used by the GROUP BY sort ** will naturally deliver rows in the order required by the ORDER BY ** clause, cancel the ephemeral table open coded earlier. ** ** This is an optimization - the correct answer should result regardless. ** Use the SQLITE_GroupByOrder flag with SQLITE_TESTCTRL_OPTIMIZER to ** disable this optimization for testing purposes.

62317. If the node is not the tree root and now has less than the minimum ** number of cells, remove it from the tree. Otherwise, update the ** cell in the parent node so that it tightly contains the updated ** node.

62318. ***** sqlite3_result ***** The following routines are used by user-defined functions to specify ** the function result. ** ** The setStrOrError() function calls sqlite3VdbeMemSetStr() to store the ** result as a string or blob but if the string or blob is too large, it ** then sets the error code to SQLITE_TOOBIG ** ** The invokeValueDestructor(P,X) routine invokes destructor function X() ** on value P is not going to be used and need to be destroyed.

62319. All optimizations

62320. ** This is called by the parser when it sees a CREATE TRIGGER statement ** up to the point of the BEGIN before the trigger actions. A Trigger ** structure is generated based on the information available and stored ** in pParse->pNewTrigger. After the trigger actions have been parsed, the ** sqlite3FinishTrigger() function is called to complete the trigger ** construction process.

62321. **comment:** There is one entry in the aRegIdx[] array for each index on the table ** being updated. Fill in aRegIdx[] with a register number that will hold ** the key for accessing each index. ** ** FIXME: Be smarter about omitting indexes that use expressions.

label: code-design

62322. X-overwrites-A

62323. This routine is never called during trigger-generation. It is ** only called from the top-level

62324. Construct the WhereLoop objects

62325. ** Set *ppModule to point at the implementation of the ICU tokenizer.

62326. Result code of a subroutine

62327. Value type (SQLITE_NULL, TEXT etc.)

62328. Current number of allocations

62329. To ensure unique results if UNION

62330. The value for the token

62331. Seek the index cursor to the start of the range.

62332. Next FKey with the same in pFrom. Next parent of pFrom

62333. ** The parser calls this routine when it sees the first token ** of an argument to the module name in a CREATE VIRTUAL TABLE statement.

62334. Return true if index X is a PRIMARY KEY index

62335. The random check-hash initializer

62336. Randomly chosen byte used as a shared lock

62337. Access to mutexes used by sqlite3_status()

62338. Nul-terminated string containing "A,B"

62339. 174

62340. ** This function is called during initialization (sqlite3_initialize()) to ** install the default pluggable cache module, assuming the user has not ** already provided an alternative.

62341. ** Advance the first term iterator in the first phrase of pNear. Set output ** variable *pbEof to true if it reaches EOF or if an error occurs. ** ** Return SQLITE_OK if successful, or an SQLite error code if an error ** occurs.

62342. #include <unicode/ucol.h>

62343. PK index if pTab is WITHOUT ROWID

62344. PASSIVE, FULL, RESTART, or TRUNCATE

62345. Scan for a consonant

62346. turn on proxy file locking

62347. First arg to xCompare()

62348. True this expression is at EOF already

62349. Type of error

62350. Number of days since 1st day of year

62351. Number of pages in database file

62352. ** Assign a variable number to an expression that encodes a wildcard ** in the original SQL statement. ** ** Wildcards consisting of a single "?" are assigned the next sequential ** variable number. ** ** Wildcards of the form "?nnn" are assigned the number "nnn". We make ** sure "nnn" is not too big to avoid a denial of service attack when ** the SQL statement comes from an external source. ** ** Wildcards of the form ":aaa", "@aaa", or "\$aaa" are assigned the same number ** as the previous instance of the same wildcard. Or if this is the first ** instance of the wildcard, the next sequential variable number is ** assigned.

62353. When adding an index to the list of indices for a table, make ** sure all indices labeled OE_Replace come after all those labeled ** OE_Ignore. This is necessary for the correct constraint check ** processing (in sqlite3GenerateConstraintChecks()) as part of ** UPDATE and INSERT statements.

62354. Opcode: ResetSorter P1 *** Delete all contents from the ephemeral table or sorter ** that is open on cursor P1. ** ** This opcode only works for cursors used for sorting and ** opened with OP_OpenEphemeral or OP_SorterOpen.

62355. ** Allocate a new page from the database file. ** ** The new page is marked as dirty. (In other words, sqlite3PagerWrite() ** has already been called on the new page.) The new page has also ** been referenced and the calling routine is responsible for calling ** sqlite3PagerUnref() on the new page when it is done. ** ** SQLITE_OK is returned on success. Any other return value indicates ** an error. *ppPage is set to NULL in the event of an error. ** ** If the "nearby" parameter is not 0, then an effort is made to ** locate a page close to the page number "nearby". This can be used in an ** attempt to keep related pages close to each other in the database file, ** which in turn can make database access faster. ** ** If the eMode parameter is BTALLOC_EXACT and the nearby page exists ** anywhere

on the free-list, then it is guaranteed to be returned. If ** eMode is BTALLOC_LT then the page returned will be less than or equal ** to nearby if any such page exists. If eMode is BTALLOC_ANY then there ** are no restrictions on which page is returned.

62356. True if resolving a partial index WHERE

62357. ** CAPI3REF: File Locking Levels *** SQLite uses one of these integer values as the second ** argument to calls it makes to the xLock() and xUnlock() methods ** of an [sqlite3_io_methods] object.

62358. ** Release a reference to an Fts5Structure object returned by an earlier ** call to fts5StructureRead() or fts5StructureDecode().

62359. (318) tridxby ::=

62360. Cannot transition directly from MEMORY to WAL. Use mode OFF ** as an intermediate

62361. synopsis: r[P1]=NULL

62362. Scan to write OP_Explain opcode for

62363. If setting the version fields to 1, do not automatically open the ** WAL connection, even if the version fields are currently set to 2.

62364. Context argument for xBusyHandler

62365. PRIMARY

62366. File descriptor returned by open()

62367. ** Argument pExpr is an (?,...) IN(...) expression. This ** function allocates and returns a nul-terminated string containing ** the affinities to be used for each column of the comparison. ** ** It is the responsibility of the caller to ensure that the returned ** string is eventually freed using sqlite3DbFree().

62368. Copy the filename and any query parameters into the zFile buffer. ** Decode %HH escape codes along the way. ** ** Within this loop, variable eState may be set to 0, 1 or 2, depending ** on the parsing context. As follows: *** 0: Parsing file-name. *** 1: Parsing name section of a name=value query parameter. *** 2: Parsing value section of a name=value query parameter.

62369. ON

62370. zStmt = sqlite3MPrintf("");

62371. This test catches attempts to make either operand of a NEAR ** operator something other than a phrase. For example, either of ** the following: *** (bracketed expression) NEAR phrase ** phrase NEAR (bracketed expression) *** *** Return an error in either case.

62372. Result to set *pbRes to

62373. in_op ::= NOT IN

62374. SQLITE_OMIT_LOOKASIDE

62375. Add the term to the terms index

62376. Generate code to cause the string zStr to be loaded into ** register iDest

62377. If the RHS of this IN(...) operator is a SELECT, and if it matters ** whether or not the SELECT result contains NULL values, check whether ** or not NULL is actually possible (it may not be, for example, due ** to NOT NULL constraints in the schema). If no NULL values are possible, ** set prRhsHasNull to 0 before continuing.

62378. Array of size nPhrase

62379. ***** Begin file fts3_tokenize_vtab.c *****

62380. ** Within the RBU_STAGE_OAL stage, each call to sqlite3rbu_step() performs ** one of the following operations.

62381. WAL

62382. HASH_INT, _POINTER, _STRING, _BINARY

62383. **comment:** ** Get an unused page. *** This works just like btreeGetPage() with the addition: *** * If the page is already in use for some other purpose, immediately ** release it and return an SQLITE_CORRUPT error. *** Make sure the isInit flag is clear
label: documentation

62384. Number of arguments used so far

62385. #include "opcodes.h"

62386. Memory cell to write value into

62387. ** Replace the current "averages" record with the contents of the buffer ** supplied as the second argument.

62388. ** Compute the difference (in milliseconds) between localtime and UTC ** (a.k.a. GMT) for the time value p where p is in UTC. If no error occurs, ** return this value and set *pRc to SQLITE_OK. *** Or, if an error does occur, set *pRc to SQLITE_ERROR. The returned value ** is undefined in this case.

62389. A real table

62390. True if UNIQUE and NOT NULL for all columns

62391. same as TK_AND, synopsis: r[P3]=(r[P1] && r[P2])

62392. *****

62393. Create VDBE to loop through the entries in pSrc that match the WHERE ** clause. For each row found, increment either the deferred or immediate ** foreign key constraint counter.

62394. Setup information needed to write frames into the WAL

62395. sign of exponent

62396. Pointer to nul-term of zCopy

62397. **comment:** Malloc a buffer to read frames into.

label: code-design

62398. ** Delete the entry that the cursor is pointing to. *** * If the BTREE_SAVEPOSITION bit of the flags parameter is zero, then ** the cursor is left pointing at an arbitrary location after the delete. ** But if that bit is set, then the cursor is left in a state such that ** the next call to BtreeNext() or BtreePrev() moves it to the same row ** as it would have been on if the call to BtreeDelete() had been omitted. *** The BTREE_AUXDELETE bit of flags indicates that one of several deletes ** associated with a single table entry and its indexes. Only one of those ** deletes is considered the "primary" delete. The primary delete occurs ** on a cursor that is not a BTREE_FORDELETE cursor. All but one delete ** operation on non-FORDELETE cursors is tagged with the AUXDELETE flag. ** The BTREE_AUXDELETE bit is a hint that is not used by this implementation, ** but which might be used by alternative storage engines.

62399. Primary key array

62400. Bytes of payload total (local+overflow)

62401. selectnowith ::= selectnowith multiselect_op oneselect

62402. IN/OUT: Pointer into buffer

62403. nearby parameter for allocateBtreePage()

62404. EV: R-40812-03570

62405. if we already have a lock, it is exclusive. ** Just adjust level and punt on outta here.

62406. Get the VDBE program ready for execution

62407. Register holding largest rowid for AUTOINCREMENT

62408. Which loop to provide hints for

62409. ** Retrieve the next token from the tokenizer cursor pCursor. This ** method should either return SQLITE_OK and set the values of the ** "OUT" variables identified below, or SQLITE_DONE to indicate that ** the end of the buffer has been reached, or an SQLite error code. *** *ppToken should be set to point at a buffer containing the ** normalized version of the token (i.e. after any case-folding and/or ** stemming has been performed). *pnBytes should be set to the length ** of this buffer in bytes. The input text that generated the token is ** identified by the byte offsets returned in *piStartOffset and ** *piEndOffset. *piStartOffset should be set to the index of the first ** byte of the token in the input buffer. *piEndOffset should be set ** to the index of the first byte just past the end of the token in ** the input buffer. *** The buffer *ppToken is set to point at is managed by the tokenizer ** implementation. It is only required to be valid until the next call ** to xNext() or xClose().

62410. Block id of child node to descend to

62411. For debugging use only:

62412. mxMmap

62413. Base register for data coming from SELECT

62414. EVIDENCE-OF: R-63257-11740 Calling sqlite3_close() or ** sqlite3_close_v2() with a NULL pointer argument is a harmless no-op.

62415. Size of pKey, or last integer key

62416. Open a transaction on the database file. Regardless of the journal ** mode, this transaction always uses a rollback journal.

62417. The comparison opcode

62418. ** Provide flag hints to the cursor.

62419. Must be a power of 2
62420. ifndef SQLITE OMIT FOREIGN KEY
62421. The following block runs if pExpr is the root of a NEAR query. ** For example, the query: ** ** "w" NEAR "x" NEAR "y" NEAR "z" ** ** which is represented in tree form as: ** ** | ** +-NEAR--- <- root of NEAR query ** | | ** +-NEAR--+ "z" ** | | ** +-NEAR--+ "y" ** | | ** "w" "x" ** ** The right-hand child of a NEAR node is always a phrase. The ** left-hand child may be either a phrase or a NEAR node. There are ** no exceptions to this - it's the way the parser in fts3_expr.c works.
62422. **comment:** One of the SQL_XXX constants above
label: code-design
62423. ** Memory barrier.
62424. ** Both *pMem1 and *pMem2 contain string values. Compare the two values ** using the collation sequence pColl. As usual, return a negative , zero ** or positive value if *pMem1 is less than, equal to or greater than ** *pMem2, respectively. Similar in spirit to "rc = (*pMem1) - (*pMem2);".
62425. Copy of input using utf-16 encoding
62426. Table to be indexed
62427. Configure the OP_TableLock instruction
62428. Pointer to MatchInfo structure
62429. EVIDENCE-OF: R-58208-19414 The first 2 bytes of a freeblock are a ** big-endian integer which is the offset in the b-tree page of the next ** freeblock in the chain, or zero if the freeblock is the last on the ** chain.
62430. pPage is not necessarily writeable since pCell might be auxiliary ** buffer space that is separate from the pPage buffer area
62431. ** This function does the work for both the xDisconnect and xDestroy methods. ** These tables have no persistent representation of their own, so xDisconnect ** and xDestroy are identical operations.
62432. Append the new serialized varint to the end of the list.
62433. Input path for each iteration of loop
62434. ** Return a pointer to the pPager->pBackup variable. The backup module ** in backup.c maintains the content of this variable. This module ** uses it opaquely as an argument to sqlite3BackupRestart() and ** sqlite3BackupUpdate() only.
62435. Set the value of a single config attribute
62436. Make this EXPR node point to the selected column
62437. Previous term
62438. ***** Single-Threaded *****
62439. Error code returned by pagerUnlockDb()
62440. Happens with a UNIQUE index on ROWID
62441. Preserve the user version
62442. ** Invert a changeset object.
62443. **comment:** Locate the table which we want to delete. This table has to be ** put in an SrcList structure because some of the subroutines we ** will be calling are designed to work with multiple tables and expect ** an SrcList* parameter instead of just a Table* parameter.
label: code-design
62444. Reset and close the cache object
62445. Name of %_data table
62446. Virtual table implementations will typically add additional fields
62447. **comment:** Ensure that the size of a VDBE does not grow too large
label: code-design
62448. ** Generate code that will do an analysis of an entire database
62449. ** Guard words
62450. Save the positions of any other cursors open on this table before ** making any modifications.
62451. ** Virtual table cursor structure.
62452. Value to be returned
62453. True if rowid is modified by this update
62454. Use the incremental approach.
62455. Expected pointer map type
62456. If the cell deleted was not located on a leaf page, then the cursor ** is currently pointing to the largest entry in the sub-tree headed ** by the child-page of the cell that was just deleted from an internal ** node. The cell from the leaf node needs to be moved to the internal ** node to replace the deleted cell.
62457. Cursor handle passed through apVal[0]
62458. Static space for the 2 default backends
62459. True if there is a NOT INDEXED clause
62460. Last update statement (or NULL)
62461. Column counter
62462. Approximate number of tokens in snippet
62463. Append header to this buffer
62464. pContext from when function registered
62465. Delete on close
62466. Of the form A==B, both columns
62467. ** CAPI3REF: Closing A Database Connection ** DESTRUCTOR: sqlite3 *** ^The sqlite3_close() and sqlite3_close_v2() routines are destructors ** for the [sqlite3] object. ** ^Calls to sqlite3_close() and sqlite3_close_v2() return [SQLITE_OK] if ** the [sqlite3] object is successfully destroyed and all associated ** resources are deallocated. ** ** ^If the database connection is associated with unfinalized prepared ** statements or unfinished sqlite3_backup objects then sqlite3_close() ** will leave the database connection open and return [SQLITE_BUSY]. ** ^If sqlite3_close_v2() is called with unfinalized prepared statements ** and/or unfinished sqlite3_backups, then the database connection becomes ** an unusable "zombie" which will automatically be deallocated when the ** last prepared statement is finalized or the last sqlite3_backup is ** finished. The sqlite3_close_v2() interface is intended for use with ** host languages that are garbage collected, and where the order in which ** destructors are called is arbitrary. ** ** Applications should [sqlite3_finalize | finalize] all [prepared statements], ** [sqlite3_blob_close | close] all [BLOB handles], and ** [sqlite3_backup_finish | finish] all [sqlite3_backup] objects associated ** with the [sqlite3] object prior to attempting to close the object. ^If ** sqlite3_close_v2() is called on a [database connection] that still has ** outstanding [prepared statements], [BLOB handles], and/or ** [sqlite3_backup] objects then it returns [SQLITE_OK] and the deallocation ** of resources is deferred until all [prepared statements], [BLOB handles], ** and [sqlite3_backup] objects are also destroyed. ** ** ^If an [sqlite3] object is destroyed while a transaction is open, ** the transaction is automatically rolled back. ** ** The C parameter to [sqlite3_close(C)] and [sqlite3_close_v2(C)] ** must be either a NULL ** pointer or an [sqlite3] object pointer obtained ** from [sqlite3_open0], [sqlite3_open160], or ** [sqlite3_open_v20], and not previously closed. ** ^Calling sqlite3_close() or sqlite3_close_v2() with a NULL pointer ** argument is a harmless no-op.
62468. The main database structure
62469. 790
62470. ** This function is called when a row is inserted into or deleted from the ** child table of foreign key constraint pFKey. If an SQL UPDATE is executed ** on the child table of pFKey, this function is invoked twice for each row ** affected - once to "delete" the old row, and then again to "insert" the ** new row. ** ** Each time it is called, this function generates VDBE code to locate the ** row in the parent table that corresponds to the row being inserted into ** or deleted from the child table. If the parent row can be found, no ** special action is taken. Otherwise, if the parent row can *not* be ** found in the parent table: ** ** Operation | FK type | Action taken ** ----- ** INSERT immediate Increment the "immediate constraint counter". ** ** DELETE immediate Decrement the "immediate constraint counter". ** ** INSERT deferred Increment the "deferred constraint counter". ** ** DELETE deferred Decrement the "deferred constraint counter". ** ** These operations are identified in the comment at the top of this file ** (fkey.c) as "I.1" and "D.1".
62471. This column was already computed by the previous index
62472. Value of 'path' column
62473. The SQL table being indexed
62474. ***** Begin file os.h *****
62475. Filesystem type name

62476. The FTS table to delete from
62477. We will be overwriting WhereLoop p[]. But before we do, first ** go through the rest of the list and delete any other entries besides ** p[] that are also supplanted by pTemplate
62478. int iCid = sqlite3_column_int(pXInfo, 0);
62479. Copy of p->aMem
62480. ** Virtual-table cursor object. *** iSpecial: ** If this is a 'special' query (refer to function fts5SpecialMatch()), ** then this variable contains the result of the query. *** iFirstRowid, iLastRowid: ** These variables are only used for FTS5_PLAN_MATCH cursors. Assuming the ** cursor iterates in ascending order of rowids, iFirstRowid is the lower ** limit of rowids to return, and iLastRowid the upper. In other words, the ** WHERE clause in the user's query might have been: *** <tbl> MATCH <expr> AND rowid BETWEEN \$iFirstRowid AND \$iLastRowid *** If the cursor iterates in descending order of rowid, iFirstRowid ** is the upper limit (i.e. the "first" rowid visited) and iLastRowid ** the lower.
62481. Jump to here if a string or blob larger than SQLITE_MAX_LENGTH ** is encountered.
62482. ** This is the maximum amount of data (in bytes) to store in the ** Fts3Table.pendingTerms hash table. Normally, the hash table is ** populated as documents are inserted/updated/deleted in a transaction ** and used to create a new segment when the transaction is committed. ** However if this limit is reached midway through a transaction, a new ** segment is created and the hash table cleared immediately.
62483. Here code is inserted which will be executed whenever the ** parser accepts
62484. ** Copy the complete content of pBtFrom into pBtTo. A transaction ** must be active for both files. *** The size of file pTo may be reduced by this operation. If anything ** goes wrong, the transaction on pTo is rolled back. If successful, the ** transaction is committed before returning.
62485. Pluggable cache module
62486. ** The sqlite3_wal_hook() callback registered by sqlite3_wal_autocheckpoint(). ** Invoke sqlite3_wal_checkpoint if the number of frames in the log file ** is greater than sqlite3.pWalArg cast to an integer (the value configured by ** wal_autocheckpoint()).
62487. 33==sqlite3LogEst(10)
62488. Cached next search point
62489. The BTree that contains the table
62490. The first fields of the two keys are equal. Compare the trailing ** fields.
62491. **comment:** ** Alternative datatype for the argument to the malloc() routine passed ** into sqlite3ParserAlloc(). The default is size_t.
 label: code-design
62492. ** Clear the current cursor position.
62493. **comment:** ** Initialize the memory allocator. *** This routine is not threadsafe. The caller must be holding a mutex ** to prevent multiple threads from entering at the same time.
 label: code-design
62494. Leaf page number to load dlidx for
62495. Size of serialized value in bytes
62496. ***** Include sqliteicu.h in the middle of main.c *****
62497. Find the column for which info is requested
62498. Space to hold MJ filename from a journal file
62499. #include <pthread.h>
62500. ** Return the rowid for the current row. In this implementation, the ** rowid is the same as the output value.
62501. ** Convert a LogEst into an integer. *** Note that this routine is only used when one or more of various ** non-standard compile-time options is enabled.
62502. **comment:** ** CAPI3REF: Memory Allocation Subsystem *** The SQLite core uses these three routines for all of its own ** internal memory allocation needs. "Core" in the previous sentence ** does not include operating-system specific VFS implementation. The ** Windows VFS uses native malloc() and free() for some operations. *** ^The sqlite3_malloc() routine returns a pointer to a block ** of memory at least N bytes in length, where N is the parameter. ** ^If sqlite3_malloc() is unable to obtain sufficient free ** memory, it returns a NULL pointer. ^If the parameter N to ** sqlite3_malloc() is zero or negative then sqlite3_malloc() returns ** a NULL pointer. *** ^The sqlite3_malloc64(N) routine works just like ** sqlite3_malloc(N) except that N is an unsigned 64-bit integer instead ** of a signed 32-bit integer. *** ^Calling sqlite3_free() with a pointer previously returned ** by sqlite3_malloc() or sqlite3_realloc() releases that memory so ** that it might be reused. ^The sqlite3_free() routine is ** a no-op if it is called with a NULL pointer. Passing a NULL pointer ** to sqlite3_free() is harmless. After being freed, memory ** should neither be read nor written. Even reading previously freed ** memory might result in a segmentation fault or other severe error. ** Memory corruption, a segmentation fault, or other severe error ** might result if sqlite3_free() is called with a non-NULL pointer that ** was not obtained from sqlite3_malloc() or sqlite3_realloc(). *** ^The sqlite3_realloc(X,N) interface attempts to resize a ** prior memory allocation X to be at least N bytes. ** ^If the X parameter to sqlite3_realloc(X,N) ** is a NULL pointer then its behavior is identical to calling ** sqlite3_malloc(N). ** ^If the N parameter to sqlite3_realloc(X,N) is zero or ** negative then the behavior is exactly the same as calling ** sqlite3_free(X). ** ^sqlite3_realloc(X,N) returns a pointer to a memory allocation ** of at least N bytes in size or NULL if insufficient memory is available. ** ^If M is the size of the prior allocation, then min(N,M) bytes ** of the prior allocation are copied into the beginning of buffer returned ** by sqlite3_realloc(X,N) and the prior allocation is freed. ** ^If sqlite3_realloc(X,N) returns NULL and N is positive, then the ** prior allocation is not freed. *** ^The sqlite3_realloc64(X,N) interfaces works the same as ** sqlite3_realloc(X,N) except that N is a 64-bit unsigned integer instead ** of a 32-bit signed integer. *** ^If X is a memory allocation previously obtained from sqlite3_malloc(), ** sqlite3_malloc64(), sqlite3_realloc(), or sqlite3_realloc64(), then ** sqlite3_msize(X) returns the size of that memory allocation in bytes. ** ^The value returned by sqlite3_msize(X) might be larger than the number ** of bytes requested when X was allocated. ^If X is a NULL pointer then ** sqlite3_msize(X) returns zero. If X points to something that is not ** the beginning of memory allocation, or if it points to a formerly ** valid memory allocation that has now been freed, then the behavior ** of sqlite3_msize(X) is undefined and possibly harmful. *** ^The memory returned by sqlite3_malloc(), sqlite3_realloc(), ** sqlite3_malloc64(), and sqlite3_realloc64() ** is always aligned to at least an 8 byte boundary, or to a ** 4 byte boundary if the [SQLITE_4_BYTE_ALIGNED_MALLOC] compile-time ** option is used. *** In SQLite version 3.5.0 and 3.5.1, it was possible to define ** the SQLITE OMIT_MEMORY_ALLOCATION which would cause the built-in ** implementation of these routines to be omitted. That capability ** is no longer provided. Only built-in memory allocators can be used. *** Prior to SQLite version 3.7.10, the Windows OS interface layer called ** the system malloc() and free() directly when converting ** filenames between the UTF-8 encoding used by SQLite ** and whatever filename encoding is used by the particular Windows ** installation. Memory allocation errors were detected, but ** they were reported back as [SQLITE_CANTOPEN] or ** [SQLITE_IOERR] rather than [SQLITE_NOMEM]. *** The pointer arguments to [sqlite3_free()] and [sqlite3_realloc()] ** must be either NULL or else pointers obtained from a prior ** invocation of [sqlite3_malloc()] or [sqlite3_realloc()] that have ** not yet been released. *** The application must not read or write any part of ** a block of memory after it has been released using ** [sqlite3_free()] or [sqlite3_realloc()].
 label: code-design
62503. Ax
62504. Context for fts3ExprTermOffsetInit()
62505. The cursor number of the table
62506. ** Flush the contents of pendingTerms to level 0 segments.
62507. ** Return the filename of the database associated with a database ** connection.
62508. If there is enough space between gap and top for one more cell pointer ** array entry offset, and if the freelist is not empty, then search the ** freelist looking for a free slot big enough to satisfy the request.
62509. Append the assignments
62510. ** These routines are Walker callbacks used to check expressions to ** see if they are "constant" for some definition of constant. The ** Walker.eCode value determines the type of "constant" we are looking ** for. *** These callback routines are used to implement the following: *** sqlite3ExprIsConstant() pWalker->eCode==1 ** sqlite3ExprIsConstantNotJoin() pWalker->eCode==2 ** sqlite3ExprIsTableConstant() pWalker->eCode==3 ** sqlite3ExprIsConstantOrFunction() pWalker->eCode==4 or 5 *** In all cases, the callbacks set Walker.eCode=0 and abort if the expression ** is found to not be a constant. *** The sqlite3ExprIsConstantOrFunction() is used for evaluating expressions ** in a CREATE TABLE statement. The Walker.eCode value is 5 when parsing ** an existing schema and 4 when processing a new statement. A bound ** parameter raises an error for new statements, but is silently converted ** to NULL for existing schemas. This allows sqlite_master tables that ** contain a bound parameter because they were generated by older versions ** of SQLite to be parsed by newer versions of SQLite without raising a ** malformed schema error.
62511. #define sqliteHashCount(H) ((H)->count) // NOT USED
62512. Set variable i to the maximum number of bytes of input to tokenize.
62513. Data for tables. NULL for indexes

62514. ** Check to see if the FROM clause term pFrom has table-valued function ** arguments. If it does, leave an error message in pParse and return ** non-zero, since pFrom is not allowed to be a table-valued function.

62515. 164

62516. Out-of-Order hidden columns

62517. whereexpr.c:

62518. due to (6)

62519. Host database connection

62520. Cursor number of LHS

62521. Decode any more doclist data that appears on the page before the ** first term.

62522. Sorter that owns this sub-task

62523. NO-OP

62524. Opcode: SorterInsert P1 P2 * * * ** Synopsis: key=r[P2] ** ** Register P2 holds an SQL index key made using the ** MakeRecord instructions. This opcode writes that key ** into the sorter P1. Data for the entry is nil.

62525. List of triggers to (potentially) fire

62526. xFindFunction

62527. **comment:** The public SQLite interface. The _FILE_OFFSET_BITS macro must appear ** first in QNX. Also, the _USE_32BIT_TIME_T macro must appear first for ** MinGW.

label: code-design

62528. If any of the expressions is an IPK column on table iBase, then return ** true. Note: The (p->iTable==iBase) part of this test may be false if the ** current SELECT is a correlated sub-query.

62529. One register allocated to each index

62530. **comment:** ** CAPI3REF: Function Auxiliary Data ** METHOD: sqlite3_context ** ** These functions may be used by (non-aggregate) SQL functions to ** associate metadata with argument values. If the same value is passed to ** multiple invocations of the same SQL function during query execution, under ** some circumstances the associated metadata may be preserved. An example ** of where this might be useful is in a regular-expression matching ** function. The compiled version of the regular expression can be stored as ** metadata associated with the pattern string. ** Then as long as the pattern string remains the same, ** the compiled regular expression can be reused on multiple ** invocations of the same function. ** ** ^The sqlite3_get_auxdata(C,N) interface returns a pointer to the metadata ** associated by the sqlite3_set_auxdata(C,N,P,X) function with the Nth argument ** value to the application-defined function. ^N is zero for the left-most ** function argument. ^If there is no metadata ** associated with the function argument, the sqlite3_get_auxdata(C,N) interface ** returns a NULL pointer. ** ** ^The sqlite3_set_auxdata(C,N,P,X) interface saves P as metadata for the N-th ** argument of the application-defined function. ^Subsequent ** calls to sqlite3_get_auxdata(C,N) return P from the most recent ** sqlite3_set_auxdata(C,N,P,X) call if the metadata is still valid or ** NULL if the metadata has been discarded. ** ^After each call to sqlite3_set_auxdata(C,N,P,X) where X is not NULL, ** SQLite will invoke the destructor function X with parameter P exactly ** once, when the metadata is discarded. ** SQLite is free to discard the metadata at any time, including: ** ^when the corresponding function parameter changes, or ** ^when [sqlite3_reset()] or [sqlite3_finalize()] is called for the ** SQL statement, or ** ^when the sqlite3_set_auxdata() is invoked again on the same ** parameter, or ** ^during the original sqlite3_set_auxdata() call when a memory ** allocation error occurs. ** ** Note the last bullet in particular. The destructor X in ** sqlite3_set_auxdata(C,N,P,X) might be called immediately, before the ** sqlite3_set_auxdata() interface even returns. Hence sqlite3_set_auxdata() ** should be called near the end of the function implementation and the ** function implementation should not make any use of P after ** sqlite3_set_auxdata() has been called. ** ** ^In practice, metadata is preserved between function calls for ** function parameters that are compile-time constants, including literal ** values and [parameters] and expressions composed from the same.)^ ** ** The value of the N parameter to these interfaces should be non-negative. ** Future enhancements may make use of negative N values to define new ** kinds of function caching behavior. ** ** These routines must be called from the same thread in which ** the SQL function is running.

label: code-design

62531. Reader to advance to next docid

62532. Name of the collating sequence, UTF-8 encoded

62533. Temp var to store a page number in

62534. ** Create a new collating function for database "db". The name is zName ** and the encoding is enc.

62535. The open database file

62536. ** The MEM structure is already a MEM_Real. Try to also make it a ** MEM_Int if we can.

62537. Advance each segment iterator until it points to the term zTerm/nTerm.

62538. Change record

62539. OUT: number of columns

62540. ** Return true if the iterator passed as the second argument currently ** points to a delete marker. A delete marker is an entry with a 0 byte ** position-list.

62541. For use by built-in VFS

62542. If the cursor is not currently open or is open on a different ** index, then fall through into OP_OpenRead to force a reopen

62543. true if shared-cache mode enabled

62544. Subquery id 2

62545. NOTNULL

62546. Step 3.

62547. maybe a number

62548. ***** Begin file fault.c *****

62549. ** Read the wal-index header from the wal-index and into pWal->hdr. ** If the wal-header appears to be corrupt, try to reconstruct the ** wal-index from the WAL before returning. ** ** Set *pChanged to 1 if the wal-index header value in pWal->hdr is ** changed by this operation. If pWal->hdr is unchanged, set *pChanged to 0. ** ** If the wal-index header is successfully read, return SQLITE_OK. ** Otherwise an SQLite error code.

62550. ** Initialize the memory allocation subsystem.

62551. Index of next ? host parameter

62552. Statement holding cursor open

62553. ** Return the margin length of cell p. The margin length is the sum ** of the objects size in each dimension.

62554. !defined(SQLITE_OMIT_ANALYZE)

62555. OUT: New merge-engine

62556. ** Parameter zName points to a nul-terminated buffer containing the name ** of a database ("main", "temp" or the name of an attached db). This ** function returns the index of the named database in db->aDb[], or ** -1 if the named db cannot be found.

62557. YYERRORSYMBOL is not defined

62558. Cursor number for the table corresponding to the index

62559. Bytes of title; includes '\0'

62560. Check the table schema is still Ok.

62561. Iterate through the b-tree hierarchy.

62562. ** CAPI3REF: Create A New Session Object ** ** Create a new session object attached to database handle db. If successful, ** a pointer to the new object is written to *ppSession and SQLITE_OK is ** returned. If an error occurs, *ppSession is set to NULL and an SQLite ** error code (e.g. SQLITE_NOMEM) is returned. ** ** It is possible to create multiple session objects attached to a single ** database handle. ** ** Session objects created using this function should be deleted using the ** [sqlite3session_delete()] function before the database handle that they ** are attached to is itself closed. If the database handle is closed before ** the session object is deleted, then the results of calling any session ** module function, including [sqlite3session_delete()] on the session object ** are undefined. ** ** Because the session module uses the [sqlite3_preupdate_hook()] API, it ** is not possible for an application to register a pre-update hook on a ** database handle that has one or more session objects attached. Nor is ** it possible to create a session object attached to a database handle for ** which a pre-update hook is already defined. The results of attempting ** either of these things are undefined. ** ** The session object will be used to create changesets for tables in ** database zDb, where zDb is either "main", or "temp", or the name of an ** attached database. It is not an error if database zDb is not attached ** to the database when the session object is created.

62563. Current number of distinct checkouts

62564. True for temp files (incl. in-memory files)

62565. Set of pages in this savepoint

62566. New journal mode
62567. SegmentWriter to flush to the db
62568. Allocate space for the output. Both the input and output doclists ** are delta encoded. If they are in ascending order (bDescDoclist==0), ** then the first docid in each list is simply encoded as a varint. For ** each subsequent docid, the varint stored is the difference between the ** current and previous docid (a positive number - since the list is in ** ascending order). *** The first docid written to the output is therefore encoded using the ** same number of bytes as it is in whichever of the input lists it is ** read from. And each subsequent docid read from the same input list ** consumes either the same or less bytes as it did in the input (since ** the difference between it and the previous value in the output must ** be a positive value less than or equal to the delta value read from ** the input list). The same argument applies to all but the first docid ** read from the 'other' list. And to the contents of all position lists ** that will be copied and merged from the input to the output. *** However, if the first docid copied to the output is a negative number, ** then the encoding of the first docid from the 'other' input list may ** be larger in the output than it was in the input (since the delta value ** may be a larger positive integer than the actual docid). *** The space required to store the output is therefore the sum of the ** sizes of the two inputs, plus enough space for exactly one of the input ** docids to grow. *** A symmetric argument may be made if the doclists are in descending ** order.
62569. ** If we get here it means the result set contains one or more "/*" ** operators that need to be expanded. Loop through each expression ** in the result set and expand them one by one.
62570. ** Code an OP_Affinity opcode to apply the column affinity string zAff ** to the n registers starting at base. *** As an optimization, SQLITE_AFF_BLOB entries (which are no-ops) at the ** beginning and end of zAff are ignored. If all entries in zAff are ** SQLITE_AFF_BLOB, then no code gets generated. *** This routine makes its own copy of zAff so that the caller is free ** to modify zAff after this routine returns.
62571. String to return via *pzImposterPK
62572. 127
62573. C89 specifies that the constant "dummy" will be initialized to all ** zeros, which is correct. MSVC generates a warning, nevertheless.
62574. Store result in an queue
62575. ** A global array of all winShmNode objects. *** The winShmMutexHeld() must be true while reading or writing this list.
62576. **comment:** ** Sub-iterator iChanged of iterator pIter has just been advanced. It still ** points to the same term though - just a different rowid. This function ** attempts to update the contents of the pIter->aFirst[] accordingly. ** If it does so successfully, 0 is returned. Otherwise 1. *** If non-zero is returned, the caller should call fts5MultiIterAdvanced() ** on the iterator instead. That function does the same as this one, except ** that it deals with more complicated cases as well.
label: code-design
62577. SQLITE_HAS_CODEC
62578. The CREATE token that begins this statement
62579. Result of previous sqlite3BtreeMoveto() or 0 ** if there have been no prior seeks on the cursor.
62580. <tbl> MATCH ? expression (or NULL)
62581. Left-most SELECT statement
62582. Checksum over all prior fields
62583. simulate multiple hosts by creating unique hostid file paths
62584. ** This routine is called to process a compound query form from ** two or more separate queries using UNION, UNION ALL, EXCEPT, or ** INTERSECT ** ** "p" points to the right-most of the two queries. the query on the ** left is p->pPrior. The left query could also be a compound query ** in which case this routine will be called recursively. *** The results of the total query are to be written into a destination ** of type eDest with parameter iParm. *** Example 1: Consider a three-way compound SQL statement. *** SELECT a FROM t1 UNION SELECT b FROM t2 UNION SELECT c FROM t3 *** This statement is parsed up as follows: *** SELECT c FROM t3 ** | ** `----> SELECT b FROM t2 ** | ** `----> SELECT a FROM t1 *** The arrows in the diagram above represent the Select.pPrior pointer. ** So if this routine is called with p equal to the t3 query, then ** pPrior will be the t2 query. p->op will be TK_UNION in this case. *** Notice that because of the way SQLite parses compound SELECTs, the ** individual selects always group from left to right.
62585. Contents of master journal file
62586. same as TK_OR, in1, in2, out3
62587. **comment:** ** Gather statistics on the sizes of memory allocations. ** nAlloc[i] is the number of allocation attempts of i*8 ** bytes. i==NCSIZE is the number of allocation attempts for ** sizes more than NCSIZE*8 bytes.
label: code-design
62588. For a rowid table, initialize the RowSet to an empty set
62589. Binary representation of iNew
62590. Current size of database file
62591. Text encoding
62592. **comment:** Langid of recently inserted document
label: documentation
62593. First byte to be locked
62594. Value for "from" column of output
62595. ** The SELECT statement iterating through the keys for the current object ** (p->objiter.pSelect) currently points to a valid row. However, there ** is something wrong with the rru_control value in the rru_control value ** stored in the (p->nCol+1)'th column. Set the error code and error message ** of the RBU handle to something reflecting this.
62596. ** If SQLITE_ENABLE_8_3_NAMES is set at compile-time and if the database ** filename in zBaseFilename is a URI with the "8_3_names=1" parameter and ** if filename in z[] has a suffix (a.k.a. "extension") that is longer than ** three characters, then shorten the suffix on z[] to be the last three ** characters of the original suffix. *** If SQLITE_ENABLE_8_3_NAMES is set to 2 at compile-time, then always ** do the suffix shortening regardless of URI parameter. *** Examples: *** test.db-journal => test.nal ** test.db-wal => test.wal ** test.db-shm => test.shm ** test.db-mj7f3319fa => test.9fa
62597. A small processing code
62598. ** This function gathers 'y' or 'b' data for a single phrase.
62599. Return from sqlite3WhereBegin()
62600. Neither side of the comparison is a column. Compare the ** results directly.
62601. Largest allocation (exclusive of internal frag)
62602. A column number within table iCur
62603. True if auto-vacuum is enabled
62604. WHERE clause information
62605. The string collected so far
62606. ** Write an error message into *pzErr
62607. Opcode: DropIndex P1 * * P4 * * * Remove the internal (in-memory) data structures that describe ** the index named P4 in database P1. This is called after an index ** is dropped from disk (using the Destroy opcode) ** in order to keep the internal representation of the ** schema consistent with what is on disk.
62608. Index of first cell to add
62609. Append the data to the string buffer.
62610. Text of the SQL statement that generated this
62611. ** Apply a delta. *** The output buffer should be big enough to hold the whole output ** file and a NUL terminator at the end. The delta_output_size() ** routine will determine this size for you. *** The delta string should be null-terminated. But the delta string ** may contain embedded NUL characters (if the input and output are ** binary files) so we also have to pass in the length of the delta in ** the lenDelta parameter. *** This function returns the size of the output file in bytes (excluding ** the final NUL terminator character). Except, if the delta string is ** malformed or intended for use with a source file other than zSrc, ** then this routine returns -1. *** Refer to the delta_create() documentation above for a description ** of the delta file format.
62612. Number of columns in the PRIMARY KEY
62613. struct Trigger
62614. Only one of WHERE_OR_SUBCLAUSE or WHERE_USE_LIMIT
62615. Temporarily holds the number of cursors assigned
62616. zero or more BTCL_* flags defined below
62617. Page 1 is always available in cache, because ** pCache->nRefSum>0
62618. ** Maximum size (in Mem3Blocks) of a "small" chunk.

62619. ** The "file format" number is an integer that is incremented whenever ** the VDBE-level file format changes. The following macros define the ** the default file format for new databases and the maximum file format ** that the library can read.

62620. Text inserted into table-name column

62621. Default locking-mode for attached dbs

62622. OS Interface

62623. The right-hand side of the AS subexpression

62624. Buffer containing segment interior node

62625. (285) columnlist ::= columnname carglist

62626. Unsorted entry count

62627. IMP: R-51463-25634

62628. Begin scanning through the ephemeral table.

62629. Create the new column data

62630. **comment:** Opcode: Goto * P2 *** *** An unconditional jump to address P2. ** The next instruction executed will be ** the one at index P2 from the beginning of ** the program. *** ** The P1 parameter is not actually used by this opcode. However, it ** is sometimes set to 1 instead of 0 as a hint to the command-line shell ** that this Goto is the bottom of a loop and that the lines from P2 down ** to the current line should be indented for EXPLAIN output.

label: code-design

62631. SQLITE_CORE || SQLITE_ENABLE_FTS3

62632. Update the Fts5Structure. It is written back to the database by the ** fts5StructureRelease() call below.

62633. No usable constraint

62634. Is a label of an object

62635. ** Free a hash table object.

62636. Current block id

62637. If input table has column "rbu_rowid"

62638. A-overwrites-X

62639. **comment:** ** This function is used after merging multiple segments into a single large ** segment to delete the old, now redundant, segment b-trees. Specifically, ** it: *** *** 1) Deletes all %_segments entries for the segments associated with ** each of the SegReader objects in the array passed as the third ** argument, and ** *** 2) deletes all %_segdir entries with level iLevel, or all %_segdir ** entries regardless of level if (iLevel<0). *** ** SQLITE_OK is returned if successful, otherwise an SQLite error code.

label: code-design

62640. ** Somewhere on pPage is a pointer to page iFrom. Modify this pointer so ** that it points to iTo. Parameter eType describes the type of pointer to ** be modified, as follows: *** PTRMAP_BTREE: pPage is a btree-page. The pointer points at a child ** page of pPage. *** PTRMAP_OVERFLOW1: pPage is a btree-page. The pointer points at an overflow ** page pointed to by one of the cells on pPage. *** PTRMAP_OVERFLOW2: pPage is an overflow-page. The pointer points at the next ** overflow page in the list.

62641. ** Compare the entries pointed to by two Fts3SegReader structures. ** Comparison is as follows: *** ** 1) EOF is greater than not EOF. *** ** 2) The current terms (if any) are compared using memcmp(). If one ** term is a prefix of another, the longer term is considered the ** larger. *** ** 3) By segment age. An older segment is considered larger.

62642. **comment:** ** CAPI3REF: Result Values From A Query ** KEYWORDS: {column access functions} ** METHOD: sqlite3_stmt *** *** Summary: ** <blockquote><table border=0 cellpadding=0 cellspacing=0> ** <tr><td>sqlite3_column_blob</td>→<td>BLOB result ** <tr><td> sqlite3_column_double</td>→<td>REAL result ** <tr><td>sqlite3_column_int</td>→<td>32-bit INTEGER result ** <tr><td> sqlite3_column_int64</td>→<td>64-bit INTEGER result ** <tr><td>sqlite3_column_text</td>→<td>UTF-8 TEXT result ** <tr><td> sqlite3_column_text16</td>→<td>UTF-16 TEXT result ** <tr><td>sqlite3_column_value</td>→<td>The result as an ** [sqlite3_value|unprotected sqlite3_value] object. ** <tr><td> </td> <td> </td>→<td>Size of a BLOB ** or a UTF-8 TEXT result in bytes ** <tr><td>sqlite3_column_bytes16 </td> </td>→<td>Size of UTF-16 ** TEXT in bytes ** <tr><td>sqlite3_column_type</td>→<td>Default ** datatype of the result ** </table></blockquote> *** *** Details: *** ^These routines return information about a single column of the current ** result row of a query. ^In every case the first argument is a pointer ** to the [prepared statement] that is being evaluated (the [sqlite3_stmt*] ** that was returned from [sqlite3_prepare_v2()] or one of its variants) ** and the second argument is the index of the column for which information ** should be returned. ^The leftmost column of the result set has the index 0. ** ^The number of columns in the result can be determined using ** [sqlite3_column_count()]. *** ** If the SQL statement does not currently point to a valid row, or if the ** column index is out of range, the result is undefined. ** These routines may only be called when the most recent call to ** [sqlite3_step()] has returned [SQLITE_ROW] and neither ** [sqlite3_reset()] nor [sqlite3_finalize()] have been called subsequently. ** If any of these routines are called after [sqlite3_reset()] or ** [sqlite3_finalize()] or after [sqlite3_step()] has returned ** something other than [SQLITE_ROW], the results are undefined. ** If [sqlite3_step()] or [sqlite3_reset()] or [sqlite3_finalize()] ** are called from a different thread while any of these routines ** are pending, then the results are undefined. *** ** The first six interfaces (_blob, _double, _int, _int64, _text, and _text16) ** each return the value of a result column in a specific data format. If ** the result column is not initially in the requested format (for example, ** if the query returns an integer but the sqlite3_column_text() interface ** is used to extract the value) then an automatic type conversion is performed. *** ** ^The sqlite3_column_type() routine returns the ** [SQLITE_INTEGER | datatype code] for the initial data type ** of the result column. ^The returned value is one of [SQLITE_INTEGER], ** [SQLITE_FLOAT], [SQLITE_TEXT], [SQLITE_BLOB], or [SQLITE_NULL]. ** The return value of sqlite3_column_type() can be used to decide which ** of the first six interface should be used to extract the column value. ** The value returned by sqlite3_column_type() is only meaningful if no ** automatic type conversions have occurred for the value in question. ** After a type conversion, the result of calling sqlite3_column_type() ** is undefined, though harmless. Future ** versions of SQLite may change the behavior of sqlite3_column_type() ** following a type conversion. *** ** If the result is a BLOB or a TEXT string, then the sqlite3_column_bytes() ** or sqlite3_column_bytes16() interfaces can be used to determine the size ** of that BLOB or string. *** ** ^If the result is a BLOB or UTF-8 string then the sqlite3_column_bytes() ** routine returns the number of bytes in that BLOB or string. ** ^If the result is a UTF-16 string, then sqlite3_column_bytes() converts ** the string to UTF-8 and then returns the number of bytes. ** ^If the result is a numeric value then sqlite3_column_bytes() uses ** [sqlite3_snprintf()] to convert that value to a UTF-8 string and returns ** the number of bytes in that string. ** ^If the result is NULL, then sqlite3_column_bytes() returns zero. ** ** ^If the result is a BLOB or UTF-16 string then the sqlite3_column_bytes16() ** routine returns the number of bytes in that BLOB or string. ** ^If the result is a UTF-8 string, then sqlite3_column_bytes16() converts ** the string to UTF-16 and then returns the number of bytes. ** ^If the result is a numeric value then sqlite3_column_bytes16() uses ** [sqlite3_snprintf()] to convert that value to a UTF-16 string and returns ** the number of bytes in that string. ** ^If the result is NULL, then sqlite3_column_bytes16() returns zero. ** ** ^The values returned by [sqlite3_column_bytes()] and ** [sqlite3_column_bytes16()] do not include the zero terminators at the end ** of the string. ^For clarity: the values returned by ** [sqlite3_column_bytes()] and [sqlite3_column_bytes16()] are the number of ** bytes in the string, not the number of characters. *** ** ^Strings returned by sqlite3_column_text() and sqlite3_column_text16(), ** even empty strings, are always zero-terminated. ^The return ** value from sqlite3_column_blob() for a zero-length BLOB is a NULL pointer. ** ** Warning: ^The object returned by [sqlite3_column_value()] is an ** [unprotected sqlite3_value] object. In a multithreaded environment, ** an unprotected sqlite3_value object may only be used safely with ** [sqlite3_bind_value()] and [sqlite3_result_value()]. ** If the [unprotected sqlite3_value] object returned by ** [sqlite3_column_value()] is used in any other way, including calls ** to routines like [sqlite3_value_int()], [sqlite3_value_text()], ** or [sqlite3_value_bytes()], the behavior is not threadsafe. ** Hence, the sqlite3_column_value() interface ** is normally only useful within the implementation of ** [application-defined SQL functions] or [virtual tables], not within ** top-level application code. *** ** The these routines may attempt to convert the datatype of the result. ** ^For example, if the internal representation is FLOAT and a text result ** is requested, [sqlite3_snprintf()] is used internally to perform the ** conversion automatically. ^The following table details the conversions ** that are applied: *** *** <blockquote> ** <table border="1"> ** <tr><th> Internal
 Type <th> Requested
 Type <th> Conversion *** *** <tr><td> NULL </td> INTEGER </td> Result is 0 ** <tr><td> NULL </td> <td> FLOAT </td> Result is 0.0 ** <tr><td> NULL </td> TEXT </td> Result is a NULL pointer ** <tr><td> NULL </td> BLOB </td> Result is a NULL pointer ** <tr><td> INTEGER </td> FLOAT </td> Convert from integer to float ** <tr><td> INTEGER </td> TEXT </td> ASCII rendering of the integer ** <tr><td> INTEGER </td> BLOB </td> Same as INTEGER->TEXT ** <tr><td> FLOAT </td> INTEGER </td> [CAST] to INTEGER ** <tr><td> FLOAT </td> TEXT </td> ASCII rendering of the float ** <tr><td> FLOAT </td> BLOB </td> [CAST] to BLOB ** <tr><td> TEXT </td> INTEGER </td> [CAST] to INTEGER ** <tr><td> TEXT </td> REAL ** <tr><td> TEXT </td> BLOB </td> No change ** <tr><td> BLOB </td> INTEGER </td> [CAST] to INTEGER ** <tr><td> BLOB </td> FLOAT </td> [CAST] to REAL ** <tr><td> BLOB </td> TEXT </td> Add a zero terminator if needed ** </table> ** </blockquote> /*** Note that when type conversions occur, pointers returned by prior ** calls to sqlite3_column_blob(), sqlite3_column_text(), and/or ** sqlite3_column_text16() may be invalidated. ** Type conversions and pointer invalidations might occur ** in the following cases: *** *** ** The initial content is a BLOB and sqlite3_column_text() or ** sqlite3_column_text16() is called. A zero-terminator might ** need to be added to the string. ** The initial content is UTF-8 text and sqlite3_column_bytes16() or ** sqlite3_column_text16() is called. The content must be converted ** to UTF-16. ** The initial content is UTF-

16 text and sqlite3_column_bytes() or ** sqlite3_column_text() is called. The content must be converted ** to UTF-8. ** *** ^Conversions between UTF-16be and UTF-16le are always done in place and do ** not invalidate a prior pointer, though of course the content of the buffer ** that the prior pointer references will have been modified. Other kinds ** of conversion are done in place when it is possible, but sometimes they ** are not possible and in those cases prior pointers are invalidated. *** ^The safest policy is to invoke these routines ** in one of the following ways: *** ** sqlite3_column_text() followed by sqlite3_column_bytes() ** sqlite3_column_blob() followed by sqlite3_column_bytes() ** sqlite3_column_text16() followed by sqlite3_column_bytes16() ** *** In other words, you should call sqlite3_column_text(), ** sqlite3_column_blob(), or sqlite3_column_text16() first to force the result ** into the desired format, then invoke sqlite3_column_bytes() or ** sqlite3_column_bytes16() to find the size of the result. Do not mix calls ** to sqlite3_column_text() or sqlite3_column_blob() with calls to ** sqlite3_column_bytes16(), and do not mix calls to sqlite3_column_text16() ** with calls to sqlite3_column_bytes(). *** ^The pointers returned are valid until a type conversion occurs as ** described above, or until [sqlite3_step()] or [sqlite3_reset()] or ** [sqlite3_finalize()] is called. ^The memory space used to hold strings ** and BLOBs is freed automatically. Do not pass the pointers returned ** from [sqlite3_column_blob()], [sqlite3_column_text()], etc. into ** [sqlite3_free()]. *** ^If a memory allocation error occurs during the evaluation of any ** of these routines, a default value is returned. The default value ** is either the integer 0, the floating point number 0.0, or a NULL ** pointer. Subsequent calls to [sqlite3_errcode()] will return ** [SQLITE_NOMEM].^

label: code-design

62643. #include "sqlite3ext.h"

62644. **comment:** ** 2008 October 7 *** ^The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ^May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** ***** This file contains code use to implement an in-memory rollback journal. *** ^The in-memory rollback journal is used to journal transactions for ** ":memory:" databases and when the journal_mode=MEMORY pragma is used. *** ^Update: The in-memory journal is also used to temporarily cache ** smaller journals that are not critical for power-loss recovery. ** For example, statement journals that are not too big will be held ** entirely in memory, thus reducing the number of file I/O calls, and ** more importantly, reducing temporary file creation events. If these ** journals become too large for memory, they are spilled to disk. But ** in the common case, they are usually small and no file I/O needs to ** occur.

label: code-design

62645. Used by fts3.c only.

62646. ** CAPI3REF: Copy And Free SQL Values ** METHOD: sqlite3_value *** ^The sqlite3_value_dup(V) interface makes a copy of the [sqlite3_value] ** object D and returns a pointer to that copy. ^The [sqlite3_value] returned ** is a [protected sqlite3_value] object even if the input is not. ** ^The sqlite3_value_dup(V) interface returns NULL if V is NULL or if a ** memory allocation fails. *** ^The sqlite3_value_free(V) interface frees an [sqlite3_value] object ** previously obtained from [sqlite3_value_dup()]. ^If V is a NULL pointer ** then sqlite3_value_free(V) is a harmless no-op.

62647. WAL_RDWR, WAL_RDONLY, or WAL_SHM_RDONLY

62648. ** This function is used as part of the big assert() procedure implemented by ** fts5AssertMultiIterSetup(). It ensures that the result currently stored ** in *pRes is the correct result of comparing the current positions of the ** two iterators.

62649. **comment:** ** Reassign page numbers so that the new pages are in ascending order. ** This helps to keep entries in the disk file in order so that a scan ** of the table is closer to a linear scan through the file. That in turn ** helps the operating system to deliver pages from the disk more rapidly. *** ^An O(n^2) insertion sort algorithm is used, but since n is never more ** than (NB+2) (a small constant), that should not be a problem. *** ^When NB==3, this one optimization makes the database about 25% faster ** for large insertions and deletions.

label: code-design

62650. The index of the cell within the node

62651. 890

62652. Next select to the left in a compound

62653. TUNING: One-time cost for computing the automatic index is ** estimated to be X*N*log2(N) where N is the number of rows in ** the table being indexed and where X is 7 (LogEst=28) for normal ** tables or 1.375 (LogEst=4) for views and subqueries. The value ** of X is smaller for views and subqueries so that the query planner ** will be more aggressive about generating automatic indexes for ** those objects, since there is no opportunity to add schema ** indexes on subqueries and views.

62654. ** The following global variable is incremented whenever the ** transfer optimization is used. This is used for testing ** purposes only - to make sure the transfer optimization really ** is happening when it is supposed to.

62655. Second part of the name of the table or view

62656. Leave error message here

62657. ** This function is called when an UPDATE or DELETE operation is being ** compiled on table pTab, which is the parent table of foreign-key pFKey. ** If the current operation is an UPDATE, then the pChanges parameter is ** passed a pointer to the list of columns being modified. If it is a ** DELETE, pChanges is passed a NULL pointer. *** ^It returns a pointer to a Trigger structure containing a trigger ** equivalent to the ON UPDATE or ON DELETE action specified by pFKey. ** If the action is "NO ACTION" or "RESTRICT", then a NULL pointer is ** returned (these actions require no special handling by the triggers ** subsystem, code for them is created by fkScanChildren()). *** ^For example, if pFKey is the foreign key and pTab is table "p" in ** the following schema: *** ^CREATE TABLE p(pk PRIMARY KEY); ** ^CREATE TABLE c(ck REFERENCES p ON DELETE CASCADE); *** ^then the returned trigger structure is equivalent to: *** ^CREATE TRIGGER ... DELETE ON p BEGIN ** DELETE FROM c WHERE ck = old.pk; ** END; *** ^The returned pointer is cached as part of the foreign key object. It ** is eventually freed along with the rest of the foreign key object by ** sqlite3FkDelete().

62658. Figure out if we have any triggers and if the table being ** deleted from is a view

62659. ** Reset the aggregate accumulator. ** ^The aggregate accumulator is a set of memory cells that hold ** intermediate results while calculating an aggregate. This ** routine generates code that stores NULLs in all of those memory ** cells.

62660. Wal-index header for current transaction

62661. ** sqlite3_test_control(FAULT_INSTALL, xCallback) *** ^Arrange to invoke xCallback() whenever sqlite3FaultSim() is called, ** if xCallback is not NULL. ** ^As a test of the fault simulator mechanism itself, sqlite3FaultSim(0) ** is called immediately after installing the new callback and the return ** value from sqlite3FaultSim(0) becomes the return from ** sqlite3_test_control().

62662. Column number of the index

62663. **comment:** PmaReaders should be normally initialized in order, as if they are ** reading from the same temp file this makes for more linear file IO. ** However, in the INCRINIT_ROOT case, if PmaReader aReadr[nTask-1] is ** in use it will block the vdbePmaReaderNext() call while it uses ** the main thread to fill its buffer. So calling PmaReaderNext() ** on this PmaReader before any of the multi-threaded PmaReaders takes ** better advantage of multi-processor hardware.

label: code-design

62664. Make sure there are enough memory cells allocated to accommodate ** the regPrev array and a trailing rowid (the rowid slot is required ** when building a record to insert into the sample column of ** the sqlite_stat4 table).

62665. ** This function is used to read a single varint from a buffer. Parameter ** pEnd points 1 byte past the end of the buffer. When this function is ** called, if *pp points to pEnd or greater, then the end of the buffer ** has been reached. In this case *pp is set to 0 and the function returns. *** ^If *pp does not point to or past pEnd, then a single varint is read ** from *pp. *pp is then set to point 1 byte past the end of the read varint. *** ^If bDescIdx is false, the value read is added to *pVal before returning. ** If it is true, the value read is subtracted from *pVal before this ** function returns.

62666. Strategy index

62667. This is the case if the data for the INSERT is coming from a ** single-row VALUES clause

62668. Delete the %_content record

62669. **comment:** The first time a column affinity string for a particular index is ** required, it is allocated and populated here. It is then stored as ** a member of the Index structure for subsequent use. *** ^The column affinity string will eventually be deleted by ** sqliteDeleteIndex() when the Index structure itself is cleaned ** up.

label: code-design

62670. ** Let any sequence of one or more vowels be represented by V and let ** C be sequence of one or more consonants. Then every word can be ** represented as: *** ^[C] (VC){m} [V] *** ^In prose: A word is an optional consonant followed by zero or ** vowel-consonant pairs followed by an optional vowel. "m" is the ** number of vowel consonant pairs. This routine computes the value ** of m for the first i bytes of a word. *** ^Return true if the m-value for z is 1 or more. In other words, ** return true if z contains at least one vowel that is followed ** by a consonant. *** ^In this routine z[] is in reverse order. So we are really looking ** for an instance of a consonant followed by a vowel.

62671. return true if number and no extra non-whitespace characters after

62672. ** Each SQLite module (virtual table definition) is defined by an ** instance of the following structure, stored in the sqlite3.aModule ** hash table.

62673. ** Increment the r-tree reference count.
62674. Any tokens larger than this (in bytes) are passed through without ** stemming.
62675. #include "config.h"
62676. OUTER
62677. If the P3 value cannot be converted into any kind of a number, ** then the seek is not possible, so jump to P2
62678. The evaluation of the IN/EXISTS/SELECT must be repeated every time it ** is encountered if any of the following is true: *** * The right-hand side is a correlated subquery *** The right-hand side is an expression list containing variables *** We are inside a trigger *** If all of the above are false, then we can run this code just once ** save the results, and reuse the same result on subsequent invocations.
62679. ** Return the ROWID of the most recent insert
62680. Database connection to notify of OOM errors
62681. Name of indexed column
62682. Savepoint name (nul-terminated)
62683. ** Lower the locking level on file descriptor pFile to eFileLock. eFileLock ** must be either NO_LOCK or SHARED_LOCK. *** If the locking level of the file descriptor is already at or below ** the requested locking level, this routine is a no-op. *** When the locking level reaches NO_LOCK, delete the lock file.
62684. Offset in pFile
62685. New name of FTS5 table
62686. Value to pass to log callback (or 0)
62687. Copy of pOrig
62688. Printf format string to append
62689. Checksum for wal-header
62690. Set to cheapest remaining token.
62691. VM being reprepared (sqlite3Reprepare())
62692. ** CAPI3REF: Status Parameters for prepared statements ** KEYWORDS: {SQLITE_STMTSTATUS counter} {SQLITE_STMTSTATUS counters} *** These preprocessor macros define integer codes that name counter ** values associated with the [sqlite3_stmt_status()] interface. ** The meanings of the various counters are as follows: *** ** <dl> ** [[SQLITE_STMTSTATUS_FULLSCAN_STEP]] <dt>SQLITE_STMTSTATUS_FULLSCAN_STEP</dt> ** <dd>^This is the number of times that SQLite has stepped forward in ** a table as part of a full table scan. Large numbers for this counter ** may indicate opportunities for performance improvement through ** careful use of indices.</dd> ** [[SQLITE_STMTSTATUS_SORT]] <dt>SQLITE_STMTSTATUS_SORT</dt> ** <dd>^This is the number of sort operations that have occurred. ** A non-zero value in this counter may indicate an opportunity to ** improvement performance through careful use of indices.</dd> ** [[SQLITE_STMTSTATUS_AUTOINDEX]] <dt>SQLITE_STMTSTATUS_AUTOINDEX</dt> ** <dd>^This is the number of rows inserted into transient indices that ** were created automatically in order to help joins run faster. ** A non-zero value in this counter may indicate an opportunity to ** improvement performance by adding permanent indices that do not ** need to be reinitialized each time the statement is run.</dd> ** [[SQLITE_STMTSTATUS_VM_STEP]] <dt>SQLITE_STMTSTATUS_VM_STEP</dt> ** <dd>^This is the number of virtual machine operations executed ** by the prepared statement if that number is less than or equal ** to 2147483647. The number of virtual machine operations can be ** used as a proxy for the total work done by the prepared statement. ** If the number of virtual machine operations exceeds 2147483647 ** then the value returned by this statement status code is undefined. ** [[SQLITE_STMTSTATUS_REPREPARE]] <dt>SQLITE_STMTSTATUS_REPREPARE</dt> ** <dd>^This is the number of times that the prepare statement has been ** automatically regenerated due to schema changes or change to ** [bound parameters] that might affect the query plan. *** [[SQLITE_STMTSTATUS_RUN]] <dt>SQLITE_STMTSTATUS_RUN</dt> ** <dd>^This is the number of times that the prepared statement has ** been run. A single "run" for the purposes of this counter is one ** or more calls to [sqlite3_step()] followed by a call to [sqlite3_reset()]. ** The counter is incremented on the first [sqlite3_step()] call of each ** cycle. ** [[SQLITE_STMTSTATUS_MEMUSED]] <dt>SQLITE_STMTSTATUS_MEMUSED</dt> ** <dd>^This is the approximate number of bytes of heap memory ** used to store the prepared statement. ^This value is not actually ** a counter, and so the resetFlag parameter to sqlite3_stmt_status() ** is ignored when the opcode is SQLITE_STMTSTATUS_MEMUSED. ** <dd> ** </dl>
62693. pList!=0 if pF->pFunc has NEEDCOLL
62694. pDone never used on non-savepoint
62695. First argument to xCmp()
62696. Configuration cookie value
62697. Next in list of all blocked connections
62698. True if this is a covering index
62699. Store results as keys in an index
62700. Total size of the database file
62701. Advance zCs to the next token. Store that token type in 'token', ** and its length in 'len' (to be used next iteration of this loop).
62702. Arguments to rank function
62703. Checksum on the header content
62704. ** Sometimes, after a file handle is closed by SQLite, the file descriptor ** cannot be closed immediately. In these cases, instances of the following ** structure are used to store the file descriptor while waiting for an ** opportunity to either close or reuse it.
62705. **comment:** Next unused file descriptor on same file
label: code-design
62706. True to defer constraints
62707. ** On machines with a small stack size, you can redefine the ** SQLITE_PRINT_BUF_SIZE to be something smaller, if desired.
62708. Default value for the new column
62709. The object to be destroyed
62710. ** Create a new sqlite3_value object, containing the value of pExpr. *** This only works for very simple expressions that consist of one constant ** token (i.e. "5", "5.1", "a string"). If the expression can ** be converted directly into a value, then the value is allocated and ** a pointer written to *ppVal. The caller is responsible for deallocating ** the value by passing it to sqlite3ValueFree() later on. If the expression ** cannot be converted to a value, then *ppVal is set to NULL.
62711. Free the connection p
62712. Instead of invoking the conflict handler, append the change blob ** to the SessionApplyCtx.constraints buffer.
62713. ** Return UTF-8 encoded English language explanation of the most recent ** error.
62714. **comment:** ** The replace() function. Three arguments are all strings: call ** them A, B, and C. The result is also a string which is derived ** from A by replacing every occurrence of B with C. The match ** must be exact. Collating sequences are not used.
label: code-design
62715. ** Release a reference to data record returned by an earlier call to ** fts5DataRead().
62716. Table for associated with an index on expr, or negative ** of the base register during check-constraint eval
62717. ** The following object holds a copy of the wal-index header content. *** The actual header in the wal-index consists of two copies of this ** object followed by one instance of the WalCkptInfo object. ** For all versions of SQLite through 3.10.0 and probably beyond, ** the locking bytes (WalCkptInfo.aLock) start at offset 120 and ** the total header size is 136 bytes. *** The szPage value can be any power of 2 between 512 and 32768, inclusive. ** Or it can be 1 to represent a 65536-byte page. The latter case was ** added in 3.7.1 when support for 64K pages was added.
62718. ** Returns non-zero if the specified path name should be used verbatim. If ** non-zero is returned from this function, the calling function must simply ** use the provided path name verbatim -OR- resolve it into a full path name ** using the GetFullPathName Win32 API function (if available).
62719. Page size
62720. RTree to search
62721. inner
62722. Least sig. 32 bits of new offset
62723. This function currently works by first transforming the UTF-16 ** encoded string to UTF-8, then invoking sqlite3_prepare(). The ** tricky bit is figuring out the pointer to return in *pzTail.
62724. ** Returns non-zero if the character should be treated as a directory ** separator.
62725. TABLE => nothing
62726. The root page of the b-tree is overfull. In this case call the ** balance_deeper() function to create a new child for the root-page ** and copy the current contents of the root-page to it. The ** next iteration of the do-loop will balance the child page.

62727. ***** Continuing where we left off in mutex_w32.c *****

62728. ** Generate code that will evaluate expression pExpr and store the ** results in register target. The results are guaranteed to appear ** in register target.

62729. Delete all indices associated with this table.

62730. Grow the output buffer so that there is sufficient space to fit the ** largest possible utf-8 character.

62731. ** Return the SQLITE_PREPARE flags for a Vdbe.

62732. ** Return the sqlite3* database handle to which the prepared statement given ** in the argument belongs. This is the same database handle that was ** the first argument to the sqlite3_prepare() that was used to create ** the statement in the first place.

62733. Set to true if constraints may cause a replace

62734. joinop ::= COMMAJOIN

62735. A row is being added to the child table. If a parent row cannot ** be found, adding the child row has violated the FK constraint. *** If this operation is being performed as part of a trigger program ** that is actually a "SET NULL" action belonging to this very ** foreign key, then omit this scan altogether. As all child key ** values are guaranteed to be NULL, it is not possible for adding ** this row to cause an FK violation.

62736. Request is out of range. Return a transient error.

62737. MUST BE LAST (see below)

62738. If the first attempt failed, it might have been due to a race ** with a writer. So get a WRITE lock and try again.

62739. **comment:** ** If we compile with the SQLITE_TEST macro set, then the following block ** of code will give us the ability to simulate a disk I/O error. This ** is used for testing the I/O recovery logic.

label: code-design

62740. Page cache allocations

62741. **comment:** This term must be of the form t1.a==t2.b where t2 is in the ** chngToIN set but t1 is not. This term will be either preceded ** or followed by an inverted copy (t2.b==t1.a). Skip this term ** and use its inversion.

label: code-design

62742. Write the referenced table cursor and column here

62743. ** The file suffix added to the data base filename in order to create the ** lock directory.

62744. Cursor of the table that is being indexed

62745. **comment:** ** CAPI3REF: Dynamically Typed Value Object ** KEYWORDS: {protected sqlite3_value} {unprotected sqlite3_value} *** SQLite uses the sqlite3_value object to represent all values ** that can be stored in a database table. SQLite uses dynamic typing ** for the values it stores. ^Values stored in sqlite3_value objects ** can be integers, floating point values, strings, BLOBS, or NULL. *** An sqlite3_value object may be either "protected" or "unprotected". ** Some interfaces require a protected sqlite3_value. Other interfaces ** will accept either a protected or an unprotected sqlite3_value. ** Every interface that accepts sqlite3_value arguments specifies ** whether or not it requires a protected sqlite3_value. The ** [sqlite3_value_dup()] interface can be used to construct a new ** protected sqlite3_value from an unprotected sqlite3_value. *** The terms "protected" and "unprotected" refer to whether or not ** a mutex is held. An internal mutex is held for a protected ** sqlite3_value object but no mutex is held for an unprotected ** sqlite3_value object. If SQLite is compiled to be single-threaded ** (with [SQLITE_THREADSAFE=0] and with [sqlite3_threadsafe()] returning 0) ** or if SQLite is run in one of reduced mutex modes ** [SQLITE_CONFIG_SINGLETHREAD] or [SQLITE_CONFIG_MULTITHREAD] ** then there is no distinction between protected and unprotected ** sqlite3_value objects and they can be used interchangeably. However, ** for maximum code portability it is recommended that applications ** still make the distinction between protected and unprotected ** sqlite3_value objects even when not strictly required. *** ^The sqlite3_value objects that are passed as parameters into the ** implementation of [application-defined SQL functions] are protected. ** ^The sqlite3_value object returned by ** [sqlite3_column_value()] is unprotected. ** Unprotected sqlite3_value objects may only be used with ** [sqlite3_result_value()] and [sqlite3_bind_value()]. ** The [sqlite3_value_blob | sqlite3_value_type()] family of ** interfaces require protected sqlite3_value objects.

label: code-design

62746. For an index btree, save the complete key content

62747. OUT: Mask of TRIGGER_BEFORE|TRIGGER_AFTER

62748. Tail recursion

62749. RECURSIVE

62750. Entries in aOvfl[]

62751. OUT: visibility of the cell

62752. Remove any and all aliases between the result set and the ** GROUP BY clause.

62753. Term searched for (or NULL)

62754. Size of term prefix

62755. MUTEX_STATIC_LRU or NULL

62756. How to allocate space. Or NULL

62757. trigger_cmd

62758. Pointer to Fts5InsertCtx object

62759. If the statement transaction is being rolled back, also restore the ** database handles deferred constraint counter to the value it had when ** the statement transaction was opened.

62760. Data yet to be written

62761. ** Free as much memory as possible from the pager.

62762. **comment:** Character classes for tokenizing *** In the sqlite3GetToken() function, a switch() on aiClass[c] is implemented ** using a lookup table, whereas a switch() directly on c uses a binary search. ** The lookup table is much faster. To maximize speed, and to ensure that ** a lookup table is used, all of the classes need to be small integers and ** all of them need to be used within the switch.

label: code-design

62763. wait 0.5 sec and try the lock again

62764. Initialize a new parser that has already been allocated.

62765. ** Helper function for exprIsDeterministic().

62766. SELECT to the right of IN operator

62767. Name of the index

62768. trnm ::= nm DOT nm

62769. API offered by current FTS version

62770. If a transaction is still open on the Btree, roll it back.

62771. Read the old.* and new.* records for the update change.

62772. ** Register the ICU extension functions with database db.

62773. ** Return the ceiling of the logarithm base 2 of iValue. *** Examples: memsys5Log(1) -> 0 ** memsys5Log(2) -> 1 ** memsys5Log(4) -> 2 ** memsys5Log(5) -> 3 ** memsys5Log(8) -> 3 ** memsys5Log(9) -> 4

62774. ** A pointer to a structure of the following type is passed as the ** argument to scored geometry callback registered using ** sqlite3_rtree_query_callback(). *** Note that the first 5 fields of this structure are identical to ** sqlite3_rtree_geometry. This structure is a subclass of ** sqlite3_rtree_geometry.

62775. Address of "next_row":

62776. Fields below are only available in SQLite 3.9.0 and later

62777. Btree cursor used for pIndex

62778. The LEMON-generated LALR(1) parser

62779. 3*nCol*nPhrase values

62780. ** This routine is used to check if the UTF-8 string zName is a legal ** unqualified name for a new schema object (table, index, view or ** trigger). All names are legal except those that begin with the string ** "sqlite_" (in upper, lower or mixed case). This portion of the namespace ** is reserved for internal use.

62781. Run-cost of the new entry

62782. If the overflow page-list cache has been allocated and the ** entry for the first required overflow page is valid, skip ** directly to it.

62783. cmd ::= createkw uniqueflag INDEX ifnotexists nm dbnm ON nm LP sortlist RP where_opt

62784. **comment:** PragTyp_XXX value

label: code-design

62785. Special case - lookup by rowid.

62786. Rootpage of table being initialized
62787. Database handle
62788. New root page image
62789. ** The following routine is called if the stack overflows.
62790. String inserted between snippets
62791. ***** Begin file vdbe.h *****
62792. **comment:** The testcase() macro should already be defined in the amalgamation. If ** it is not, make it a no-op.
 label: code-design
62793. OUT: Next overflow page number
62794. PRIMARY KEY index for WITHOUT ROWID tables
62795. Least recently used entry has the smallest value
62796. Restriction (19)
62797. To downgrade to shared, simply update our internal notion of the ** lock state. No need to mess with the file on disk.
62798. **comment:** ** Lock the virtual table so that it cannot be disconnected. ** Locks nest. Every lock should have a corresponding unlock. ** If an unlock is omitted, resources leaks will occur. ** ** If a disconnect is attempted while a virtual table is locked, ** the disconnect is deferred until all locks have been removed.
 label: code-design
62799. SQLITE_STMTSTATUS_FULLSCAN_STEP
62800. Register holding co-routine entry-point
62801. Opcode: OpenRead P1 P2 P3 P4 P5 ** Synopsis: root=P2 iDb=P3 *** Open a read-only cursor for the database table whose root page is ** P2 in a database file. The database file is determined by P3. ** P3==0 means the main database, P3==1 means the database used for ** temporary tables, and P3>1 means used the corresponding attached ** database. Give the new cursor an identifier of P1. The P1 ** values need not be contiguous but all P1 values should be small integers. ** It is an error for P1 to be negative. ** ** If P5!=0 then use the content of register P2 as the root page, not ** the value of P2 itself. ** ** There will be a read lock on the database whenever there is an ** open cursor. If the database was unlocked prior to this instruction ** then a read lock is acquired as part of this instruction. A read ** lock allows other processes to read the database but prohibits ** any other process from modifying the database. The read lock is ** released when all cursors are closed. If this instruction attempts ** to get a read lock but fails, the script terminates with an ** SQLITE_BUSY error code. ** ** The P4 value may be either an integer (P4_INT32) or a pointer to ** a KeyInfo structure (P4_KEYINFO). If it is a pointer to a KeyInfo ** structure, then said structure defines the content and collating ** sequence of the index being opened. Otherwise, if P4 is an integer ** value, it is set to the number of columns in the table. ** ** See also: OpenWrite, ReopenIdx
62802. Second argument for valueNew()
62803. Compare function to use
62804. If this is stat4 data, then calculate aAvgEq[] values for all ** sample columns except the last. The last is always set to 1, as ** once the trailing PK fields are considered all index keys are ** unique.
62805. ** Possible values for WAL.readOnly
62806. Index of right-hand Fts5SegIter
62807. ** Return the bitmask for the given cursor number. Return 0 if ** iCursor is not in the set.
62808. SQLITE_INROSPECTION_PRAGMAS
62809. **comment:** Some compilers complain about constants of the form 0x7fffffffffffff. ** Others complain about 0x7fffffffffffffLL. The following macro seems ** to provide the constant while making all compilers happy.
 label: code-design
62810. SQLITE_OPEN_EXCLUSIVE is used to make sure that a new file is ** created. SQLite doesn't use it to indicate "exclusive access" ** as it is usually understood.
62811. The table to code triggers from
62812. **comment:** Initialize all term iterators in the NEAR object.
 label: code-design
62813. **comment:** ** Macro used to suppress compiler warnings for unused parameters.
 label: code-design
62814. ** This function is also purely an internal test. It does not contribute to ** FTS functionality, or even the integrity-check, in any way.
62815. Zero all hash-table entries that correspond to frame numbers greater ** than pWal->hdr.mxFrame.
62816. Offset into the file to begin writing at
62817. ** This routine will drop an existing named index. This routine ** implements the DROP INDEX statement.
62818. SELECT to read %_segdir entry
62819. **comment:** ** This API allows applications to modify the global configuration of ** the SQLite library at run-time. ** ** This routine should only be called when there are no outstanding ** database connections or memory allocations. This routine is not ** threadsafe. Failure to heed these warnings can lead to unpredictable ** behavior.
 label: code-design
62820. Hint read from %_stat table
62821. Current block image
62822. ** This version of the memory allocator is used only when ** SQLITE_ENABLE_MEMSYS5 is defined.
62823. ** Implementation of the sqlite3_pcache.xDestroy method. ** ** Destroy a cache allocated using pcache1Create().
62824. 0x1d
62825. ** Check to see if the given expression is a LIKE or GLOB operator that ** can be optimized using inequality constraints. Return TRUE if it is ** so and false if not. ** ** In order for the operator to be optimizable, the RHS must be a string ** literal that does not begin with a wildcard. The LHS must be a column ** that may only be NULL, a string, or a BLOB, never a number. (This means ** that virtual tables cannot participate in the LIKE optimization.) The ** collating sequence for the column on the LHS must be appropriate for ** the operator.
62826. REPLACE_DOCSIZE
62827. Obtain a reference to the index structure and allocate a new segment-id ** for the new level-0 segment.
62828. Cannot EXPRDUP_REDUCE this Expr
62829. Part of a parenthesized FROM clause
62830. Apply change to "main" db of this handle
62831. ** Parameter zSrcData points to a buffer containing the data for ** page iSrcPg from the source database. Copy this data into the ** destination database.
62832. ***** Begin file whereexpr.c *****
62833. Number of bytes yet to be written
62834. ** Allowed values for the NameContext, ncFlags field. ** ** Value constraints (all checked via assert()): ** NC_HasAgg == SF_HasAgg ** NC_MinMaxAgg == SF_MinMaxAgg == SQLITE_FUNC_MINMAX **
62835. Each iteration of this loop gobbles up a contiguous run of separators, ** then the next token.
62836. If the database may be auto-vacuum capable (if SQLITE_OMIT_AUTOVACUUM ** is not defined), then it is important to call OP_Destroy on the ** table and index root-pages in order, starting with the numerically ** largest root-page number. This guarantees that none of the root-pages ** to be destroyed is relocated by an earlier OP_Destroy. i.e. if the ** following were coded: ** ** OP_Destroy 4 0 ** ... ** OP_Destroy 5 0 ** ** and root page 5 happened to be the largest root-page number in the ** database, then root page 5 would be moved to page 4 by the ** "OP_Destroy 4 0" opcode. The subsequent "OP_Destroy 5 0" would hit ** a free-list page.
62837. ** Process a modifier to a date-time stamp. The modifiers are ** as follows: ** ** NNN days ** NNN hours ** NNN minutes ** NNN.NNNNN seconds ** NNN months ** NNN years ** start of month ** start of year ** start of week ** start of day ** weekday N ** unixepoch ** localtime ** utc ** ** Return 0 on success and 1 if there is any kind of error. If the error ** is in a system call (i.e. localtime()), then an error message is written ** to context pCtx. If the error is an unrecognized modifier, no error is ** written to pCtx.
62838. ** The buffer pointed to by argument zNode (size nNode bytes) contains an ** interior node of a b-tree segment. The zTerm buffer (size nTerm bytes) ** contains a term. This function searches the sub-tree headed by the zNode ** node for the range of leaf nodes that may contain the specified term ** or terms for which the specified term is a prefix. ** ** If piLeaf is not NULL, then *piLeaf is set to the blockid of the ** left-most leaf node in the tree that may contain the specified term. ** If piLeaf2 is not NULL, then *piLeaf2 is set to the blockid of the ** right-most leaf node that may contain a term for which the specified ** term is a prefix. ** ** It is possible that the range of returned leaf nodes does not contain ** the specified term or any terms for which it is a prefix. However, if the **

segment does contain any such terms, they are stored within the identified ** range. Because this function only inspects interior segment nodes (and ** never loads leaf nodes into memory), it is not possible to be sure. *** If an error occurs, an error code other than SQLITE_OK is returned.

62839. ** CAPI3REF: Changegroup Handle

62840. ** Flush the contents of memory to a real file on disk.

62841. Search for self-joins in this FROM clause

62842. Number of columns in index + pk/rowid

62843. If nBest is still 0, then the index must be empty.

62844. In the absence of explicit truth probabilities, use heuristics to ** guess a reasonable truth probability.

62845. F_UNLCK, F_RDLCK, or F_WRLCK

62846. Loop addresses

62847. distinct ::= ALL

62848. IN/OUT: Size of buffer *ppList in bytes

62849. #include "wal.h"

62850. Bytes of memory required

62851. ON UPDATE action

62852. Index of this phrase in matchinfo() results

62853. ** Expression pVar is guaranteed to be an SQL variable. pExpr may be any ** type of expression. ** ** If pExpr is a simple SQL value - an integer, real, string, blob ** or NULL value - then the VDBE currently being prepared is configured ** to re-prepare each time a new value is bound to variable pVar. ** **

Additionally, if pExpr is a simple SQL value and the value is the ** same as that currently bound to variable pVar, non-zero is returned. ** Otherwise, if the values are not the same or if pExpr is not a simple ** SQL value, zero is returned.

62854. Get an appropriate name for the column

62855. Current insert batch

62856. ** Advance the cursor to the next row in the table that matches the ** search criteria. ** ** Return SQLITE_OK if nothing goes wrong. SQLITE_OK is returned ** even if we reach end-of-file. The fts5EofMethod() will be called ** subsequently to determine whether or not an EOF was hit.

62857. Factor out constant terms

62858. ** Return SQLITE_LOCKED_SHAREDCACHE if another user of the same shared ** btree as the argument handle holds an exclusive lock on the ** sqlite_master table. Otherwise SQLITE_OK.

62859. ** CAPI3REF: Last Insert Rowid ** METHOD: sqlite3 *** ^Each entry in most SQLite tables (except for [WITHOUT ROWID] tables) ** has a unique 64-bit signed ** integer key called the [ROWID | "rowid"]. ^The rowid is always available ** as an undeclared column named ROWID, OID, or _ROWID_ as long as those ** names are not also used by explicitly declared columns. ^If ** the table has a column of type [INTEGER PRIMARY KEY] then that column ** is another alias for the rowid. *** ^The sqlite3_last_insert_rowid(D) interface usually returns the [rowid] of ** the most recent successful [INSERT] into a rowid table or [virtual table] ** on database connection D. ^Inserts into [WITHOUT ROWID] tables are not ** recorded. ^If no successful [INSERT]s into rowid tables have ever occurred ** on the database connection D, then sqlite3_last_insert_rowid(D) returns ** zero. ** ** As well as being set automatically as rows are inserted into database ** tables, the value returned by this function may be set explicitly by ** [sqlite3_set_last_insert_rowid()] ** ** Some virtual table implementations may INSERT rows into rowid tables as ** part of committing a transaction (e.g. to flush data accumulated in memory ** to disk). In this case subsequent calls to this function return the rowid ** associated with these internal INSERT operations, which leads to ** unintuitive results. Virtual table implementations that do write to rowid ** tables in this way can avoid this problem by restoring the original ** rowid value using [sqlite3_set_last_insert_rowid()] before returning ** control to the user. *** ^If an [INSERT] occurs within a trigger then this routine will ** return the [rowid] of the inserted row as long as the trigger is ** running. Once the trigger program ends, the value returned ** by this routine reverts to what it was before the trigger was fired.)^ ** ** ^An [INSERT] that fails due to a constraint violation is not a ** successful [INSERT] and does not change the value returned by this ** routine. ^Thus INSERT OR FAIL, INSERT OR IGNORE, INSERT OR ROLLBACK, ** and INSERT OR ABORT make no changes to the return value of this ** routine when their insertion fails. ^When INSERT OR REPLACE ** encounters a constraint violation, it does not fail. The ** INSERT continues to completion after deleting rows that caused ** the constraint problem so INSERT OR REPLACE will always change ** the return value of this interface.)^ ** ** ^For the purposes of this routine, an [INSERT] is considered to ** be successful even if it is subsequently rolled back. ** ** This function is accessible to SQL statements via the ** [last_insert_rowid()] SQL function. ** ** If a separate thread performs a new [INSERT] on the same ** database connection while the [sqlite3_last_insert_rowid()] ** function is running and thus changes the last insert [rowid], ** then the value returned by [sqlite3_last_insert_rowid()] is ** unpredictable and might not equal either the old or the new ** last insert [rowid].

62860. Check to see if a legacy fts3 table has been "upgraded" by the ** addition of a %_stat table so that it can use incremental merge.

62861. ** Free a WhereInfo structure

62862. Estimate the upper limit on the number of leaf nodes in a new segment ** created by merging the oldest :2 segments from absolute level :1. See ** function sqlite3Fts3Incrmerge() for details.

62863. ** Return the approximate number of bytes of memory currently ** used by the pager and its associated cache.

62864. Write the resulting vtab structure here

62865. Tokenize return code

62866. True if attempting to append

62867. If the fts5YNOERRORRECOVERY macro is defined, then do not attempt to ** do any kind of error recovery. Instead, simply invoke the syntax ** error routine and continue going as if nothing had happened. ** ** Applications can set this macro (for example inside %include) if ** they intend to abandon the parse upon the first syntax error seen.

62868. An existing page in the cache

62869. Storage for BLOB written into %_stat

62870. Sync at the end of each transaction

62871. Position lists for current row

62872. ** Return the database associated with the Vdbe.

62873. ***** End of pcache.h *****

62874. True if sPoint is valid

62875. If opening an attached database, the encoding much match ENC(db)

62876. Name of column in the right table

62877. SQLITE_DEBUG && !defined(SQLITE_MUTEX OMIT)

62878. ** Join two expressions using an AND operator. If either expression is ** NULL, then just return the other expression. ** ** If one side or the other of the AND is known to be false, then instead ** of returning an AND expression, just return a constant expression with ** a value of false.

62879. Sync the WAL to disk

62880. ** Each tokenizer module registered with the FTS5 module is represented ** by an object of the following type. All such objects are stored as part ** of the Fts5Global.pTok list.

62881. OUT: MemPage handle (may be NULL)

62882. True if analyzing arguments to an agg func

62883. **comment:** Clean up the rbu_tmp_xxx table for the previous table. It ** cannot be dropped as there are currently active SQL statements. ** But the contents can be deleted.

label: code-design

62884. ***** End of date.c *****

62885. Backup process error code

62886. ** When this function is called, buffer *ppList (size *pnList bytes) contains ** a position list that may (or may not) feature multiple columns. This ** function adjusts the pointer *ppList and the length *pnList so that they ** identify the subset of the position list that corresponds to column iCol. ** ** If there are no entries in the input position list for column iCol, then ** *pnList is set to zero before returning. ** ** If parameter bZero is non-zero, then any part of the input list following ** the end of the output list is zeroed before returning.

62887. QUERY

62888. OUT: Number of bytes in output changeset

62889. Offset into the data

62890. Pointer to start of phrase position list

62891. True if in a normal write operation
62892. ** This file contains the implementation of an in-memory hash table used ** to accumulate "term -> doclist" content before it is flushed to a level-0 ** segment.
62893. cross
62894. Deferred constraints are stored here
62895. Insert an OP_Halt at the end of the sub-program.
62896. Address of MustBeInt instruction
62897. ** Write data to an rbuVfs-file.
62898. ** Given a token, return a string that consists of the text of that ** token. Space to hold the returned string ** is obtained from sqliteMalloc() and must be freed by the calling ** function. *** Any quotation marks (ex: "name", 'name', [name], or `name`) that ** surround the body of the token are removed. *** Tokens are often just pointers into the original SQL text and so ** are not \000 terminated and are not persistent. The returned string ** is \000 terminated and is persistent.
62899. ** PRAGMA [schema].auto_vacuum ** PRAGMA [schema].auto_vacuum=N *** Get or set the value of the database 'auto-vacuum' parameter. ** The value is one of: 0 NONE 1 FULL 2 INCREMENTAL
62900. Start of column names in pragCName[]
62901. The offset for WriteFile.
62902. ** This function is used to allocate and populate UnpackedRecord ** structures intended to be compared against sample index keys stored ** in the sqlite_stat4 table. *** A single call to this function populates zero or more fields of the ** record starting with field iVal (fields are numbered from left to ** right starting with 0). A single field is populated if: *** * (pExpr==0). In this case the value is assumed to be an SQL NULL, *** * The expression is a bound variable, and this is a reprepare, or *** * The sqlite3ValueFromExpr() function is able to extract a value ** from the expression (i.e. the expression is a literal value). *** Or, if pExpr is a TK_VECTOR, one field is populated for each of the ** vector components that match either of the two latter criteria listed ** above. *** Before any value is appended to the record, the affinity of the ** corresponding column within index pIdx is applied to it. Before ** this function returns, output parameter *pnExtract is set to the ** number of values appended to the record. *** When this function is called, *ppRec must either point to an object ** allocated by an earlier call to this function, or must be NULL. If it ** is NULL and a value can be successfully extracted, a new UnpackedRecord ** is allocated (and *ppRec set to point to it) before returning. *** Unless an error is encountered, SQLITE_OK is returned. It is not an ** error if a value cannot be extracted from pExpr. If an error does ** occur, an SQLite error code is returned.
62903. We are trying for an exclusive lock but another thread in this ** same process is still holding a shared lock.
62904. If pSortCsr is non-NULL, then this call is being made as part of ** processing for a "... MATCH <expr> ORDER BY rank" query (ePlan is ** set to FT5_PLAN_SORTED_MATCH). pSortCsr is the cursor that will ** return results to the user for this query. The current cursor ** (pCursor) is used to execute the query issued by function ** fts5CursorFirstSorted() above.
62905. ***** Begin file pcache.h *****
62906. Verify that the every entry in the mapping region is reachable ** via the hash table. This turns out to be a really, really expensive ** thing to check, so only do this occasionally - not on every ** iteration.
62907. It is not safe to allow the reader to continue here if frames ** may have been appended to the log before READ_LOCK(0) was obtained. ** When holding READ_LOCK(0), the reader ignores the entire log file, ** which implies that the database file contains a trustworthy ** snapshot. Since holding READ_LOCK(0) prevents a checkpoint from ** happening, this is usually correct. *** However, if frames have been appended to the log (or if the log ** is wrapped and written for that matter) before the READ_LOCK(0) ** is obtained, that is not necessarily true. A checkpointer may ** have started to backfill the appended frames but crashed before ** it finished. Leaving a corrupt image in the database file.
62908. If azData[0] is not an SQL NULL value, it is the rowid of a ** record to delete from the r-tree table. The following block does ** just that.
62909. Destination database handle
62910. ** Enable or disable extension loading. Extension loading is disabled by ** default so as not to open security holes in older applications.
62911. Opcode: NotFound P1 P2 P3 P4 * ** Synopsis: key=r[P3@P4] *** If P4==0 then register P3 holds a blob constructed by MakeRecord. If ** P4>0 then register P3 is the first of P4 registers that form an unpacked ** record. *** Cursor P1 is on an index btree. If the record identified by P3 and P4 ** is not the prefix of any entry in P1 then a jump is made to P2. If P1 ** does contain an entry whose prefix matches the P3/P4 record then control ** falls through to the next instruction and P1 is left pointing at the ** matching entry. *** This operation leaves the cursor in a state where it cannot be ** advanced in either direction. In other words, the Next and Prev ** opcodes do not work after this operation. *** See also: Found, NotExists, NoConflict
62912. ** Propagate all EP_Propagate flags from the Expr.x.pList into ** Expr.flags.
62913. Index of column in parent tbl
62914. Do the comparison
62915. State must be valid.
62916. Number of elements on left side vector
62917. ** Streaming versions of changegroup_add().
62918. ** xFilter - Initialize a cursor to point at the start of its data.
62919. List to be merged
62920. **comment:** ** The default SQLite sqlite3_vfs implementations do not allocate ** memory (actually, os_unix.c allocates a small amount of memory ** from within OsOpen(), but some third-party implementations may. ** So we test the effects of a malloc() failing and the sqlite3OsXXX() ** function returning SQLITE_IOERR_NOMEM using the DO_OS_MALLOC_TEST macro. *** The following functions are instrumented for malloc() failure ** testing: *** sqlite3OsRead() ** sqlite3OsWrite() ** sqlite3OsSync() ** sqlite3OsFileSize() ** sqlite3OsLock() ** sqlite3OsCheckReservedLock() ** sqlite3OsFileControl() ** sqlite3OsShmMap() ** sqlite3OsOpen() ** sqlite3OsDelete() ** sqlite3OsAccess() ** sqlite3OsFullPathname() **
label: code-design
62921. Create the content table
62922. Match these columns only
62923. Next unread data byte
62924. indexed_opt ::=
62925. Take a copy of the sample. Add two 0x00 bytes the end of the buffer. ** This is in case the sample record is corrupted. In that case, the ** sqlite3VdbeRecordCompare() may read up to two varints past the ** end of the allocated buffer before it realizes it is dealing with ** a corrupt record. Adding the two 0x00 bytes prevents this from causing ** a buffer overread.
62926. New value
62927. Case 1: expr IN (SELECT ...) *** Generate code to write the results of the select into the temporary ** table allocated and opened above.
62928. Result of the comparison of pIn1 against pIn3
62929. Parsing context used to create this Vdbe
62930. 63
62931. Integer primary key index
62932. If eMode==BTALLOC_EXACT and a query of the pointer-map ** shows that the page 'nearby' is somewhere on the free-list, then ** the entire-list will be searched for that page.
62933. Doclist to iterate through
62934. Content is irrelevant for ** 1. the sizeof() function, ** 2. the length(X) function if X is a blob, and ** 3. if the content length is zero. ** So we might as well use bogus content rather than reading ** content from disk. *** Although sqlite3VdbeSerialGet() may read at most 8 bytes from the ** buffer passed to it, debugging function VdbeMemPrettyPrint() may ** read up to 16. So 16 bytes of bogus content is supplied.
62935. Bytes to allocate for each journal fd
62936. Override onError to this if not OE_Default
62937. Adjust pTemplate cost upward so that it is costlier than p since ** pTemplate is a proper subset of p
62938. autoinc ::= AUTOINCR
62939. The database of pDest
62940. See if the checksum matches the master journal name
62941. ** A single cell from a node, serialized
62942. ** This is called to implement the special "VALUES('merge', \$nMerge)" ** INSERT command.
62943. The page to move to.
62944. Fill in pC->aType[i] and aOffset[i] values through the p2-th field.

62945. Grow the Pager.aSavepoint array using realloc(). Return SQLITE_NOMEM ** if the allocation fails. Otherwise, zero the new portion in case a ** malloc failure occurs while populating it in the for(...) loop below.

62946. If this is a REPLACE, first remove the current entry (if any)

62947. OUT: Prepared statement handle

62948. Information about the WHERE clause

62949. ** CAPI3REF: Delete A Changegroup Object

62950. ** Return the total number of key-value operations (inserts, deletes or ** updates) that have been performed on the target database since the ** current RBU update was started.

62951. ** Invoke the busy handler for a btree.

62952. **comment:** Update the page-size to match the value read from the journal. ** Use a testcase() macro to make sure that malloc failure within ** PagerSetPagesize() is tested.

label: code-design

62953. Mode parameter for allocateBtreePage()

62954. ... of the TK_xx values...

62955. Generate code that deals with a rowid collision

62956. Chunk of wal-index containing header

62957. NOT => nothing

62958. Free all allocated memory and reset the JsonString object back to its ** initial state.

62959. Iterator to advance

62960. 1000

62961. For comparing with previous entry

62962. languageid

62963. **comment:** ** These no-op macros are used in front of interfaces to mark those ** interfaces as either deprecated or experimental. New applications ** should not use deprecated interfaces - they are supported for backwards ** compatibility only. Application writers should be aware that ** experimental interfaces are subject to change in point releases. *** These macros used to resolve to various kinds of compiler magic that ** would generate warning messages when they were used. But that ** compiler magic ended up generating such a flurry of bug reports ** that we have taken it all out and gone back to using simple ** noop macros.

label: code-design

62964. Open conch file

62965. Configured cache size

62966. Value is a RowSet object

62967. **comment:** ** Adjust the WhereLoop.nOut value downward to account for terms of the ** WHERE clause that reference the loop but which are not used by an ** index. * ** For every WHERE clause term that is not used by the index ** and which has a truth probability assigned by one of the likelihood(), ** likely(), or unlikely() SQL functions, reduce the estimated number ** of output rows by the probability specified. *** TUNING: For every WHERE clause term that is not used by the index ** and which does not have an assigned truth probability, heuristics ** described below are used to try to estimate the truth probability. ** TODO --> Perhaps this is something that could be improved by better ** table statistics. *** Heuristic 1: Estimate the truth probability as 93.75%. The 93.75% ** value corresponds to -1 in LogEst notation, so this means decrement ** the WhereLoop.nOut field for every such WHERE clause term. *** Heuristic 2: If there exists one or more WHERE clause terms of the ** form "x==EXPR" and EXPR is not a constant 0 or 1, then make sure the ** final output row estimate is no greater than 1/4 of the total number ** of rows in the table. In other words, assume that x==EXPR will filter ** out at least 3 out of 4 rows. If EXPR is -1 or 0 or 1, then maybe the ** "x" column is boolean or else -1 or 0 or 1 is a common default value ** on the "x" column and so in that case only cap the output row estimate ** at 1/2 instead of 1/4.

label: code-design

62968. ** Called when we are unable to satisfy an allocation of nBytes.

62969. **comment:** Unmap any pages of the existing mapping that cannot be reused.

label: code-design

62970. 155

62971. First byte of buffer to write

62972. ** The actual function that does the work of creating a new module. ** This function implements the sqlite3_create_module() and ** sqlite3_create_module_v2() interfaces.

62973. "content_rowid=" option value

62974. True if field width constant starts with zero

62975. make our hash too "full".

62976. 0=entries. 1=leaf node. 2+ for higher

62977. Size before spilling occurs

62978. Which db file is being initialized

62979. ** Free an Fts3MultiSegReader allocated by fts3TermSegReaderCursor().

62980. synopsis: r[P3]=rowset(P1)

62981. All above except SYNCHRONOUS

62982. Language id of row being written

62983. ** This function is called by the xUpdate() method for an INSERT operation. ** The apVal parameter is passed a copy of the apVal argument passed by ** SQLite to the xUpdate() method. i.e: *** apVal[0] Not used for INSERT. ** apVal[1] rowid ** apVal[2] Left-most user-defined column ** ... ** apVal[p->nColumn+1] Right-most user-defined column ** apVal[p->nColumn+2] Hidden column with same name as table ** apVal[p->nColumn+3] Hidden "docid" column (alias for rowid) ** apVal[p->nColumn+4] Hidden languageid column

62984. ** The xDisconnect() virtual table method.

62985. Counter used to generate aColCache[],lru values

62986. Value extracted from pLower

62987. Keep cursor valid

62988. ** An instance of the following structure encodes all information that can ** be gleaned from the CREATE VIRTUAL TABLE statement. *** And all information loaded from the %_config table. *** nAutomerge: ** The minimum number of segments that an auto-merge operation should ** attempt to merge together. A value of 1 sets the object to use the ** compile time default. Zero disables auto-merge altogether. *** zContent: ** ** zContentRowid: ** The value of the content_rowid= option, if one was specified. Or ** the string "rowid" otherwise. This text is not quoted - if it is ** used as part of an SQL statement it needs to be quoted appropriately. *** zContentExprlist: ** ** pzErrMsg: ** This exists in order to allow the fts5_index.c module to return a ** decent error message if it encounters a file-format version it does ** not understand. *** bColumnsize: ** True if the %_docszie table is created. *** bPrefixIndex: ** This is only used for debugging. If set to false, any prefix indexes ** are ignored. This value is configured using: *** INSERT INTO tbl(tbl, rank) VALUES('prefix-index', \$bPrefixIndex); **

62989. Allocate nArg registers to martial the arguments to VUpdate. Then ** create and open the ephemeral table in which the records created from ** these arguments will be temporarily stored.

62990. ** This routine is the core allocator for Expr nodes. *** Construct a new expression node and return a pointer to it. Memory ** for this node and for the pToken argument is a single allocation ** obtained from sqlite3DbMalloc(). The calling function ** is responsible for making sure the node eventually gets freed. *** If dequote is true, then the token (if it exists) is dequoted. ** If dequote is false, no dequoting is performed. The deQuote ** parameter is ignored if pToken is NULL or if the token does not ** appear to be quoted. If the quotes were of the form "... (double-quotes) ** then the EP_DblQuoted flag is set on the expression node. *** Special case: If op==TK_INTEGER and pToken points to a string that ** can be translated into a 32-bit integer, then the token is not ** stored in u.zToken. Instead, the integer values is written ** into u.iValue and the EP_IntValue flag is set. No extra storage ** is allocated to hold the integer text and the dequote flag is ignored.

62991. Cursor used to iterate through aLeft

62992. ** Launch a background thread to run xTask(pIn).

62993. ** Return the N-th AND-connected subterm of pTerm. Or if pTerm is not ** a conjunction, then return just pTerm when N==0. If N is exceeds ** the number of available subterms, return NULL.

62994. ** CAPI3REF: Opening A New Database Connection ** CONSTRUCTOR: sqlite3 *** ^These routines open an SQLite database file as specified by the ** filename argument. ^The filename argument is interpreted as UTF-8 for ** sqlite3_open() and sqlite3_open_v2() and as UTF-16 in the native byte ** order for sqlite3_open16(). ^A [database connection] handle is usually ** returned in *ppDb, even if an error occurs. The only exception is that ** if SQLite is unable to allocate memory to hold the [sqlite3] object, ** a NULL will be written into *ppDb instead of pointer to the [sqlite3] ** object.)^ ^If the database is opened (and/or created) successfully, then ** [SQLITE_OK] is returned. Otherwise an [error code] is returned.)^ ^The ** [sqlite3_errmsg0] or [sqlite3_errmsg16()] routines can be used to obtain ** an English language description of the error following a failure of any ** of the sqlite3_open() routines. *** ^The default encoding will be UTF-8 for databases created using ** sqlite3_open() or sqlite3_open_v2(). ^The default encoding for databases ** created using sqlite3_open16() will be UTF-16 in the native byte order. *** Whether or not an error occurs when it is opened, resources ** associated with the [database connection] handle should be released by ** passing it to [sqlite3_close()] when it is no longer required. *** The sqlite3_open_v2() interface works like sqlite3_open() ** except that it accepts two additional parameters for additional control ** over the new database connection. ^The flags parameter to ** sqlite3_open_v2() can take one of ** the following three values, optionally combined with the ** [SQLITE_OPEN_NOMUTEX], [SQLITE_OPEN_FULLMUTEX], [SQLITE_OPEN_SHARED_CACHE], ** [SQLITE_OPEN_PRIVATECACHE], and/or [SQLITE_OPEN_URI] flags:)^ *** <dl> ** ^</dl>

** ^(<dt>[SQLITE_OPEN_READONLY]</dt> ** <dd>The database is opened in read-only mode. If the database does not ** already exist, an error is returned.</dd>) ^ **

** ^(<dt>[SQLITE_OPEN_READWRITE]</dt> ** <dd>The database is opened for reading and writing if possible, or reading ** only if the file is write protected by the operating system. In either ** case the database must already exist, otherwise an error is returned.</dd>) ^ ** ^(<dt>

[SQLITE_OPEN_READWRITE] | [SQLITE_OPEN_CREATE])</dt> ** <dd>The database is opened for reading and writing, and is created if ** it does not already exist. This is the behavior that is always used for ** sqlite3_open() and sqlite3_open16().</dd>) ^ ** </dl> ** ^ If the 3rd parameter to sqlite3_open_v2() is not one of the ** combinations shown above optionally combined with other ** [SQLITE_OPEN_READONLY] | [SQLITE_OPEN_* bits] ** then the behavior is undefined. *** ^If the [SQLITE_OPEN_NOMUTEX] flag is set, then the database connection ** opens in the multi-thread [threading mode] as long as the single-thread ** mode has not been set at compile-time or start-time. ^If the ** [SQLITE_OPEN_FULLMUTEX] flag is set then the database connection opens ** in the serialized [threading mode] unless single-thread was ** previously selected at compile-time or start-time. ** ^The [SQLITE_OPEN_SHARED_CACHE] flag causes the database connection to be ** eligible to use [shared cache mode], regardless of whether or not shared ** cache is enabled using [sqlite3_enable_shared_cache()]. ^The ** [SQLITE_OPEN_PRIVATECACHE] flag causes the database connection to not ** participate in [shared cache mode] even if it is enabled. *** ^The fourth parameter to sqlite3_open_v2() is the name of the ** [sqlite3_vfs] object that defines the operating system interface that ** the new database connection should use. ^If the fourth parameter is ** a NULL pointer then the default [sqlite3_vfs] object is used. *** ^If the filename is ".memory.", then a private, temporary in-memory database ** is created for the connection. ^This in-memory database will vanish when ** the database connection is closed. Future versions of SQLite might ** make use of additional special filenames that begin with the ":" character. ** It is recommended that when a database filename actually does begin with a ":" character you should prefix the filename with a pathname such as ** "./" to avoid ambiguity. *** ^ If the filename is an empty string, then a private, temporary ** on-disk database will be created. ^This private database will be ** automatically deleted as soon as the database connection is closed. *** ^[[URI filenames in sqlite3_open()]] <h3>URI Filenames</h3> *** ^ If [URI filename] interpretation is enabled, and the filename argument ** begins with "file:", then the filename is interpreted as a URI. ^URI ** filename interpretation is enabled if the [SQLITE_OPEN_URI] flag is ** set in the fourth argument to sqlite3_open_v2(), or if it has ** been enabled globally using the [SQLITE_CONFIG_URI] option with the ** [sqlite3_config()] method or by the [SQLITE_USE_URI] compile-time option. ** As of SQLite version 3.7.7, URI filename interpretation is turned off ** by default, but future releases of SQLite might enable URI filename ** interpretation by default. See "[URI filenames]" for additional ** information. *** ^ URI filenames are parsed according to RFC 3986. ^If the URI contains an ** authority, then it must be either an empty string or the string ** "localhost". ^If the authority is not an empty string or "localhost", an ** error is returned to the caller. ^The fragment component of a URI, if ** present, is ignored. *** ^ SQLite uses the path component of the URI as the name of the disk file ** which contains the database. ^If the path begins with a '/' character, ** then it is interpreted as an absolute path. ^If the path does not begin ** with a '/' (meaning that the authority section is omitted from the URI) ** then the path is interpreted as a relative path. ** ^ (On windows, the first component of an absolute path ** is a drive specification (e.g. "C:").) ^ ** ^[[core URI query parameters]] ** The query component of a URI may contain parameters that are interpreted ** either by SQLite itself, or by a [VFS] custom VFS implementation. ** SQLite and its built-in [VFSes] interpret the ** following query parameters: *** ** vfs: ^The "vfs" parameter may be used to specify the name of ** a VFS object that provides the operating system interface that should ** be used to access the database file on disk. ^If this option is set to ** an empty string the default VFS object is used. ^Specifying an unknown ** VFS is an error. ^If sqlite3_open_v2() is used and the vfs option is ** present, then the VFS specified by the option takes precedence over ** the value passed as the fourth parameter to sqlite3_open_v2(). *** mode: ^The mode parameter may be set to either "ro", "rw", ** "rwc", or "memory". Attempting to set it to any other value is ** an error). ^ ** ^ If "ro" is specified, then the database is opened for read-only ** access, just as if the [SQLITE_OPEN_READONLY] flag had been set in the ** third argument to sqlite3_open_v2(). ^If the mode option is set to ** "rw", then the database is opened for read-write (but not create) ** access, as if SQLITE_OPEN_READWRITE (but not SQLITE_OPEN_CREATE) had ** been set. ^Value "rwc" is equivalent to setting both ** SQLITE_OPEN_READWRITE and SQLITE_OPEN_CREATE. ^If the mode option is ** set to "memory" then a pure [in-memory database] that never reads ** or writes from disk is used. ^It is an error to specify a value for ** the mode parameter that is less restrictive than that specified by ** the flags passed in the third parameter to sqlite3_open_v2(). *** cache: ^The cache parameter may be set to either "shared" or ** "private". ^Setting it to "shared" is equivalent to setting the ** SQLITE_OPEN_SHARED_CACHE bit in the flags argument passed to ** sqlite3_open_v2(). ^Setting the cache parameter to "private" is ** equivalent to setting the SQLITE_OPEN_PRIVATECACHE bit. ** ^If sqlite3_open_v2() is used and the "cache" parameter is present in ** a URI filename, its value overrides any behavior requested by setting ** SQLITE_OPEN_PRIVATECACHE or SQLITE_OPEN_SHARED_CACHE flag. *** psow: ^The psow parameter indicates whether or not the ** [powersafe overwrite] property does or does not apply to the ** storage media on which the database file resides. *** nolock: ^The nolock parameter is a boolean query parameter ** which if set disables file locking in rollback journal modes. This ** is useful for accessing a database on a filesystem that does not ** support locking. Caution: Database corruption might result if two ** or more processes write to the same database and any one of those ** processes uses nolock=1. *** immutable: ^The immutable parameter is a boolean query ** parameter that indicates that the database file is stored on ** read-only media. ^When immutable is set, SQLite assumes that the ** database file cannot be changed, even by a process with higher ** privilege, and so the database is opened read-only and all locking ** and change detection is disabled. Caution: Setting the immutable ** property on a database file that does in fact change can result ** in incorrect query results and/or [SQLITE_CORRUPT] errors. ** See also: [SQLITE_IOCAP_IMMUTABLE]. *** *** ^ Specifying an unknown parameter in the query component of a URI is not an ** error. Future versions of SQLite might understand additional query ** parameters. See "[query parameters with special meaning to SQLite]" for ** additional information. ** ^[[URI filename examples]] <h3>URI filename examples</h3> *** <table border="1" align="center" cellpadding="5" ** <tr><th> URI filenames <th> Results ** <tr><td> file:/data.db <td> Open the file "data.db" in the current directory. ** <tr><td> file:/home/fred/data.db
 ** file://home/fred/data.db
 ** file://localhost/home/fred/data.db
 <td> ** Open the database file "/home/fred/data.db". ** <tr><td> file://darkstar/home/fred/data.db <td> ** An error. "darkstar" is not a recognized authority. ** <tr><td style="white-space:nowrap"> ** file:///C:/Documents%20and%20Settings/fred/Desktop/data.db ** <td> Windows only: Open the file "data.db" on fred's desktop on drive ** C:. Note that the %20 escaping in this example is not strictly ** necessary - space characters can be used literally ** in URI filenames. ** <tr><td> file:/data.db?mode=ro&cache=private <td> ** Open file "data.db" in the current directory for read-only access. ** Regardless of whether or not shared-cache mode is enabled by ** default, use a private cache. ** <tr><td> file:/home/fred/data.db?vfs=unix-dotfile <td> ** Open file "/home/fred/data.db". Use the special VFS "unix-dotfile" ** that uses dot-files in place of posix advisory locking. ** <tr><td> file:/data.db?mode=readonly <td> ** An error. "readonly" is not a valid option for the "mode" parameter. ** </table> *** ^ URI hexadecimal escape sequences (%HH) are supported within the path and ** query components of a URI. A hexadecimal escape sequence consists of a ** percent sign - "%" - followed by exactly two hexadecimal digits ** specifying an octet value. ^Before the path or query components of a ** URI filename are interpreted, they are encoded using UTF-8 and all ** hexadecimal escape sequences replaced by a single byte containing the ** corresponding octet. If this process generates an invalid UTF-8 encoding, ** the results are undefined. *** Note to Windows users: The encoding used for the filename argument ** of sqlite3_open() and sqlite3_open_v2() must be UTF-8, not whatever ** codepage is currently defined. Filenames containing international ** characters must be converted to UTF-8 prior to passing them into ** sqlite3_open() or sqlite3_open_v2(). *** Note to Windows Runtime users: The temporary directory must be set ** prior to calling sqlite3_open() or sqlite3_open_v2(). Otherwise, various ** features that require the use of temporary files may fail. *** See also: [sqlite3_temp_directory]

62995. 124

62996. Number of hidden columns if TABLE is virtual

62997. Next available cursor

62998. **comment:** LIMIT value. NULL means not used

label: code-design

62999. Index of statement to evaluate

63000. ** This is the equivalent of fts5MergePrefixLists() for detail=none mode. ** In this case the buffers consist of a delta-encoded list of rowids only.

63001. SrcList to be returned

63002. Length of the delta

63003. Sorter key to copy into pOut

63004. Reset high-water mark if true
63005. OUT: Changeset iterator handle
63006. Schema that contains this table
63007. # include "sqlite3ext.h"
63008. DISTINCT
63009. **comment:** Mapping between variable names and numbers
 label: code-design
63010. Opcode: DropTable P1 * * P4 * * * Remove the internal (in-memory) data structures that describe ** the table named P4 in database P1. This is called after a table ** is dropped from disk (using the Destroy opcode) in order to keep ** the internal representation of the ** schema consistent with what is on disk.
63011. expr ::= expr likeop expr ESCAPE expr
63012. Pointer to input string
63013. if we are not initializing, ** build the sqlite_master entry
63014. Temp file
63015. FTS3 cursor object
63016. String to be tokenized
63017. Function Return Code
63018. ** Do a memory allocation with statistics and alarms. Assume the ** lock is already held.
63019. Required file size
63020. If the pParse->declareVtab flag is set, do not delete any table ** structure built up in pParse->pNewTable. The calling code (see vtab.c) ** will take responsibility for freeing the Table structure.
63021. TRIGGER
63022. Callback function to invoke
63023. SQLITE_OS_WINRT
63024. The unqualified db name
63025. Buffer to load WAL header into
63026. Fts5 index object
63027. ** Argument pIdxInfo is already populated with all constraints that may ** be used by the virtual table identified by pBuilder->pNew->iTab. This ** function marks a subset of those constraints usable, invokes the ** xBestIndex method and adds the returned plan to pBuilder. ** * A constraint is marked usable if: ** ** * Argument mUsable indicates that its prerequisites are available, and ** ** * It is not one of the operators specified in the mExclude mask passed ** as the fourth argument (which in practice is either WO_IN or 0). ** ** * Argument mPreq is a mask of tables that must be scanned before the ** virtual table in question. These are added to the plans prerequisites ** before it is added to pBuilder. ** ** * Output parameter *pbIn is set to true if the plan added to pBuilder ** uses one or more WO_IN terms, or false otherwise.
63028. If this is the magic sqlite_sequence table used by autoincrement, ** then record a pointer to this table in the main database structure ** so that INSERT can find the table easily.
63029. This function does not work with detail=none databases.
63030. ***** Start of highlight() implementation.
63031. 50
63032. Current term read from page
63033. ePragTyp:
63034. Copy data from the log to the database file.
63035. Assign a bit from the bitmask to every term in the FROM clause. ** * The N-th term of the FROM clause is assigned a bitmask of 1<<N. ** * The rule of the previous sentence ensures that if X is the bitmask for ** a table T, then X-1 is the bitmask for all other tables to the left of T. ** Knowing the bitmask for all tables to the left of a left join is ** important. Ticket #3015. ** * Note that bitmasks are created for all pTabList->nSrc tables in ** pTabList, not just the first nTabList tables. nTabList is normally ** equal to pTabList->nSrc but might be shortened to 1 if the ** WHERE_OR_SUBCLAUSE flag is set.
63036. Buffer to gobble string/bareword from
63037. 86
63038. P4 is a pointer to a SubProgram structure
63039. VM being reprepared
63040. The P4 operand
63041. Doclist data
63042. xFetch, xUnfetch
63043. ** When this function is called, the database file has been completely ** updated to reflect the changes made by the current transaction and ** synced to disk. The journal file still exists in the file-system ** though, and if a failure occurs at this point it will eventually ** be used as a hot-journal and the current transaction rolled back. ** * This function finalizes the journal file, either by deleting, ** truncating or partially zeroing it, so that it cannot be used ** for hot-journal rollback. Once this is done the transaction is ** irrevocably committed. ** * If an error occurs, an IO error code is returned and the pager ** moves into the error state. Otherwise, SQLITE_OK is returned.
63044. **comment:** Takes a fully configured proxy locking-style unix file and switches ** the local lock file path
 label: code-design
63045. Name of the container - used for error messages
63046. Resume scanning at this->pWC->a[this->k]
63047. Match **
63048. True to reset page prior to first page rollback
63049. ** Return a bitmask where 1s indicate that the corresponding column of ** the table is used by an index. Only the first 63 columns are considered.
63050. Node of the TARGET in pParse
63051. ** Obtain a snapshot handle for the snapshot of database zDb currently ** being read by handle db.
63052. same as TK_LT, jump, in1, in3
63053. Extra information if (eOperator & WO_OR)!=0
63054. The cursor
63055. aKWCode[i] is the parser symbol code for the i-th keyword
63056. ** Recursively delete an expression tree.
63057. If the OS does not have posix_fallocate(), fake it. Write a ** single byte to the last byte in each block that falls entirely ** within the extended region. Then, if required, a single byte ** at offset (nSize-1), to set the size of the file correctly. ** This is a similar technique to that used by glibc on systems ** that do not have a real fallocate() call.
63058. Recently parsed phrase
63059. ** Make an shallow copy of pFrom into pTo. Prior contents of ** pTo are freed. The pFrom->z field is not duplicated. If ** pFrom->z is used, then pTo->z points to the same thing as pFrom->z ** and flags gets srcType (either MEM_Ephem or MEM_Static).
63060. ** Acquire a page if it is already in the in-memory cache. Do ** not read the page from disk. Return a pointer to the page, ** or 0 if the page is not in cache. ** * See also sqlite3PagerGet(). The difference between this routine ** and sqlite3PagerGet() is that _get() will go to the disk and read ** in the page if the page is not already in cache. This routine ** returns NULL if the page is not in cache or if a disk I/O error ** has ever happened.
63061. ** This function is not available on Windows CE or WinRT.
63062. ***** Begin file fts3_expr.c *****
63063. The btree to set the safety level on
63064. xDestroy
63065. 49
63066. ** * If *pRc is not SQLITE_OK when this function is called, it is a no-op. ** Otherwise, fts3EvalPhraseStart() is called on all phrases within the ** expression. Also the Fts3Expr.bDeferred variable is set to true for any ** expressions for which all descendant tokens are deferred. ** * If parameter bOptOk is zero, then it is guaranteed that the ** Fts3Phrase.doclist.aAll/nAll variables contain the entire doclist for ** each phrase in the expression (subject to deferred token

processing). ** Or, if bOptOk is non-zero, then one or more tokens within the expression ** may be loaded incrementally, meaning doclist.aAll/nAll is not available. *** If an error occurs within this function, *pRc is set to an SQLite error ** code before returning.

63067. PRAGMA main.index_xinfo = \$zIdx

63068. Read (and consume) the next character from the input pattern.

63069. Buffer of at least nBuf bytes

63070. ** Retrieve information about the current candidate snippet of snippet ** iterator pIter.

63071. Information about AUTOINCREMENT counters

63072. String or BLOB value

63073. Variables above this point are populated when the expression is ** parsed (by code in fts3_expr.c). Below this point the variables are ** used when evaluating the expression.

63074. Current index

63075. **comment:** ** NOTE: Windows CE is handled differently here due its lack of the Win32 ** API LockFileEx.

label: test

63076. ** pCur points at an index entry created using the OP_MakeRecord opcode. ** Read the rowid (the last field in the record) and store it in *rowid. ** Return SQLITE_OK if everything works, or an error code otherwise. *** pCur might be pointing to text obtained from a corrupt database file. ** So the content cannot be trusted. Do appropriate checks on the content.

63077. ** Size of struct Mem not including the Mem.zMalloc member or anything that ** follows.

63078. Size of column in tokens

63079. ** Assuming the current table columns are "a", "b" and "c", and the zObj ** parameter is passed "old", return a string of the form: ** ** "old.a, old.b, old.b" ** ** With the column names escaped. *** For tables with implicit rowids - RBU_PK_EXTERNAL and RBU_PK_NONE, append ** the text ", old._rowid_" to the returned value.

63080. No-op

63081. Database in which to run SQL

63082. The LIMIT expression. NULL if there is no limit

63083. The final Fts3Expr data structure, including the Fts3Phrase, ** Fts3PhraseToken structures token buffers are all stored as a single ** allocation so that the expression can be freed with a single call to ** sqlite3_free(). Setting this up requires a two pass approach. *** The first pass, in the block below, uses a tokenizer cursor to iterate ** through the tokens in the expression. This pass uses fts3ReallocOrFree() ** to assemble data in two dynamic buffers: ** ** Buffer p: Points to the Fts3Expr structure, followed by the Fts3Phrase ** structure, followed by the array of Fts3PhraseToken ** structures. This pass only populates the Fts3PhraseToken array. *** Buffer zTemp: Contains copies of all tokens. *** The second pass, in the block that begins "if(rc==SQLITE_DONE)" below, ** appends buffer zTemp to buffer p, and fills in the Fts3Expr and Fts3Phrase ** structures.

63084. Array of tables to search

63085. synopsis: r[P2@P3]=r[P1@P3]

63086. Locking style specific state

63087. Left-hand Fts5SegIter

63088. ** On some systems, calls to fchown() will trigger a message in a security ** log if they come from non-root processes. So avoid calling fchown() if ** we are not running as root.

63089. True if the subquery uses aggregate functions

63090. DEFERRABLE

63091. Invoke the token callback

63092. Bytes of temp space

63093. If this is a non-covering index scan, add in the cost of ** doing table lookups. The cost will be 3x the number of ** lookups. Take into account WHERE clause terms that can be ** satisfied using just the index, and that do not require a ** table lookup.

63094. RBU handle

63095. If the opcode is TK_TRIGGER, then the expression is a reference ** to a column in the new.* or old.* pseudo-tables available to ** trigger programs. In this case Expr.iTable is set to 1 for the ** new.* pseudo-table, or 0 for the old.* pseudo-table. Expr.iColumn ** is set to the column of the pseudo-table to read, or to -1 to ** read the rowid field.

63096. (311) nmnum ::= ON

63097. If pNew is still NULL, try to create an entirely new mapping.

63098. Normal (non-error) return.

63099. 267

63100. Display all terms of the WHERE clause

63101. CURTYPE_BTREE. Btree cursor

63102. Halt the sub-program. Return control to the parent frame.

63103. If STRERROR_R_CHAR_P (set by autoconf scripts) or __USE_GNU is defined, ** assume that the system provides the GNU version of strerror_r() that ** returns a pointer to a buffer containing the error message. That pointer ** may point to aErr[], or it may point to some static storage somewhere. ** Otherwise, assume that the system provides the POSIX version of ** strerror_r(), which always writes an error message into aErr[]. *** If the code incorrectly assumes that it is the POSIX version that is ** available, the error message will often be an empty string. Not a ** huge problem. Incorrectly concluding that the GNU version is available ** could lead to a segfault though.

63104. same as TK_Rem, in1, in2, out3

63105. Make sure the sort order is compatible in an ORDER BY clause. ** Sort order is irrelevant for a GROUP BY clause.

63106. **comment:** ** A phrase. One or more terms that must appear in a contiguous sequence ** within a document for it to match.

label: documentation

63107. Rtree node cache

63108. Tokenizer cursor open on zDoc/nDoc

63109. For use by application VFS

63110. Usually the path zFilename should not be a relative pathname. The ** exception is when opening the proxy "conch" file in builds that ** include the special Apple locking styles.

63111. Pointer to one journal within MJ file

63112. cmd ::= select

63113. ***** Include os_setup.h in the middle of os.h *****

63114. Index of constraint to use

63115. Query for existance of table only

63116. 68..6f hijklmno

63117. ** Provide a default value for SQLITE_TEMP_STORE in case it is not specified ** on the command-line

63118. 120

63119. rand_s() is not available with MinGW

63120. Fifth parameter is an unsigned 16-bit integer

63121. Instruction number of OP_Function

63122. **comment:** ** A list of expressions. Each expression may optionally have a ** name. An expr/name combination can be used in several ways, such ** as the list of "expr AS ID" fields following a "SELECT" or in the ** list of "ID = expr" items in an UPDATE. A list of expressions can ** also be used as the argument to a function, in which case the a.zName ** field is not used. *** By default the Expr.zSpan field holds a human-readable description of ** the expression that is used in the generation of error messages and ** column labels. In this case, Expr.zSpan is typically the text of a ** column expression as it exists in a SELECT statement. However, if ** the bSpanIsTab flag is set, then zSpan is overloaded to mean the name ** of the result column in the form: DATABASE.TABLE.COLUMN. This later ** form is used for name resolution with nested FROM clauses.

label: code-design

63123. End of file

63124. Commit by truncating journal

63125. Associated FTS5 index

63126. Content size or sub-node count
63127. IMP: R-03371-37637
63128. Convert the connection into a zombie and then close it.
63129. ** This lookup table is used to help decode the first byte of ** a multi-byte UTF8 character.
63130. Number of arguments in subprograms
63131. Size of subclassed sqlite3_file
63132. Handle for accessing the blob returned here
63133. Usable size of the page
63134. Source database handle
63135. OUT: Number of bytes in token
63136. for() loop iterator variable
63137. sqlite3.aDb[] index of db to checkpoint
63138. True for patchset, false for changeset
63139. **comment:** ** The packing computed by the previous block is biased toward the siblings ** on the left side (siblings with smaller keys). The left siblings are ** always nearly full, while the right-most sibling might be nearly empty. ** The next block of code attempts to adjust the packing of siblings to ** get a better balance. ** ** This adjustment is more than an optimization. The packing above might ** be so out of balance as to be illegal. For example, the right-most ** sibling might be completely empty. This adjustment is not optional.
label: code-design
63140. Thread context used to create new PMA
63141. Non-zero if connection is in exclusive mode
63142. ** Argument aCol points to an array of integers containing one entry for ** each table column. This function reads the %_docszie record for the ** specified rowid and populates aCol[] with the results. ** ** An SQLite error code is returned if an error occurs, or SQLITE_OK ** otherwise.
63143. Input "x" is a sequence of unsigned characters that represent a ** big-endian integer. Return the equivalent native integer
63144. IN/OUT: Right-hand buffer pointer
63145. ** json_remove(JSON, PATH, ...) ** ** Remove the named elements from JSON and return the result. malformed ** JSON or PATH arguments result in an error.
63146. ** The first time the bm25() function is called for a query, an instance ** of the following structure is allocated and populated.
63147. First argument to pass to xTableFilter
63148. Verify that constants are precomputed correctly
63149. ** Take an EXCLUSIVE lock on the database file.
63150. Current entry in aRowidOffset[]
63151. pPage
63152. ** CAPI3REF: Checkpoint a database ** METHOD: sqlite3 *** ** ^{(The sqlite3_wal_checkpoint(D,X) is equivalent to ** [sqlite3_wal_checkpoint_v2](D,X, [SQLITE_CHECKPOINT_PASSIVE],0,0).)^ ** **} In brief, sqlite3_wal_checkpoint(D,X) causes the content in the ** [write-ahead log] for database X on [database connection] D to be ** transferred into the database file and for the write-ahead log to ** be reset. See the [checkpointing] documentation for addition ** information. ** ** This interface used to be the only way to cause a checkpoint to ** occur. But then the newer and more powerful [sqlite3_wal_checkpoint_v2()] ** interface was added. This interface is retained for backwards ** compatibility and as a convenience for applications that need to manually ** start a callback but which do not need the full power (and corresponding ** complication) of [sqlite3_wal_checkpoint_v2()].
63153. Mask of NEW.* columns accessed by BEFORE triggers
63154. Write error message here
63155. If the number of pages in the database is not available from the ** WAL sub-system, determine the page count based on the size of ** the database file. If the size of the database file is not an ** integer multiple of the page-size, round up the result.
63156. pPager->stmtJSize = 0;
63157. Also frees aRegIdx[] and aToOpen[]
63158. Values between 16384 and 2097151
63159. Always the first entry
63160. **comment:** Calculate the IDF (Inverse Document Frequency) for phrase i. ** This is done using the standard BM25 formula as found on wikipedia: ** ** $IDF = \log((N - nHit + 0.5) / (nHit + 0.5))$ ** ** where "N" is the total number of documents in the set and nHit ** is the number that contain at least one instance of the phrase ** under consideration. ** ** The problem with this is that if ($N < 2*nHit$), the IDF is ** negative. Which is undesirable. So the mimimum allowable IDF is ** (1e-6) - roughly the same as a term that appears in just over ** half of set of 5,000,000 documents.
label: documentation
63161. **comment:** If malloc() failed during an encoding conversion within an ** sqlite3_column_XXX API, then set the return code of the statement to ** SQLITE_NOMEM. The next call to _step() (if any) will return SQLITE_ERROR ** and _finalize() will return NOMEM.
label: code-design
63162. 133
63163. Descend to the child node of the cell that the cursor currently ** points at. This is the right-child if (iIdx==pPage->nCell).
63164. 2
63165. A table in the schema
63166. Array of unquoted target column names
63167. Create the imposter table used to write to this index.
63168. NOTREACHED
63169. Add this WhereLoop to the end of pPath->aLoop[]
63170. Prefix query
63171. Take data to be written from here
63172. EVIDENCE-OF: R-05119-02637 The 4-byte big-endian integer at offset 36 ** stores stores the total number of pages on the freelist.
63173. Bitmask of schema verified databases
63174. The tokenizer always gives us a token
63175. Obtain a reference to the root node to initialize Rtree.iDepth
63176. Skip backwards past any 0x00 varints.
63177. In this mode, write each query result to the key of the temporary ** table iParm.
63178. Index of old.* value to retrieve
63179. Special case: No FROM clause
63180. Write the sorted results here
63181. Increment the reference count of an existing page
63182. limit_opt ::= LIMIT expr
63183. 0x08
63184. The complex case - There is a multi-file write-transaction active. ** This requires a master journal file to ensure the transaction is ** committed atomically.
63185. Statements to read/write/delete a record from xxx_rowid
63186. if HAVE_ISNAN
63187. Size of the original file in pages
63188. Rowid components
63189. ** Allocate and zero memory.
63190. Flags to send to sqlite3PagerGet()
63191. ***** Begin file vtab.c *****
63192. ** Return the index in pList of the identifier named zId. Return -1 ** if not found.
63193. If a SHARED lock is requested, and some thread using this PID already ** has a SHARED or RESERVED lock, then increment reference counts and ** return SQLITE_OK.
63194. VERIFY: F13021
63195. SQLITE_OS_WINRT

63196. **" or "%"

63197. ** Argument pIn points to a character that is part of a nul-terminated ** string. Return a pointer to the first character following *pIn in ** the string that is not a "bareword" character.

63198. ** pArray is a pointer to an array of objects. Each object in the ** array is szEntry bytes in size. This routine uses sqlite3DbRealloc() ** to extend the array so that there is space for a new object at the end. ** ** When this function is called, *pnEntry contains the current size of ** the array (in entries - so the allocation is ((**pnEntry) * szEntry) bytes ** in total). ** ** If the realloc() is successful (i.e. if no OOM condition occurs), the ** space allocated for the new object is zeroed, *pnEntry updated to ** reflect the new size of the array and a pointer to the new allocation ** returned. *pIdx is set to the index of the new array entry in this case. ** ** Otherwise, if the realloc() fails, *pIdx is set to -1, *pnEntry remains ** unchanged and a copy of pArray returned.

63199. ** Set the buffer to contain nData/pData. If an OOM error occurs, leave an ** the error code in p. If an error has already occurred when this function ** is called, it is a no-op.

63200. Number of pages in the LRU list

63201. ** We do not trust systems to provide a working fdatasync(). Some do. ** Others do not. To be safe, we will stick with the (slightly slower) ** fsync(). If you know that your system does support fdatasync() correctly, ** then simply compile with -Dfdatasync=fdatasync or -DHAVE_FDATASYNC

63202. The first column is in this register

63203. ** Reset an StrAccum string. Reclaim all malloced memory.

63204. Value extracted from pUpper

63205. same as TK_LT, synopsis: IF r[P3]<r[P1]

63206. ** Pragma virtual table module xConnect method.

63207. Single step in trigger program

63208. 1380

63209. **comment:** Suppress a harmless compiler warning
label: code-design

63210. Jump here to break out of the inner loop

63211. Append the rowid to the output

63212. Common case: early out without every having to acquire a mutex

63213. ** Close the cursor. For additional information see the documentation ** on the xClose method of the virtual table interface.

63214. Check if it is an FTS4 special argument.

63215. Size of pHeap[]

63216. EVIDENCE-OF: R-22564-11647 The header begins with a single varint ** which determines the total number of bytes in the header. The varint ** value is the size of the header in bytes including the size varint ** itself.

63217. Mask of public flags

63218. Seconds

63219. 211

63220. **comment:** ** The argument to this macro must be of type u32. On a little-endian ** architecture, it returns the u32 value that results from interpreting ** the 4 bytes as a big-endian value. On a big-endian architecture, it ** returns the value that would be produced by interpreting the 4 bytes ** of the input value as a little-endian integer.
label: code-design

63221. Expr struct EXPR_REDUCEDSIZE bytes only

63222. 194

63223. **comment:** TODO: It would be better to have some system for reusing statement ** handles here, rather than preparing a new one for each query. But that ** is not possible as SQLite reference counts the virtual table objects. ** And since the statement required here reads from this very virtual ** table, saving it creates a circular reference. ** ** If SQLite a built-in statement cache, this wouldn't be a problem.
label: code-design

63224. If the auto-commit flag is set and this is the only active writer ** VM, then we do either a commit or rollback of the current transaction. ** ** Note: This block also runs if one of the special errors handled ** above has occurred.

63225. No checkpoint on close()|DETACH

63226. b &= (0x7f<<28)|(0x7f<<14)|(0x7f);

63227. **comment:** Unused arg for sqlite3WhereOkOnePass()
label: code-design

63228. Node chain of patch for JNODE_PATCH

63229. likeop ::= NOT LIKE_KW|MATCH

63230. What operation to perform

63231. ** Close the RBU handle.

63232. Register holding counter

63233. ***** End of wal.h *****

63234. (309) nmnum ::= plus_num (OPTIMIZED OUT)

63235. Which entry in the FROM clause

63236. values ::= values COMMA LP exprlist RP

63237. Result code from subfunctions

63238. __APPLE__ or not __APPLE__

63239. true to ignore case differences

63240. search key and new file ID

63241. Check to see if pExpr is a duplicate of another aggregate ** function that is already in the pAggInfo structure

63242. Ensure both child nodes have node numbers assigned to them by calling ** nodeWrite(). Node pRight always needs a node number, as it was created ** by nodeNew() above. But node pLeft sometimes already has a node number. ** In this case avoid the all to nodeWrite().

63243. ** Read and decode the "averages" record from the database. ** ** Parameter anSize must point to an array of size nCol, where nCol is ** the number of user defined columns in the FTS table.

63244. Number of bytes in buffer aFrame[]

63245. Size of aTask[] array

63246. ** Compute a string that describes the P4 parameter for an opcode. ** Use zTemp for any required temporary buffer space.

63247. The user-supplied minor token value. This ** is the value of the token

63248. ** Advance the iterator passed as the only argument.

63249. Smallest possible allocation in bytes

63250. **comment:** The following could be done by calling fts5SegIterLoadNPos(). But ** this block is particularly performance critical, so equivalent ** code is inlined. ** ** Later: Switched back to fts5SegIterLoadNPos() because it supports ** detail=none mode. Not ideal.
label: code-design

63251. Name of that semaphore

63252. EVIDENCE-OF: R-56861-42673 sqlite3_prepare_v3() differs from ** sqlite3_prepare_v2() only in having the extra prepFlags parameter, ** which is a bit array consisting of zero or more of the ** SQLITE_PREPARE_* flags. ** ** Proof by comparison to the implementation of sqlite3_prepare_v2() ** directly above.

63253. Jump out of the pTerm loop

63254. Search for an existing statement. If one is found, shift it to the front ** of the LRU queue and return immediately. Otherwise, leave nUp pointing ** to the number of statements currently in the cache and pUp to the ** last object in the list.

63255. ***** Begin file fts3_aux.c *****

63256. Right and left size of LIKE operator

63257. Condition 4

63258. DELETE, or UPDATE and return

63259. For a BITVEC_SZ of 512, this would be 34,359,739.

63260. Figure out the total size of the current row in tokens.

63261. 1 value
63262. The MEM_Cleared bit is only allowed on NULLs
63263. pItr now points at the 64-bit integer key value, a variable length ** integer. The following block moves pItr to point at the first byte ** past the end of the key value.
63264. ** Allocate memory, either lookaside (if possible) or heap. ** If the allocation fails, set the mallocFailed flag in ** the connection pointer. *** ** If db!=0 and db->mallocFailed is true (indicating a prior malloc ** failure on the same database connection) then always return 0. ** Hence for a particular database connection, once malloc starts ** failing, it fails consistently until mallocFailed is reset. ** This is an important assumption. There are many places in the ** code that do things like this: *** ** int *a = (int*)sqlite3DbMallocRaw(db, 100); ** int *b = (int*)sqlite3DbMallocRaw(db, 200); ** if(b) a[10] = 9; *** ** In other words, if a subsequent malloc (ex: "b") worked, it is assumed ** that all prior mallocs (ex: "a") worked too. *** ** The sqlite3MallocRawNN() variant guarantees that the "db" parameter is ** not a NULL pointer.
63265. Which level of the tree we are on
63266. (296) constlist_opt ::= COMMA constlist
63267. ***** Begin file global.c *****
63268. Read the page-size and sector-size journal header fields.
63269. Read the nKeep field of the next term.
63270. Pager page handle
63271. Saved value of db->nChange
63272. The blocking transaction has been concluded. Or there never was a ** blocking transaction. In either case, invoke the notify callback ** immediately.
63273. read only FS/ lockless
63274. ** Allocate a new MergeEngine object capable of handling up to ** nReader PmaReader inputs. *** ** nReader is automatically rounded up to the next power of two. ** nReader may not exceed SORTER_MAX_MERGE_COUNT even after rounding up.
63275. Jump here to continue with next cycle
63276. ** Rtree virtual table module xOpen method.
63277. ** The sizes for serial types less than 128
63278. pDone always used on sub-journals
63279. Last freeblock extends past page end
63280. PREFIX
63281. ** Compare the key of the index entry that cursor pC is pointing to against ** the key string in pUnpacked. Write into *pRes a number ** that is negative, zero, or positive if pC is less than, equal to, ** or greater than pUnpacked. Return SQLITE_OK on success. *** ** pUnpacked is either created without a rowid or is truncated so that it ** omits the rowid at the end. The rowid at the end of the index entry ** is ignored as well. Hence, this routine only compares the prefixes ** of the keys prior to the final rowid, not the entire key.
63282. New binary node to insert into expression tree
63283. ** Ensure that the sub-journal file is open. If it is already open, this ** function is a no-op. *** ** SQLITE_OK is returned if everything goes according to plan. An ** SQLITE_IOERR_XXX error code is returned if a call to sqlite3OsOpen() ** fails.
63284. IN/OUT: Pointer to poslist
63285. If (res==0) is true, then pRec must be equal to sample i.
63286. Jump here for next cycle
63287. Context pointer for xChunk callback
63288. **comment:** ** Set bit pngo of the BtShared.pHasContent bitvec. This is called ** when a page that previously contained data becomes a free-list leaf ** page. *** ** The BtShared.pHasContent bitvec exists to work around an obscure ** bug caused by the interaction of two useful IO optimizations surrounding ** free-list leaf pages: *** ** 1) When all data is deleted from a page and the page becomes ** a free-list leaf page, the page is not written to the database ** (as free-list leaf pages contain no meaningful data). Sometimes ** such a page is not even journaled (as it will not be modified, ** why bother journailling it?). *** ** 2) When a free-list leaf page is reused, its content is not read ** from the database or written to the journal file (why should it ** be, if it is not at all meaningful?). *** By themselves, these optimizations work fine and provide a handy ** performance boost to bulk delete or insert operations. However, if ** a page is moved to the free-list and then reused within the same ** transaction, a problem comes up. If the page is not journaled when ** it is moved to the free-list and it is also not journaled when it ** is extracted from the free-list and reused, then the original data ** may be lost. In the event of a rollback, it may not be possible ** to restore the database to its original configuration. *** ** The solution is the BtShared.pHasContent bitvec. Whenever a page is ** moved to become a free-list leaf page, the corresponding bit is ** set in the bitvec. Whenever a leaf page is extracted from the free-list, ** optimization 2 above is omitted if the corresponding bit is already ** set in BtShared.pHasContent. The contents of the bitvec are cleared ** at the end of every transaction.
label: code-design
63289. ** 2007 October 14 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the C functions that implement a memory ** allocation subsystem for use by SQLite. *** ** This version of the memory allocation subsystem omits all ** use of malloc(). The SQLite user supplies a block of memory ** before calling sqlite3_initialize() from which allocations ** are made and returned by the xMalloc() and xRealloc() ** implementations. Once sqlite3_initialize() has been called, ** the amount of memory available to SQLite is fixed and cannot ** be changed. *** ** This version of the memory allocation subsystem is included ** in the build only if SQLITE_ENABLE_MEMSYS3 is defined.
63290. **comment:** Next unused slot in aEquiv[]
label: code-design
63291. Locate the index named by the INDEXED BY clause, if any.
63292. Top of the co-routine
63293. **comment:** ** This function returns the collation sequence for database native text ** encoding identified by the string zName, length nName. *** ** If the requested collation sequence is not available, or not available ** in the database native encoding, the collation factory is invoked to ** request it. If the collation factory does not supply such a sequence, ** and the sequence is available in another text encoding, then that is ** returned instead. *** ** If no versions of the requested collations sequence are available, or ** another error occurs, NULL is returned and an error message written into ** pParse. *** ** This routine is a wrapper around sqlite3FindCollSeq(). This routine ** invokes the collation factory if the named collation cannot be found ** and generates an error message. ** See also: sqlite3FindCollSeq(), sqlite3GetCollSeq()
label: code-design
63294. The PRIMARY KEY index
63295. First time through the loop
63296. SQLITE_OS_WINCE
63297. ** Add N to the value of a status record. The caller must hold the ** appropriate mutex. (Locking is checked by assert()). *** ** The StatusUp() routine can accept positive or negative values for N. ** The value of N is added to the current status value and the high-water ** mark is adjusted if necessary. *** ** The StatusDown() routine lowers the current value by N. The highwater ** mark is unchanged. N must be non-negative for StatusDown().
63298. 7
63299. Loop over tables of a schema
63300. **comment:** ** Return a nul-terminated string consisting of nByte comma separated ** "?" expressions. For example, if nByte is 3, return a pointer to ** a buffer containing the string "?,?,?". *** ** The memory for the returned string is obtained from sqlite3_malloc(). ** It is the responsibility of the caller to eventually free it using ** sqlite3_free(). *** ** If an OOM error is encountered when allocating space for the new ** string, an error code is left in the rhu handle passed as the first ** argument and NULL is returned. Or, if an error has already occurred ** when this function is called, NULL is returned immediately, without ** attempting the allocation or modifying the stored error code.
label: code-design
63301. SQLITE_AFF_INTEGER
63302. The "NEAR" node (FTS5_TERM)
63303. Undo the local locks
63304. ** Allowed values for RowSet.rsFlags
63305. ** Make sure all writes to a particular file are committed to disk. *** ** If dataOnly==0 then both the file itself and its metadata (file ** size, access time, etc) are synced. If dataOnly!=0 then only the ** file data is synced. *** ** Under Unix, also make sure that the directory entry for the file ** has been created by fsync-ing

the directory that contains the file. ** If we do not do this and we encounter a power failure, the directory ** entry for the journal might not exist after we reboot. The next ** SQLite to access the file will not know that the journal exists (because ** the directory entry for the journal was never created) and the transaction ** will not roll back - possibly leading to database corruption.

63306. ** If LcsIterator.iCol is set to the following value, the iterator has ** finished iterating through all offsets for all columns.

63307. Force an SQLITE_TOOBIG error.

63308. ** Mark the VDBE as one that can only be run one time.

63309. SQLITE_STATUS_MALLOC_SIZE

63310. If this is an EXPLAIN, skip this step

63311. Number of key fields in each record

63312. ** Context pointer passed down through the tree-walk.

63313. Text to tokenize

63314. An SQLITE_NOMEM error.

63315. True if resolving names in a CHECK constraint

63316. Shared memory segment used for locking

63317. Fts5 Backend to iterate within

63318. OUT: Prepared SELECT statement

63319. If no error has occurred, return the calculated score. Otherwise, ** throw an SQL exception.

63320. FAIL => ID

63321. ***** Begin file sqliteInt.h *****

63322. File descriptor for database

63323. ***** Begin file pager.h *****

63324. ** An SQL user-function registered to do the work of an ATTACH statement. The ** three arguments to the function come directly from an attach statement: **
** ATTACH DATABASE x AS y KEY z ** ** SELECT sqlite_attach(x, y, z) ** ** If the optional "KEY z" syntax is omitted, an SQL NULL is passed as the ** third argument.

63325. Space for pStep->zTarget

63326. When doing a nested parse, one can include terms in an expression ** that look like this: #1 #2 ... These terms refer to registers ** in the virtual machine. #N is the N-th register.

63327. getenv("USERPROFILE")

63328. ** Add WhereLoop entries to handle OR terms. This works for either ** btrees or virtual tables.

63329. ** Structure for changeset iterators.

63330. Number of columns in table zName

63331. 0x15

63332. Changes whenever database file changes

63333. Size of argument array

63334. sz value

63335. Number of bytes available on the local btree page

63336. End of the PRAGMA switch

63337. Call lockBtree() until either pBt->pPage1 is populated or ** lockBtree() returns something other than SQLITE_OK. lockBtree() ** may return SQLITE_OK but leave pBt->pPage1 set to 0 if after ** reading page 1 it discovers that the page-size of the database ** file is not pBt->pageSize. In this case lockBtree() will update ** pBt->pageSize to the page-size of the file on disk.

63338. ** Close a VDBE cursor and release all the resources that cursor ** happens to hold.

63339. Flags to pass to OsSync() (or 0)

63340. UTF-8 -> UTF-16 Big-endian

63341. ***** Begin file os_common.h *****

63342. If the sub-journal was opened successfully (or was already open), ** write the journal record into the file.

63343. Unlock notify callback

63344. Last position from pp1

63345. xCreate

63346. Fake SrcList for pParse->pNewTable

63347. Transaction state

63348. ** The argument is an IN operator with a list (not a subquery) on the ** right-hand side. Return TRUE if that list is constant.

63349. ***** End of tokenize.c *****

63350. Conditional skipped

63351. Pretend we already have a lock

63352. P4 is a pointer to a transient string

63353. ONEPASS is ok with multiple rows

63354. Ephemeral table used for DISTINCT processing

63355. In theory, it is only necessary to write the 28 bytes that the ** journal header consumes to the journal file here. Then increment the ** Pager.journalOff variable by JOURNAL_HDR_SZ so that the next ** record is written to the following sector (leaving a gap in the file ** that will be implicitly filled in by the OS). *** However it has been discovered that on some systems this pattern can ** be significantly slower than contiguously writing data to the file, ** even if that means explicitly writing data to the block of ** (JOURNAL_HDR_SZ - 28) bytes that will not be used. So that is what ** is done. *** The loop is required here in case the sector-size is larger than the ** database page size. Since the zHeader buffer is only Pager.pageSize ** bytes in size, more than one call to sqlite3OsWrite() may be required ** to populate the entire journal header sector.

63356. This routine is only called from b-tree and only when there are no ** outstanding pages. This implies that the pager state should either ** be OPEN or READER. READER is only possible if the pager is or was in ** exclusive access mode.

63357. Find the first two PmaReaders to compare. The one that was just ** advanced (iPrev) and the one next to it in the array.

63358. **comment:** ** Indicate that all subsequent calls to sqlite3Fts5IndexWrite() pertain ** to the document with rowid iRowid.
label: documentation

63359. Pointer to database schema (possibly shared)

63360. ** Return the size of the database file in pages. If there is any kind of ** error, return ((unsigned int)-1).

63361. Step 2: If the page was found in the hash table, then return it. ** If the page was not in the hash table and createFlag is 0, abort. ** Otherwise (page not in hash and createFlag!=0) continue with ** subsequent steps to try to create the page.

63362. Load the entire master journal file into space obtained from ** sqlite3_malloc() and pointed to by zMasterJournal. Also obtain ** sufficient space (in zMasterPtr) to hold the names of master ** journal files extracted from regular rollback-journals.

63363. Corruption

63364. ** Commute a comparison operator. Expressions of the form "X op Y" ** are converted into "Y op X". ** ** If left/right precedence rules come into play when determining the ** collating sequence, then COLLATE operators are adjusted to ensure ** that the collating sequence does not change. For example: ** "Y collate NOCASE op X" becomes "X op Y" because any collation sequence on ** the left hand side of a comparison overrides any collation sequence ** attached to the right. For the same reason the EP_Collate flag ** is not commuted.

63365. SQLITE_ENABLE_COLUMN_USED_MASK

63366. zFilename

63367. ** Write meta-information back into the database. Meta[0] is ** read-only and may not be written.

63368. ** This structure is a subclass of sqlite3_file. Each open memory-journal ** is an instance of this class.

63369. ADD

63370. Value to return

63371. Access to the winShmNode object is serialized by the caller

63372. Names of table columns

63373. ** Check to see if the pThis entry of pTabList is a self-join of a prior view. ** If it is, then return the SrcList_item for the prior view. If it is not, ** then return 0.

63374. New value for pMem->n
63375. Skip indices that do not change
63376. True if "-" flag is present
63377. ** Compute the Hour, Minute, and Seconds from the julian day number.
63378. Functions used only by user authorization logic
63379. Auth callback return code
63380. ** CAPI3REF: Determine If An SQL Statement Is Complete ** ** These routines are useful during command-line input to determine if the ** currently entered text seems to form a complete SQL statement or ** if additional input is needed before sending the text into ** SQLite for parsing. ^These routines return 1 if the input string ** appears to be a complete SQL statement. ^A statement is judged to be ** complete if it ends with a semicolon token and is not a prefix of a ** well-formed CREATE TRIGGER statement. ^Semicolons that are embedded within ** string literals or quoted identifier names or comments are not ** independent tokens (they are part of the token in which they are ** embedded) and thus do not count as a statement terminator. ^Whitespace ** and comments that follow the final semicolon are ignored. ** ** ^These routines return 0 if the statement is incomplete. ^If a ** memory allocation fails, then SQLITE_NOMEM is returned. ** ** ^These routines do not parse the SQL statements thus ** will not detect syntactically incorrect SQL. ** ** ^^(If SQLite has not been initialized using [sqlite3_initialize()] prior ** to invoking sqlite3_complete16() then sqlite3_initialize() is invoked ** automatically by sqlite3_complete16(). If that initialization fails, ** then the return value from sqlite3_complete16() will be non-zero ** regardless of whether or not the input SQL is complete.)^ ** ** The input to [sqlite3_complete()] must be a zero-terminated ** UTF-8 string. ** ** The input to [sqlite3_complete16()] must be a zero-terminated ** UTF-16 string in native byte order.
63381. Initial value of write-counter
63382. Malloc failed while inserting the new entry. This can only ** happen if there was no previous entry for this token.
63383. Extra information associated with the page
63384. All the rest are undocumented and are for internal use only
63385. **comment:** Clean up the WHERE clause constructed above.
 label: code-design
63386. Floating point sum
63387. The following structure represents a single element of the ** parser's stack. Information stored includes: ** ** + The state number for the parser at this level of the stack. ** ** + The value of the token stored at this level of the stack. ** (In other words, the "major" token.) ** ** + The semantic value stored at this level of the stack. This is ** the information used by the action routines in the grammar. ** It is sometimes called the "minor" token. ** ** After the "shift" half of a SHIFTREDUCE action, the stateno field ** actually contains the reduce action for the second half of the ** SHIFTREDUCE.
63388. ***** Include pragma.h in the middle of pragma.c *****
63389. If we compiled with the SQLITE_NO_SYNC flag, then syncing is a ** no-op. But go ahead and call fstat() to validate the file ** descriptor as we need a method to provoke a failure during ** coverage testing.
63390. Set default values
63391. ccons ::= DEFAULT MINUS term
63392. If foreign-key support is enabled, rewrite the CREATE TABLE ** statements corresponding to all child tables of foreign key constraints ** for which the renamed table is the parent table.
63393. (1)
63394. Mask of src visited by(..)
63395. SQLITE_MISUSE
63396. Make sure all objects are contained in this database
63397. One of SHARED_LOCK, RESERVED_LOCK etc.
63398. unsigned int szPma
63399. Opcode: VUpdate P1 P2 P3 P4 P5 ** Synopsis: data=r[P3@P2] ** ** P4 is a pointer to a virtual table object, an sqlite3_vtab structure. ** This opcode invokes the corresponding xUpdate method. P2 values ** are contiguous memory cells starting at P3 to pass to the xUpdate ** invocation. The value in register (P3+P2-1) corresponds to the ** p2th element of the argv array passed to xUpdate. ** ** The xUpdate method will do a DELETE or an INSERT or both. ** The argv[0] element (which corresponds to memory cell P3) ** is the rowid of a row to delete. If argv[0] is NULL then no ** deletion occurs. The argv[1] element is the rowid of the new ** row. This can be NULL to have the virtual table select the new ** rowid for itself. The subsequent elements in the array are ** the values of columns in the new row. ** ** If P2==1 then no insert is performed. argv[0] is the rowid of ** a row to delete. ** ** P1 is a boolean flag. If it is set to true and the xUpdate call ** is successful, then the value returned by sqlite3_last_insert_rowid() ** is set to the value of the rowid for the row just inserted. ** ** P5 is the error actions (OE_Replacement, OE_Fail, OE_Ignore, etc) to ** apply in the case of a constraint failure on an insert or update.
63400. Output leaf pages left to write
63401. Check if the main database is a zipvfs db. If it is, set the upper ** level pager to use "journal_mode=off". This prevents it from ** generating a large journal using a temp file.
63402. The beginning and end of the LRU list
63403. Number of key columns in the index
63404. This function should only be called on a sharable b-tree after it ** has been determined that no other b-tree holds a conflicting lock.
63405. ** Convert the N-th element of pStmt->pColName[] into a string using ** xFunc() then return that string. If N is out of range, return 0. ** ** There are up to 5 names for each column. useType determines which ** name is returned. Here are the names: ** ** 0 The column name as it should be displayed for output ** 1 The datatype name for the column ** 2 The name of the database that the column derives from ** 3 The name of the table that the column derives from ** 4 The name of the table column that the result column derives from ** ** If the result is not a simple column reference (if it is an expression ** or a constant) then useTypes 2, 3, and 4 return NULL.
63406. ** Rollback the transaction in progress. ** ** If tripCode is not SQLITE_OK then cursors will be invalidated (tripped). ** Only write cursors are tripped if writeOnly is true but all cursors are ** tripped if writeOnly is false. Any attempt to use ** a tripped cursor will result in an error. ** ** This will release the write lock on the database file. If there ** are no active cursors, it also releases the read lock.
63407. ** All of the code in this file may be omitted by defining a single ** macro.
63408. Size of IndexSample.anEq[] and so on
63409. If the iterator is already at the end of the changeset, return DONE.
63410. ** Use sqlite3OsFileControl() when we are doing something that might fail ** and we need to know about the failures. Use sqlite3OsFileControlHint() ** when simply tossing information over the wall to the VFS and we do not ** really care if the VFS receives and understands the information since it ** is only a hint and can be safely ignored. The sqlite3OsFileControlHint() ** routine has no return value since the return value would be meaningless.
63411. Opcode: Column P1 P2 P3 P4 P5 ** Synopsis: r[P3]=PX ** ** Interpret the data that cursor P1 points to as a structure built using ** the MakeRecord instruction. (See the MakeRecord opcode for additional ** information about the format of the data.) Extract the P2-th column ** from this record. If there are less than (P2+1) ** values in the record, extract a NULL. ** ** The value extracted is stored in register P3. ** ** If the record contains fewer than P2 fields, then extract a NULL. Or, ** if the P4 argument is a P4_MEM use the value of the P4 argument as ** the result. ** ** If the OPFLAG_CLEARCACHE bit is set on P5 and P1 is a pseudo-table cursor, ** then the cache of the cursor is reset prior to extracting the column. ** The first OP_Column against a pseudo-table after the value of the content ** register has changed should have this bit set. ** ** If the OPFLAG_LENGTHARG and OPFLAG_TYPEOFARG bits are set on P5 then ** the result is guaranteed to only be used as the argument of a length() ** or typeof() function, respectively. The loading of large blobs can be ** skipped for length() and all content loading can be skipped for typeof().
63412. ** This function is used to help parse position-lists. When this function is ** called, *pp may point to the start of the next varint in the position-list ** being parsed, or it may point to 1 byte past the end of the position-list ** (in which case ** pp will be a terminator bytes POS_END (0) or ** (1)). ** ** If *pp points past the end of the current position-list, set *pi to ** POSITION_LIST_END and return. Otherwise, read the next varint from *pp, ** increment the current value of *pi by the value read, and set *pp to ** point to the next value before returning. ** ** Before calling this routine *pi must be initialized to the value of ** the previous position, or zero if we are reading the first position ** in the position-list. Because positions are delta-encoded, the value ** of the previous position is needed in order to compute the value of ** the next position.
63413. ** Implementation of FTS3 xRename method. Rename an fts3 table.
63414. First byte past the iStart buffer
63415. worrying about sub-dividing and re-hashing.
63416. EVIDENCE-OF: R-57343-49114 Value is a big-endian IEEE 754-2008 64-bit ** floating point number.
63417. Table to open

63418. Last possible cell index
63419. ** This routine checks that the sqlite3.nVdbeActive count variable ** matches the number of vdbe's in the list sqlite3.pVdbe that are ** currently active. An assertion fails if the two counts do not match. ** This is an internal self-check only - it is not an essential processing ** step. *** This is a no-op if NDEBUG is defined.
63420. 70..77 pqrsuvwxyz
63421. (328) anylist ::= anylist ANY
63422. Root frame of VDBE
63423. Compound SELECTs that have an ORDER BY clause are handled separately.
63424. The IdxRowid and Seek opcodes are combined because of the commonality ** of sqlite3VdbeCursorRestore() and sqlite3VdbeIdxRowid().
63425. Suffix length
63426. #define WAL_HDRSIZE 24
63427. Like RESTART but also truncate WAL
63428. **comment:** ** Clean up a VDBE after execution but do not delete the VDBE just yet. ** Write any error messages into *pzErrMsg. Return the result code. *** After this routine is run, the VDBE should be ready to be executed ** again. *** To look at it another way, this routine resets the state of the ** virtual machine from VDBE_MAGIC_RUN or VDBE_MAGIC_HALT back to ** VDBE_MAGIC_INIT.
label: code-design
63429. ** PRAGMA data_store_directory ** PRAGMA data_store_directory = ""||"directory_name" ** ** Return or set the local value of the data_store_directory flag. Changing ** the value sets a specific directory to be used for database files that ** were specified with a relative pathname. Setting to a null string reverts ** to the default database directory, which for database files specified with ** a relative path will probably be based on the current directory for the ** process. Database file specified with an absolute path are not impacted ** by this setting, regardless of its value. **
63430. If sqlite3_prepare returns a tail pointer, we calculate the ** equivalent pointer into the UTF-16 string by counting the unicode ** characters between zSql8 and zTail8, and then returning a pointer ** the same number of characters into the UTF-16 string.
63431. If the cache contains a page with page-number pgno, remove it ** from its hash chain. Also, if the PGHDR_NEED_SYNC flag was set for ** page pgno before the 'move' operation, it needs to be retained ** for the page moved there.
63432. ** Generate code for the trigger program associated with trigger p on ** table pTab. The reg, orconf and ignoreJump parameters passed to this ** function are the same as those described in the header function for ** sqlite3CodeRowTrigger()
63433. **comment:** ** Pointers to the end of sqlite3GlobalConfig.pScratch memory ** (so that a range test can be used to determine if an allocation ** being freed came from pScratch) and a pointer to the list of ** unused scratch allocations.
label: code-design
63434. 74
63435. The last field of the index should be an integer - the ROWID. ** Verify that the last entry really is an integer.
63436. Best score of columns checked so far
63437. **comment:** Database connection for malloc errors
label: code-design
63438. ** Swap two objects of type TYPE.
63439. ** Write the nRec Field - the number of page records that follow this ** journal header. Normally, zero is written to this value at this time. ** After the records are added to the journal (and the journal synced, ** if in full-sync mode), the zero is overwritten with the true number ** of records (see syncJournal()). *** A faster alternative is to write 0xFFFFFFFF to the nRec field. When ** reading the journal this value tells SQLite to assume that the ** rest of the journal file contains valid page records. This assumption ** is dangerous, as if a failure occurred whilst writing to the journal ** file it may contain some garbage data. There are two scenarios ** where this risk can be ignored: *** ** When the pager is in no-sync mode. Corruption can follow a ** power failure in this case anyway. *** ** When the SQLITE_IOCAP_SAFE_APPEND flag is set. This guarantees ** that garbage data is never appended to the journal file.
63440. Determine whether we should code this trigger
63441. This both clears the schemas and reduces the size of the db->aDb[] ** array.
63442. ** Return a copy of index structure pStruct. Except, promote as many ** segments as possible to level iPromote. If an OOM occurs, NULL is ** returned.
63443. ** CAPI3REF: Database Snapshot ** KEYWORDS: {snapshot} {sqlite3_snapshot} ** EXPERIMENTAL ** ** An instance of the snapshot object records the state of a [WAL mode] ** database for some specific point in history. *** In [WAL mode], multiple [database connections] that are open on the ** same database file can each be reading a different historical version ** of the database file. When a [database connection] begins a read ** transaction, that connection sees an unchanging copy of the database ** as it existed for the point in time when the transaction first started. ** Subsequent changes to the database from other connections are not seen ** by the reader until a new read transaction is started. *** The sqlite3_snapshot object records state information about an historical ** version of the database file so that it is possible to later open a new read ** transaction that sees that historical version of the database rather than ** the most recent version. *** The constructor for this object is [sqlite3_snapshot_get()]. The ** [sqlite3_snapshot_open()] method causes a fresh read transaction to refer ** to an historical snapshot (if possible). The destructor for ** sqlite3_snapshot objects is [sqlite3_snapshot_free()].
63444. Waiting connection
63445. refact ::= NO ACTION
63446. Open file handle on the write-ahead log file.
63447. Return no rowids later than this
63448. OUT: Set to 1 if EOF
63449. ** This function is used internally to remove the page pPage from the ** PGroup LRU list, if is part of it. If pPage is not part of the PGroup ** LRU list, then this function is a no-op. ** ** The PGroup mutex must be held when this function is called.
63450. ** Argument p points to a buffer containing utf-8 text that is n bytes in ** size. Return the number of bytes in the nChar character prefix of the ** buffer, or 0 if there are less than nChar characters in total.
63451. If the module has been registered and includes a Create method, ** invoke it now. If the module has not been registered, return an ** error. Otherwise, do nothing.
63452. **comment:** Only JUMP opcodes and the short list of special opcodes in the switch ** below need to be considered. The mkopcodeh.tcl generator script groups ** all these opcodes together near the front of the opcode list. Skip ** any opcode that does not need processing by virtue of the fact that ** it is larger than SQLITE_MAX_JUMP_OPCODE, as a performance optimization.
label: code-design
63453. ** This variable is set to false when running tests for which the on disk ** structures should not be corrupt. Otherwise, true. If it is false, extra ** assert() conditions in the fts5 code are activated - conditions that are ** only true if it is guaranteed that the fts5 database is not corrupt.
63454. IMP: R-15569-63625
63455. 610
63456. Pointer p currently points at the first byte of an offset list. The ** following block advances it to point one byte past the end of ** the same offset list.
63457. The name of the new table
63458. Change-list for UPDATE, NULL for DELETE
63459. ** Append a record of the current state of page pPg to the sub-journal. ** ** If successful, set the bit corresponding to pPg->pgno in the bitvecs ** for all open savepoints before returning. *** This function returns SQLITE_OK if everything is successful, an IO ** error code if the attempt to write to the sub-journal fails, or ** SQLITE_NOMEM if a malloc fails while setting a bit in a savepoint ** bitvec.
63460. If >0, bytes of buffer aNode[] loaded
63461. Buffer to append to
63462. The following callback (if not NULL) is invoked on every VDBE branch ** operation. Set the callback using SQLITE_TESTCTRL_VDBE_COVERAGE.
63463. OUT: Created tokenizer
63464. Only ephemeral cursors can be duplicated
63465. Number of columns in current object
63466. Index of database containing pTab
63467. Compute the total free space on the page ** EVIDENCE-OF: R-23588-34450 The two-byte integer at offset 1 gives the ** start of the first freeblock on the page, or is zero if there are no ** freeblocks.
63468. ** Estimate the location of a particular key among all keys in an ** index. Store the results in aStat as follows: *** aStat[0] Est. number of rows less than pRec ** aStat[1] Est. number of rows equal to pRec ** ** Return the index of the sample that is the smallest sample that ** is greater than or equal to pRec. Note that

this index is not an index ** into the aSample[] array - it is an index into a virtual set of samples ** based on the contents of aSample[] and the number of fields in record ** pRec.

63469. Value of 'ncell' column

63470. if(pSrcList)

63471. For a table with implicit rowids, append "old._rowid_" to the list.

63472. The checkpoint status information

63473. synopsis: r[P2]=P1

63474. Number of pages in the database file

63475. Left input list

63476. One of the WHERE_DISTINCT_* values

63477. 1x

63478. Locate and if necessary initialize the target table object

63479. Truncate WAL to this size on reset

63480. 28

63481. Loop through each expression in <exprlist>.

63482. ***** Begin file complete.c *****

63483. The string to compare against the glob

63484. The locking context for the proxy lock

63485. Regs for sorter record

63486. A direct lookup on the rowid or docid column. Assign a cost of 1.0.

63487. If an error occurred above, free the Pager structure and close the file.

63488. ROWID is never NULL

63489. True (1) if Y,M,D are valid

63490. ** The results of a SELECT can be distributed in several ways, as defined ** by one of the following macros. The "SRT" prefix means "SELECT Result ** Type".

*** SRT_Union Store results as a key in a temporary index ** identified by pDest->iSDParm. *** SRT_Except Remove results from the temporary index pDest->iSDParm. *** SRT_Exists Store a 1 in memory cell pDest->iSDParm if the result ** set is not empty. *** SRT_Discard Throw the results away. This is used by SELECT ** statements within triggers whose only purpose is ** the side-effects of functions. *** All of the above are free to ignore their ORDER BY clause. Those that ** follow must honor the ORDER BY clause. *** SRT_Output Generate a row of output (using the OP_ResultRow ** opcode) for each row in the result set. *** SRT_Mem Only valid if the result is a single column. ** Store the first column of the first result row ** in register pDest->iSDParm then abandon the rest ** of the query. This destination implies "LIMIT 1". *** SRT_Set The result must be a single column. Store each ** row of result as the key in table pDest->iSDParm. ** Apply the affinity pDest->affSdst before storing ** results. Used to implement "IN (SELECT ...)". *** SRT_EphemTab Create an temporary table pDest->iSDParm and store ** the result there. The cursor is left open after ** returning. This is like SRT_Table except that ** this destination uses OP_OpenEphemeral to create ** the table first. *** SRT_Coroutine Generate a co-routine that returns a new row of ** results each time it is invoked. The entry point ** of the co-routine is stored in register pDest->iSDParm ** and the result row is stored in pDest->nDest registers ** starting with pDest->iSdst. *** SRT_Table Store results in temporary table pDest->iSDParm. ** SRT_Fifo This is like SRT_EphemTab except that the table ** is assumed to already be open. SRT_Fifo has ** the additional property of being able to ignore ** the ORDER BY clause. *** SRT_DistFifo Store results in a temporary table pDest->iSDParm. ** But also use temporary table pDest->iSDParm+1 as ** a record of all prior results and ignore any duplicate ** rows. Name means: "Distinct Fifo". *** SRT_Queue Store results in priority queue pDest->iSDParm (really ** an index). Append a sequence number so that all entries ** are distinct. *** SRT_DistQueue Store results in priority queue pDest->iSDParm only if ** the same record has never been stored before. The ** index at pDest->iSDParm+1 hold all prior stores.

63491. Name of trigger

63492. Size of output buffer in bytes

63493. ** If the memory cell contains a value that must be freed by ** invoking the external callback in Mem.xDel, then this routine ** will free that value. It also sets Mem.flags to MEM_Null. ** This is a helper routine for sqlite3VdbeMemSetNull() and ** for sqlite3VdbeMemRelease(). Use those other routines as the ** entry point for releasing Mem resources.

63494. bNoLock

63495. ***** Begin file utf.c *****

63496. ** Allocate heap space to hold an Index object with nCol columns. ** Increase the allocation size to provide an extra nExtra bytes ** of 8-byte aligned space after the Index object and return a ** pointer to this extra space in *ppExtra.

63497. A fake table from which we get the result set

63498. ** These values must match the values defined in wal.c for the equivalent ** locks. These are not magic numbers as they are part of the SQLite file ** format.

63499. Index column used for ORDER BY

63500. 00..07

63501. Data to decode list-of-rowids from

63502. Database holding FTS index (e.g. "main")

63503. ** xSetOutputs callback used when: ** detail=col, ** there is a column filter, and ** the table contains 100 or fewer columns. ** The last point is to ensure all column numbers are stored as ** single-byte varints.

63504. ***** Begin file auth.c *****

63505. **comment:** Not needed. Only used to silence a warning.

label: requirement

63506. Iterator

63507. build the NEW.* reference row. Note that if there is an INTEGER ** PRIMARY KEY into which a NULL is being inserted, that NULL will be ** translated into a unique ID for the row. But on a BEFORE trigger, ** we do not know what the unique ID will be (because the insert has ** not happened yet) so we substitute a rowid of -1

63508. Do Nothing

63509. Determine the value of the flags parameter passed to POSIX function ** open(). These must be calculated even if open() is not called, as ** they may be stored as part of the file handle and used by the ** 'conch file' locking functions later on.

63510. 301

63511. The byte immediately before the last 0x00 byte has the 0x80 bit ** set. So the last 0x00 is only a varint 0 if there are 8 more 0x80 ** bytes before a[ii].

63512. One of the SQLITE_AFF... values

63513. ** The first value in the apVal[] array is assumed to contain an integer. ** This function tests if there exist any documents with docid values that ** are different from that integer. i.e. if deleting the document with docid ** pRowid would mean the FTS3 table were empty. *** If successful, *isEmpty is set to true if the table is empty except for ** document pRowid, or false otherwise, and SQLITE_OK is returned. If an ** error occurs, an SQLite error code is returned.

63514. Array of primary key flags, or NULL

63515. Temporary expression node

63516. 200

63517. Configured FCNTL_MMAP_SIZE value

63518. OUT: Size of buffer at *ppOut

63519. The PRIMARY KEY index for WITHOUT ROWID tables

63520. Special query text

63521. Number of entries in alIdx[]

63522. ** Return a boolean value for a query parameter.

63523. The table the contains the triggers

63524. Synthesized from VALUES clause

63525. ** Buffer abBuff[] contains a list of varints, all small enough to fit ** in a 32-bit integer. Return the size of the largest prefix of this ** list nMax bytes or less in size.

63526. DATA, MISSING, CONFLICT, CONSTRAINT

63527. Trigger opcode

63528. FTS5 Configuration

63529. Sync the master journal file. If the IOCAP_SEQUENTIAL device ** flag is set this is not required.
63530. ccons ::= defer_subclause
63531. ** Delete a virtual table handle allocated by fts5InitVtab().
63532. For looping over pages
63533. same as TK_CONCAT, synopsis: r[P3]=r[P2]+r[P1]
63534. ***** End of whereexpr.c *****
63535. Literal "old" token
63536. **comment:** ** CAPI3REF: Column Names In A Result Set ** METHOD: sqlite3_stmt ** ** ^These routines return the name assigned to a particular column ** in the result set of a [SELECT] statement. ^The sqlite3_column_name() ** interface returns a pointer to a zero-terminated UTF-8 string ** and sqlite3_column_name16() returns a pointer to a zero-terminated ** UTF-16 string. ^The first parameter is the [prepared statement] ** that implements the [SELECT] statement. ^The second parameter is the ** column number. ^The leftmost column is number 0. ** ** ^The returned string pointer is valid until either the [prepared statement] ** is destroyed by [sqlite3_finalize()] or until the statement is automatically ** reprepared by the first call to [sqlite3_step()] for a particular run ** or until the next call to ** sqlite3_column_name() or sqlite3_column_name16() on the same column. ** ** ^If sqlite3_malloc() fails during the processing of either routine ** (for example during a conversion from UTF-8 to UTF-16) then a ** NULL pointer is returned. ** ** ^The name of a result column is the value of the "AS" clause for ** that column, if there is an AS clause. If there is no AS clause ** then the name of the column is unspecified and may change from ** one release of SQLite to the next.
label: code-design
63537. Tokenizer creation arguments
63538. **comment:** Expressions of the form *** expr1 IN () ** expr1 NOT IN () ** ** simplify to constants 0 (false) and 1 (true), respectively, ** regardless of the value of expr1.
label: code-design
63539. Maximum page size. The upper bound on this value is 65536. This a limit ** imposed by the use of 16-bit offsets within each page. ** ** Earlier versions of SQLite allowed the user to change this value at ** compile time. This is no longer permitted, on the grounds that it creates ** a library that is technically incompatible with an SQLite library ** compiled with a different limit. If a process operating on a database ** with a page-size of 65536 bytes crashes, then an instance of SQLite ** compiled with the default page-size limit will not be able to rollback ** the aborted transaction. This could lead to database corruption.
63540. Currently there is an SQL level transaction open on the vacuum ** database. No locks are held on any other files (since the main file ** was committed at the btree level). So it safe to end the transaction ** by manually setting the autoCommit flag to true and detaching the ** vacuum database. The vacuum_db journal file is deleted when the pager ** is closed by the DETACH.
63541. ')
63542. ** Implementations of scalar functions for case mapping - upper() and ** lower(). Function upper() converts its input to upper-case (ABC). ** Function lower() converts to lower-case (abc). ** ** ICU provides two types of case mapping, "general" case mapping and ** "language specific". Refer to ICU documentation for the differences ** between the two. ** ** To utilise "general" case mapping, the upper() or lower() scalar ** functions are invoked with one argument: ** ** upper('ABC') -> 'abc' ** lower('abc') -> 'ABC' ** ** To access ICU "language specific" case mapping, upper() or lower() ** should be invoked with two arguments. The second argument is the name ** of the locale to use. Passing an empty string ("") or SQL NULL value ** as the second argument is the same as invoking the 1 argument version ** of upper() or lower(). ** ** lower('T', 'en_us') -> 'i' ** lower('T', 'tr_tr') -> '\u131' (small dotless i) ** ** http://www.icu-project.org/userguide posix.html#case_mappings
63543. **comment:** ** Prepare a statement used to insert rows into the "rbu_tmp_xxx" table. ** Specifically a statement of the form: *** INSERT INTO rbu_tmp_xxx VALUES(?, ?, ? ...); ** ** The number of bound variables is equal to the number of columns in ** the target table, plus one (for the rbu_control column), plus one more ** (for the rbu_rowid column) if the target table is an implicit IPK or ** virtual table.
label: code-design
63544. IMP: R-54100-20147
63545. ** This function is used by both blob_open() and blob_reopen(). It seeks ** the b-tree cursor associated with blob handle p to point to row iRow. ** If successful, SQLITE_OK is returned and subsequent calls to ** sqlite3_blob_read() or sqlite3_blob_write() access the specified row. ** ** If an error occurs, or if the specified row does not exist or does not ** contain a value of type TEXT or BLOB in the column nominated when the ** blob handle was opened, then an error code is returned and *pzErr may ** be set to point to a buffer containing an error message. It is the ** responsibility of the caller to free the error message buffer using ** sqlite3DbFree(). ** ** If an error does occur, then the b-tree cursor is closed. All subsequent ** calls to sqlite3_blob_read(), blob_write() or blob_reopen() will ** immediately return SQLITE_ABORT.
63546. **comment:** Prevent warning when SQLITE_THREADSAFE=0
label: code-design
63547. Generate new record numbers semi-randomly
63548. ** sqlite3_test_control(BITVEC_TEST, size, program) ** ** Run a test against a Bitvec object of size. The program argument ** is an array of integers that defines the test. Return -1 on a ** memory allocation error, 0 on success, or non-zero for an error. ** See the sqlite3BitvecBuiltinTest() for additional information.
63549. Pointer to vtab instance
63550. ** This function sets the value of the sqlite3_value object passed as the ** first argument to a copy of the string or blob held in the aData[] ** buffer. SQLITE_OK is returned if successful, or SQLITE_NOMEM if an OOM ** error occurs.
63551. Index of next byte to read from input
63552. xUnlock
63553. Omit if ExprList.u.x.iOrderByCol
63554. If there are active savepoints and any of them were created ** since the most recent journal header was written, update the ** PagerSavepoint.iHdrOffset fields now.
63555. ** Initialize a string accumulator. ** ** p: The accumulator to be initialized. ** db: Pointer to a database connection. May be NULL. Lookaside ** memory is used if not NULL. db->mallocFailed is set appropriately ** when not NULL. ** zBase: An initial buffer. May be NULL in which case the initial buffer ** is malloced. ** n: Size of zBase in bytes. If total space requirements never exceed ** n then no memory allocations ever occur. ** mx: Maximum number of bytes to accumulate. If mx==0 then no memory ** allocations will ever occur.
63556. If the QUERY_SCAN flag is set, all other flags must be clear.
63557. xTestCallback
63558. Page 1 of the database file
63559. File handle from winOpen
63560. Last element in link-list. Valid for 1st elem only
63561. Destructor for Mem.z - only valid if MEM_Dyn
63562. Preserve the default page cache size
63563. copy digits to exponent
63564. ** Return the collating function associated with a function.
63565. ** Free any prior content in *pz and replace it with a copy of zNew.
63566. Use this for the table cursor, if there is one
63567. The LHS of the IN operator
63568. pPager->stmtInUse = 0;
63569. Expressions defining the result set
63570. Context for active vtab connect/create
63571. ** Convert from YYYY-MM-DD HH:MM:SS to julian day. We always assume ** that the YYYY-MM-DD is according to the Gregorian calendar. ** ** Reference: Meeus page 61
63572. ** Performance statistics
63573. uniqueflag ::= UNIQUE
63574. Function definition
63575. SQL_SELECT_INDEXES ** Return the list of valid segment indexes for absolute level ?
63576. **comment:** ** This function is now an anachronism. It used to be used to recover from a ** malloc() failure, but SQLite now does this automatically.
label: code-design

63577. ** We need to define _XOPEN_SOURCE as follows in order to enable ** recursive mutexes on most Unix systems and fchmod() on OpenBSD. ** But _XOPEN_SOURCE define causes problems for Mac OS X, so omit ** it.

63578. Neither X nor Y have COLLATE operators, but X has a non-default ** collating sequence. So add the EP_Collate marker on X to cause ** it to be searched first.

63579. Create the INSERT statement to write to the target PK b-tree

63580. **comment:** ** Hard-coded maximum amount of data to accumulate in memory before flushing ** to a level 0 PMA. The purpose of this limit is to prevent various integer ** overflows. 512MiB.
label: code-design

63581. Fix items to this schema

63582. xAccess

63583. ** Each time a blob is read from the %_data table, it is padded with this ** many zero bytes. This makes it easier to decode the various record formats ** without overreading if the records are corrupt.

63584. Forward declaration for the function that does the work of ** the virtual table module xCreate() and xConnect() methods.

63585. ** Arguments pList/nList contain the doclist for token iToken of phrase p. ** It is merged into the main doclist stored in p->doclist.aAll/nAll. ** ** This function assumes that pList points to a buffer allocated using ** sqlite3_malloc(). This function takes responsibility for eventually ** freeing the buffer. ** ** SQLITE_OK is returned if successful, or SQLITE_NOMEM if an error occurs.

63586. True for descending docids

63587. It is acceptable to use a read-only (mmap) page for any page except ** page 1 if there is no write-transaction open or the ACQUIRE_READONLY ** flag was specified by the caller. And so long as the db is not a ** temporary or in-memory database.

63588. First Mem address for previous GROUP BY

63589. ** CAPI3REF: Session Object Handle

63590. ** The suggested maximum number of in-memory pages to use for ** the main database table and for temporary tables. ** ** IMPLEMENTATION-OF: R-30185-15359 The default suggested cache size is -2000, ** which means the cache size is limited to 2048000 bytes of memory. ** IMPLEMENTATION-OF: R-48205-43578 The default suggested cache size can be ** altered using the SQLITE_DEFAULT_CACHE_SIZE compile-time options.

63591. ***** End of notify.c *****

63592. **comment:** This is the 2-bit case and we are on the second iteration and ** current term is from the first iteration. So skip this term.
label: code-design

63593. Step 2

63594. ** This function is called by the xUpdate() method as part of an INSERT ** operation. It adds entries for each term in the new record to the ** pendingTerms hash table. ** ** Argument apVal is the same as the similarly named argument passed to ** fts3InsertData(). Parameter iDocid is the docid of the new row.

63595. ** If the expression passed as the only argument is of type TK_VECTOR ** return the number of expressions in the vector. Or, if the expression ** is a sub-select, return the number of columns in the sub-select. For ** any other type of expression, return 1.

63596. ** This function is called after transitioning from PAGER_UNLOCK to ** PAGER_SHARED state. It tests if there is a hot journal present in ** the file-system for the given pager. A hot journal is one that ** needs to be played back. According to this function, a hot-journal ** file exists if the following criteria are met: **
** * The journal file exists in the file system, and ** * No process holds a RESERVED or greater lock on the database file, and ** * The database file itself is greater than 0 bytes in size, and ** * The first byte of the journal file exists and is not 0x00. ** ** If the current size of the database file is 0 but a journal file ** exists, that is probably an old journal left over from a prior ** database with the same name. In this case the journal file is ** just deleted using OsDelete, *pExists is set to 0 and SQLITE_OK ** is returned. ** ** This routine does not check if there is a master journal filename ** at the end of the file. If there is, and that master journal file ** does not exist, then the journal file is not really hot. In this ** case this routine will return a false-positive. The pager_playback() ** routine will discover that the journal file is not really hot and ** will not roll it back. ** ** If a hot-journal file is found to exist, *pExists is set to 1 and ** SQLITE_OK returned. If no hot-journal file is present, *pExists is ** set to 0 and SQLITE_OK returned. If an IO error occurs while trying ** to determine whether or not a hot-journal file exists, the IO error ** code is returned and the value of *pExists is undefined.

63597. ** Debug tracing macros. Enable by changing the "0" to "1" and ** recompiling. ** ** When sqlite3PcacheTrace is 1, single line trace messages are issued. ** When sqlite3PcacheTrace is 2, a dump of the pcache showing all cache entries ** is displayed for many operations, resulting in a lot of output.

63598. Elements of the linked list at Vdbe.pAuxData

63599. Number of tokens in the phrase

63600. ** This routine reactivates the memory allocator and clears the ** db->mallocFailed flag as necessary. ** ** The memory allocator is not restarted if there are running ** VDBEs.

63601. current position in zInput

63602. Estimate the average row size for the table and for all implied indices

63603. Use pPager->journalOff as the effective size of the main rollback ** journal. The actual file might be larger than this in **
PAGER_JOURNALMODE_TRUNCATE or PAGER_JOURNALMODE_PERSIST. But anything ** past pPager->journalOff is off-limits to us.

63604. ***** End of pragma.h *****

63605. ** An rtree structure node.

63606. Length of z in bytes (excl. nul-term)

63607. force new CREATE statements into vacuum_db

63608. Rowid in sqlite_sequence

63609. PRAGMA

63610. FROM clause term being analyzed

63611. Array of segment iterators

63612. Write the new.* vector

63613. Bytes of zBuf[] currently used

63614. ** Include the configuration header output by 'configure' if we're using the ** autoconf-based build

63615. The new column

63616. 19

63617. True if *pRowid really is in the table

63618. Inode number

63619. Rowid used for the averages record

63620. P2

63621. 269

63622. Set up an sqlite3_backup object. sqlite3_backup.pDestDb must be set ** to 0. This is used by the implementations of sqlite3_backup_step() ** and sqlite3_backup_finish() to detect that they are being called ** from this function, not directly by the user.

63623. If the VDBE has been run even partially, then transfer the error code ** and error message from the VDBE into the main database structure. But ** if the VDBE has just been set to run but has not actually executed any ** instructions yet, leave the main database error information unchanged.

63624. ** Get a reference to pPage1 of the database file. This will ** also acquire a readlock on that file. ** ** SQLITE_OK is returned on success. If the file is not a ** well-formed database file, then SQLITE_CORRUPT is returned. ** SQLITE_BUSY is returned if the database is locked. SQLITE_NOMEM ** is returned if we run out of memory.

63625. FROM clause

63626. **comment:** ** If SQLITE_DEBUG_SORTER_THREADS is defined, this module outputs various ** messages to stderr that may be helpful in understanding the performance ** characteristics of the sorter in multi-threaded mode.
label: code-design

63627. EVIDENCE-OF: R-28594-02890 The one-byte flag at offset 0 indicating ** the b-tree page type.

63628. ** The parser calls this routine when it first sees a CREATE VIRTUAL TABLE ** statement. The module name has been parsed, but the optional list ** of parameters that follow the module name are still pending.

63629. Data space required for this record

63630. Set the EOF flag if either all synonym iterators are at EOF or an ** error has occurred.

63631. Run the progress counter just before returning.

63632. ** Search a FuncDefHash for a function with the given name. Return ** a pointer to the matching FuncDef if found, or 0 if there is no match.

63633. Enable load_extension() SQL func

63634. **comment:** FROM clause term being coded
label: code-design

63635. Started out as two integer operands

63636. Set to true if segment is appendable

63637. IN: Left hand input list

63638. b: p1<<14 | p3 (unmasked)

63639. Advance the iterator for each token in the phrase once.

63640. Min-heap used for checking cell coverage

63641. **comment:** ***** General Interfaces ***** Initialize and shutdown the page cache subsystem.
Neither of these ** functions are threadsafe.
label: code-design

63642. JSON type values

63643. ** Create a new aggregate context for p and return a pointer to ** its pMem->z element.

63644. When transitioning from TRUNCATE or PERSIST to any other journal ** mode except WAL, unless the pager is in locking_mode=exclusive mode, ** delete the journal file.

63645. ** This structure is used to pass data from sqlite3_get_table() through ** to the callback function it uses to build the result.

63646. ** This is a public wrapper for the winMbcToUtf8() function.

63647. Flag parameters, such as SF_Distinct

63648. ** Create the mutex and shared memory used for locking in the file ** descriptor pFile

63649. ** This function is called from within a pre-update callback to retrieve ** a field of the row currently being updated or inserted.

63650. f0..f7

63651. Read & merge multiple PMAs

63652. ** Add the virtual table pVTab to the array sqlite3.aVTrans[]. Space should ** have already been reserved using growVTrans().

63653. ** Special setup for VxWorks

63654. ** Return true if z[] begins with 4 (or more) hexadecimal digits

63655. FROM clause: A list of all tables to be scanned

63656. Step 4. Try to recycle a page.

63657. Number of open transactions (read + write)

63658. Or if it is a non-periodic sample. Add it in this case too.

63659. Locks are within range

63660. 580

63661. ** Decode a segment-data rowid from the %_data table. This function is ** the opposite of macro FTS5_SEGMENT_ROWID().

63662. Page flags

63663. The table to insert into. aka TABLE

63664. ** Test for access permissions. Return true if the requested permission ** is available, or false otherwise.

63665. Invoke this callback routine

63666. Ran out of input before finding the table name. Return NULL.

63667. ** Truncate the file. *** If the journal file is already on disk, truncate it there. Or, if it ** is still in main memory but is being truncated to zero bytes in size, ** ignore

63668. Call the xFindFunction method on the virtual table implementation ** to see if the implementation wants to overload this function

63669. Sometimes the code for a subquery will be generated more than ** once, if the subquery is part of the WHERE clause in a LEFT JOIN, ** for example. In that case, do not regenerate the code to manifest ** a view or the co-routine to implement a view. The first instance ** is sufficient, though the subroutine to manifest the view does need ** to be invoked again.

63670. The database we are looking in

63671. Read the schema cookie from the database. If it does not match the ** value stored as part of the in-memory schema representation, ** set Parse.rc to SQLITE_SCHEMA.

63672. True if we use the index only

63673. **comment:** There must be at least one argument passed to this function (otherwise ** the non-overloaded version would have been called instead of this one).
label: code-design

63674. YYFALLBACK

63675. #define SQLITE_IGNORE 2 // Also used by sqlite3_authorizer() callback

63676. Number of entries in apMem[]

63677. ** Each sample stored in the sqlite_stat3 table is represented in memory ** using a structure of this type. See documentation at the top of the ** analyze.c source file for additional information.

63678. ** Include standard header files as necessary

63679. At this point variables should be set as follows: ** nPayload Total payload size in bytes ** pPayload Begin writing payload here ** spaceLeft Space available at pPayload. If nPayload>spaceLeft, ** that means content must spill into overflow pages. ** *pnSize Size of the local cell (not counting overflow pages) ** pPrior Where to write the pgno of the first overflow page ** ** Use a call to btreeParseCellPtr() to verify that the values above ** were computed correctly.

63680. Thread-sanitizer reports that the following is an unsafe read, ** as some other thread may be in the process of updating the value ** of the aReadMark[] slot. The assumption here is that if that is ** happening, the other client may only be increasing the value, ** not decreasing it. So assuming either that either the "old" or ** "new" version of the value is read, and not some arbitrary value ** that would never be written by a real client, things are still ** safe.

63681. Do an integrity check on each database file

63682. If the DELETE has generated immediate foreign key constraint ** violations, halt the VDBE and return an error at this point, before ** any modifications to the schema are made. This is because statement ** transactions are not able to rollback schema changes. *** If the SQLITE_DeferFKs flag is set, then this is not required, as ** the statement transaction will not be rolled back even if FK ** constraints are violated.

63683. ** Change the value of the opcode, or P1, P2, P3, or P5 operands ** for a specific instruction.

63684. Transformation type code

63685. **comment:** Magic number 1 is the WAL_CKPT_LOCK lock. Preventing SQLite from ** taking this lock also prevents any checkpoints from occurring. ** todo: really, it's not clear why this might occur, as ** wal_autocheckpoint ought to be turned off.
label: code-design

63686. Close the transaction, if one was opened.

63687. ** Return code from the parse-tree walking primitives and their ** callbacks.

63688. ** Discard the entire contents of the in-memory page-cache.

63689. There is no INDEXED BY clause. Create a fake Index object in local ** variable sPk to represent the rowid primary key index. Make this ** fake index the first in a chain of Index objects with all of the real ** indices to follow

63690. 2147418112

63691. Allocate space for the new sqlite3_shm object. Also speculatively ** allocate space for a new winShmNode and filename.

63692. Total bytes of space to allocate

63693. **comment:** Variable DUMMY1 is initialized to a negative value above. Elsewhere ** in the FTS code the variable that the third argument to xNext points to ** is initialized to zero before the first (*but not necessarily ** subsequent*) call to xNext(). This is done for a particular application ** that needs to know whether or not the tokenizer is being used for ** snippet generation or for some other purpose. *** Extreme care is required when writing code to depend on this ** initialization. It is not a documented part of the tokenizer interface. ** If a tokenizer is used directly by any code outside of FTS, this ** convention might not be respected.
label: code-design

63694. The attempt to extend the existing mapping failed. Free it.

63695. Number of deferred fk violations

63696. The operator

63697. IN/OUT: Left input list

63698. Timezone was set explicitly
63699. Number of backtraces on this alloc
63700. Figure out how many savepoints will still be active after this ** operation. Store this value in nNew. Then free resources associated ** with any savepoints that are destroyed by this operation.
63701. ***** Begin file vacuum.c *****
63702. ** This function does a "phrase" merge of two doclists. In a phrase merge, ** the output contains a copy of each position from the right-hand input ** doclist for which there is a position in the left-hand input doclist ** exactly nDist tokens before it. *** If the docids in the input doclists are sorted in ascending order, ** parameter bDescDoclist should be false. If they are sorted in ascending ** order, it should be passed a non-zero value. *** The right-hand input doclist is overwritten by this function.
63703. ** CAPI3REF: Configure an auto-checkpoint ** METHOD: sqlite3 *** ^The [sqlite3_wal_autocheckpoint(D,N)] is a wrapper around ** [sqlite3_wal_hook()] that causes any database on [database connection] D ** to automatically [checkpoint] ** after committing a transaction if there are N or ** more frames in the [write-ahead log] file. ^Passing zero or ** a negative value as the nFrame parameter disables automatic ** checkpoints entirely. *** ^The callback registered by this function replaces any existing callback ** registered using [sqlite3_wal_hook()]. ^Likewise, registering a callback ** using [sqlite3_wal_hook()] disables the automatic checkpoint mechanism ** configured by this function. *** ^The [wal_autocheckpoint pragma] can be used to invoke this interface ** from SQL. ** ** ^Checkpoints initiated by this mechanism are ** [sqlite3_wal_checkpoint_v2|PASSIVE]. *** ^Every new [database connection] defaults to having the auto-checkpoint ** enabled with a threshold of 1000 or [SQLITE_DEFAULT_WAL_AUTOCHECKPOINT] ** pages. The use of this interface ** is only necessary if the default setting is found to be suboptimal ** for a particular application.
63704. Set this flag to 1
63705. Original value of pNew->u.btree.nEq
63706. Database is closed
63707. outer
63708. ** Link the chunk at mem3.aPool[i] so that is on the list rooted ** at *pRoot.
63709. Index in sParse.aNode[] of current row
63710. Specific token to advance
63711. ** Set the title string for subsequent allocations.
63712. ** Names of locks. This routine is used to provide debugging output and is not ** a part of an ordinary build.
63713. Index added to make the name unique
63714. OUT: *ppToken is the token text
63715. Context object for function invocation
63716. Copy of idxNum search parameter
63717. Second and subsequent calls get processed here
63718. For looping through SELECT statements
63719. Variable 'dist' stores the number of tokens read since the most ** recent TK_DOT or TK_ON. This means that when a WHEN, FOR or BEGIN ** token is read and 'dist' equals 2, the condition stated above ** to be met. *** Note that ON cannot be a database, table or column name, so ** there is no need to worry about syntax like ** "CREATE TRIGGER ... ON ON.ON BEGIN ..." etc.
63720. ** This function is called when the root page of a b-tree structure is ** overfull (has one or more overflow pages). *** A new child page is allocated and the contents of the current root ** page, including overflow cells, are copied into the child. The root ** page is then overwritten to make it an empty page with the right-child ** pointer pointing to the new page. *** Before returning, all pointer-map entries corresponding to pages ** that the new child-page now contains pointers to are updated. The ** entry corresponding to the new right-child pointer of the root ** page is also updated. *** If successful, *ppChild is set to contain a reference to the child ** page and SQLITE_OK is returned. In this case the caller is required ** to call releasePage() on *ppChild exactly once. If an error occurs, ** an error code is returned and *ppChild is set to 0.
63721. Check for authorization to create an index.
63722. Type field
63723. Total number of rows in FTS table
63724. Used when p4type is P4_EXPR
63725. Uses OS features not supported on host
63726. The root page of the PRIMARY KEY is the table root page
63727. Store result in reg[P2] rather than jump
63728. skip leading spaces
63729. ** Walk the expression tree pExpr and increase the aggregate function ** depth (the Expr.op2 field) by N on every TK_AGG_FUNCTION node. ** This needs to occur when copying a TK_AGG_FUNCTION node from an ** outer query into an inner subquery. *** incrAggFunctionDepth(pExpr,n) is the main routine. incrAggDepth(..) ** is a helper function - a callback for the tree walker.
63730. EVIDENCE-OF: R-59615-42828 A value of 10 (0x0a) means the page is a ** leaf index b-tree page.
63731. **comment:** ** Flags passed as the third argument to sqlite3BtreeCursor(). *** For read-only cursors the wrFlag argument is always zero. For read-write ** cursors it may be set to either (BTREE_WRCSR|BTREE_FORDELETE) or just ** (BTREE_WRCSR). If the BTREE_FORDELETE bit is set, then the cursor will ** only be used by SQLite for the following: *** to seek to and then delete specific entries, and/or *** to read values that will be used to create keys that other ** BTREE_FORDELETE cursors will seek to and delete. *** The BTREE_FORDELETE flag is an optimization hint. It is not used by ** by this, the native b-tree engine of SQLite, but it is available to ** alternative storage engines that might be substituted in place of this ** b-tree system. For alternative storage engines in which a delete of ** the main table row automatically deletes corresponding index rows, ** the FORDELETE flag hint allows those alternative storage engines to ** skip a lot of work. Namely: FORDELETE cursors may treat all SEEK ** and DELETE operations as no-ops, and any READ operation against a ** FORDELETE cursor may return a null row: 0x01 0x00.
label: code-design
63732. If a transient index is required, create it by calling ** sqlite3BtreeCreateTable() with the BTREE_BLOBKEY flag before ** opening it. If a transient table is required, just use the ** automatically created table with root-page 1 (an BLOB_INTKEY table).
63733. Buffer size
63734. ** Add all WhereLoop objects for all tables
63735. ***** NOTES ON THE DESIGN OF THE PAGER ***** This comment block describes invariants that hold when using a rollback ** journal. These invariants do not apply for journal_mode=WAL, ** journal_mode=MEMORY, or journal_mode=OFF. *** Within this comment block, a page is deemed to have been synced ** automatically as soon as it is written when PRAGMA synchronous=OFF. ** Otherwise, the page is not synced until the xSync method of the VFS ** is called successfully on the file containing the page. *** Definition: A page of the database file is said to be "overwriteable" if ** one or more of the following are true about the page: *** (a) The original content of the page as it was at the beginning of ** the transaction has been written into the rollback journal and ** synced. *** (b) The page was a freelist leaf page at the start of the transaction. *** (c) The page number is greater than the largest page that existed in ** the database file at the start of the transaction. *** (1) A page of the database file is never overwritten unless one of the ** following are true: *** (a) The page and all other pages on the same sector are overwriteable. *** (b) The atomic page write optimization is enabled, and the entire ** transaction other than the update of the transaction sequence ** number consists of a single page change. *** (2) The content of a page written into the rollback journal exactly matches ** both the content in the database when the rollback journal was written ** and the content in the database at the beginning of the current ** transaction. *** (3) Writes to the database file are an integer multiple of the page size ** in length and are aligned on a page boundary. *** (4) Reads from the database file are either aligned on a page boundary and ** an integer multiple of the page size in length or are taken from the ** first 100 bytes of the database file. *** (5) All writes to the database file are synced prior to the rollback journal ** being deleted, truncated, or zeroed. *** (6) If a master journal file is used, then all writes to the database file ** are synced prior to the master journal being deleted. *** Definition: Two databases (or the same database at two points in time) ** are said to be "logically equivalent" if they give the same answer to ** all queries. Note in particular the content of freelist leaf ** pages can be changed arbitrarily without affecting the logical equivalence ** of the database. *** (7) At any time, if any subset, including the empty set and the total set, ** of the unsynced changes to a rollback journal are removed and the ** journal is rolled back, the resulting database file will be logically ** equivalent to the database file at the beginning of the transaction. *** (8) When a transaction is rolled back, the xTruncate method of the VFS ** is called to restore the database file to the same size it was at ** the beginning of the transaction. (In some VFSes, the xTruncate ** method is a no-op, but that does not change the fact the SQLite will ** invoke it.) *** (9) Whenever the database file is modified, at least one bit in the range ** of bytes from 24 through 39 inclusive will be changed prior to releasing ** the EXCLUSIVE lock, thus signaling other connections on the same ** database to flush their caches. *** (10) The pattern of bits in bytes 24 through 39 shall not repeat in less ** than one billion transactions. *** (11) A database file is well-formed at the beginning and at

the conclusion ** of every transaction. *** (12) An EXCLUSIVE lock is held on the database file when writing to ** the database file. *** (13) A SHARED lock is held on the database file while reading any ** content out of the database file. ***

63736. ** Register a new collation sequence with the database handle db.

63737. Current node image

63738. Decode the position list tail at the start of the page

63739. Open the sqlite_stat1 table on this cursor

63740. ** Return status data for a single loop within query pStmt.

63741. Figure out how many records are in the journal. Abort early if ** the journal is empty.

63742. Affinity to use

63743. ** The second argument to this function contains the text of an SQL statement ** that returns a single integer value. The statement is compiled and executed ** using database connection db. If successful, the integer value returned ** is written to *piVal and SQLITE_OK returned. Otherwise, an SQLite error ** code is returned and the value of *piVal after returning is not defined.

63744. First argument passed to xConflict

63745. If the secure_delete option is enabled, then ** always fully overwrite deleted information with zeros.

63746. ** When a sub-program is executed (OP_Program), a structure of this type ** is allocated to store the current value of the program counter, as ** well as the current memory cell array and various other frame specific ** values stored in the Vdbe struct. When the sub-program is finished, ** these values are copied back to the Vdbe from the VdbeFrame structure, ** restoring the state of the VM to as it was before the sub-program ** began executing. *** The memory for a VdbeFrame object is allocated and managed by a memory ** cell in the parent (calling) frame. When the memory cell is deleted or ** overwritten, the VdbeFrame object is not freed immediately. Instead, it ** is linked into the Vdbe.pDelFrame list. The contents of the Vdbe.pDelFrame ** list is deleted when the VM is reset in VdbeHalt(). The reason for doing ** this instead of deleting the VdbeFrame immediately is to avoid recursive ** calls to sqlite3VdbeMemRelease() when the memory cells belonging to the ** child frame are released. *** The currently executing frame is stored in Vdbe.pFrame. Vdbe.pFrame is ** set to NULL if the currently executing frame is the main program.

63747. WHEN clause for the trigger

63748. No WHERE clause

63749. Can't reestablish the shared lock. Sqlite can't deal, this is ** a critical I/O error

63750. True if PAGER_GET_NOCONTENT is set

63751. ** Deprecated external interface. Internal/core SQLite code ** should call sqlite3TransferBindings. *** It is misuse to call this routine with statements from different ** database connections. But as this is a deprecated interface, we ** will not bother to check for that condition. *** If the two statements contain a different number of bindings, then ** an SQLITE_ERROR is returned. Nothing else can go wrong, so otherwise ** SQLITE_OK is returned.

63752. 255

63753. Bytes of data available in buffer

63754. Put error message here if not 0

63755. The following are used by the fts3_snippet.c module.

63756. For each expression in the list

63757. **comment:** ** The macro MEMDB is true if we are dealing with an in-memory database. ** We do this as a macro so that if the SQLITE OMIT_MEMORYDB macro is set, ** the value of MEMDB will be a constant and the compiler will optimize ** out code that would never execute.

label: code-design

63758. ** Recommended number of samples for sqlite_stat4

63759. Memory registers holding LIMIT & OFFSET counters

63760. (pKey1/nKey1) is a number or a null

63761. No need to log a failure to lock

63762. Min-heap used for analyzing cell coverage

63763. Index for p->aIndex[]

63764. Load table details if required

63765. ** Given table pTab, return a list of all the triggers attached to ** the table. The list is connected by Trigger.pNext pointers. *** All of the triggers on pTab that are in the same database as pTab ** are already attached to pTab->pTrigger. But there might be additional ** triggers on pTab in the TEMP schema. This routine prepends all ** TEMP triggers on pTab to the beginning of the pTab->pTrigger list ** and returns the combined list. *** To state it another way: This routine returns a list of all triggers ** that fire off of pTab. The list will include any TEMP triggers on ** pTab as well as the triggers listed in pTab->pTrigger.

63766. source and destination must both be WITHOUT ROWID or not

63767. **comment:** Loop through all %_segdir entries for segments in this index with ** levels equal to or greater than iAbsLevel. As each entry is visited, ** updated it to set (level = -1) and (idx = N), where N is 0 for the ** oldest segment in the range, 1 for the next oldest, and so on. *** In other words, move all segments being promoted to level -1, ** setting the "idx" fields as appropriate to keep them in the same ** order. The contents of level -1 (which is never used, except ** transiently here), will be moved back to level iAbsLevel below.

label: code-design

63768. ** Return true if the iterator is at EOF or if an error has occurred. ** False otherwise.

63769. Previously obtained value for P4

63770. **comment:** Loop through the tables in the main database. For each, do ** an "INSERT INTO vacuum_db.xxx SELECT * FROM main.xxx;" to copy ** the contents to the temporary database.

label: code-design

63771. The offset for ReadFile.

63772. CURTYPE_SORTER. Sorter object

63773. Elements in aOp[]

63774. pPager->state = PAGER_UNLOCK;

63775. **comment:** Each page cache (or PCache) belongs to a PGroup. A PGroup is a set ** of one or more PCaches that are able to recycle each other's unpinned ** pages when they are under memory pressure. A PGroup is an instance of ** the following object. *** This page cache implementation works in one of two modes: ** ** (1) Every PCache is the sole member of its own PGroup. There is ** one PGroup per PCache. *** (2) There is a single global PGroup that all PCaches are a member ** of. *** Mode 1 uses more memory (since PCache instances are not able to rob ** unused pages from other PCaches) but it also operates without a mutex, ** and is therefore often faster. Mode 2 requires a mutex in order to be ** threadsafe, but recycles pages more efficiently. *** For mode (1), PGroup.mutex is NULL. For mode (2) there is only a single ** PGroup which is the pcache1.grp global variable and its mutex is ** SQLITE_MUTEX_STATIC_LRU.

label: code-design

63776. Column names corresponding to IDLIST.

63777. ** CAPI3REF: Set a table filter on a Session Object. *** The second argument (xFilter) is the "filter callback". For changes to rows ** in tables that are not attached to the Session object, the filter is called ** to determine whether changes to the table's rows should be tracked or not. ** If xFilter returns 0, changes is not tracked. Note that once a table is ** attached, xFilter will not be called again.

63778. ** Scan the column type name zType (length nType) and return the ** associated affinity type. *** This routine does a case-independent search of zType for the ** substrings in the following table. If one of the substrings is ** found, the corresponding affinity is returned. If zType contains ** more than one of the substrings, entries toward the top of ** the table take priority. For example, if zType is 'BLOBINT', ** SQLITE_AFF_INTEGER is returned. *** Substring | Affinity ** ----- ** INT | SQLITE_AFF_INTEGER ** CHAR | SQLITE_AFF_TEXT ** CLOB | SQLITE_AFF_TEXT ** TEXT | SQLITE_AFF_TEXT ** BLOB' | SQLITE_AFF_BLOB ** REAL' | SQLITE_AFF_REAL ** FLOA' | SQLITE_AFF_REAL ** DOUB' | SQLITE_AFF_REAL *** If none of the substrings in the above table are found, ** SQLITE_AFF_NUMERIC is returned.

63779. Number of fields in pRec

63780. ** Given the list of WhereLoop objects at pWInfo->pLoops, this routine ** attempts to find the lowest cost path that visits each WhereLoop ** once. This path is then loaded into the pWInfo->a[].pWLoop fields. *** Assume that the total number of output rows that will need to be sorted ** will be nRowEst (in the 10*log2 representation). Or, ignore sorting ** costs if nRowEst==0. *** Return SQLITE_OK on success or SQLITE_NOMEM of a memory allocation ** error occurs.

63781. Parse array

63782. ** This is a helper function used by sessionMergeUpdate(). *** When this function is called, both *paOne and *paTwo point to a value ** within a change record. Before it returns, both have been advanced so ** as to point to the next value in the record. *** If, when this function is called, *paTwo points to a valid value (i.e. ** *paTwo[0] is not 0x00 - the "no value" placeholder), a copy of the *paTwo ** pointer is returned and *pnVal is set to the number of bytes in the ** serialized value. Otherwise, a copy of *paOne is returned and *pnVal ** set to the number of bytes in the value at *paOne. If *paOne points ** to the "no value" placeholder, *pnVal is set to 1. In other words: *** if(*paTwo is valid) return *paTwo; ** return *paOne; **

63783. Value for "end_block" field

63784. a column type code

63785. Store result as data with an automatic rowid

63786. Initialize the structure. The sqlite3_index_info structure contains ** many fields that are declared "const" to prevent xBestIndex from ** changing them. We have to do some funky casting in order to ** initialize those fields.

63787. ** The following set of routines are used to disable the simulated ** I/O error mechanism. These routines are used to avoid simulated ** errors in places where we do not care about errors. *** Unless -DSQLITE_TEST=1 is used, these routines are all no-ops ** and generate no code.

63788. Drop all SQLITE_MASTER table and index entries that refer to the ** table. The program name loops through the master table and deletes ** every row that refers to a table of the same name as the one being ** dropped. Triggers are handled separately because a trigger can be ** created in the temp database that refers to a table in another ** database.

63789. ** Locate a user function given a name, a number of arguments and a flag ** indicating whether the function prefers UTF-16 over UTF-8. Return a ** pointer to the FuncDef structure that defines that function, or return ** NULL if the function does not exist. *** If the createFlag argument is true, then a new (blank) FuncDef ** structure is created and linked into the "db" structure if a ** no matching function previously existed. *** If nArg is -2, then the first valid function found is returned. A ** function is valid if xSFunc is non-zero. The nArg==(-2) ** case is used to see if zName is a valid function name for some number ** of arguments. If nArg is -2, then createFlag must be 0. *** If createFlag is false, then a function with the required name and ** number of arguments may be returned even if the eTextRep flag does not ** match that requested.

63790. True if aOverflow is valid

63791. Table on the left with matching column name

63792. Text encoding used by this database

63793. Make the new connection a child of the unixShmNode

63794. If the VM did not run to completion or if it encountered an ** error, then it might not have been halted properly. So halt ** it now.

63795. EVIDENCE-OF: R-02776-14802 The cell pointer array consists of K 2-byte ** integer offsets to the cell contents.

63796. True exponent is either not used or is well-formed

63797. OUT: SQLITE_INSERT, DELETE or UPDATE

63798. Added ???

63799. ** 2006 June 7 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This header file defines the SQLite interface for use by ** shared libraries that want to be imported as extensions into ** an SQLite instance. Shared libraries that intend to be loaded ** as extensions by SQLite should #include this file instead of ** sqlite3.h.

63800. Number of references to columns in other FROM clauses

63801. ** Allowed values for Select.selFlags. The "SF" prefix stands for ** "Select Flag". *** Value constraints (all checked via assert()) ** SF_HasAgg == NC_HasAgg ** SF_MinMaxAgg == NC_MinMaxAgg == SQLITE_FUNC_MINMAX ** SF_FixedLimit == WHERE_USE_LIMIT

63802. If there is a doclist-index, check that it looks right.

63803. **comment:** "[...]" immediately follows the "*". We have to do a slow ** recursive search in this case, but it is an unusual case.
label: code-design

63804. ** Return the column of index pIdx that corresponds to table ** column iCol. Return -1 if not found.

63805. ** Memory map or remap the file opened by file-descriptor pFd (if the file ** is already mapped, the existing mapping is replaced by the new). Or, if ** there already exists a mapping for this file, and there are still ** outstanding xFetch() references to it, this function is a no-op. *** If parameter nByte is non-negative, then it is the requested size of ** the mapping to create. Otherwise, if nByte is less than zero, then the ** requested size is the size of the file on disk. The actual size of the ** created mapping is either the requested size or the value configured ** using SQLITE_FCNTL_MMAP_LIMIT, whichever is smaller. *** ** SQLITE_OK is returned if no error occurs (even if the mapping is not ** recreated as a result of outstanding references) or an SQLite error ** code otherwise.

63806. ** Populate the Fts5Config.zContentExprlist string.

63807. Additional methods may be added in future releases

63808. The comparison operator

63809. Connection that caused SQLITE_LOCKED

63810. Read the wal-index header.

63811. ** Return true if the parser passed as the first argument is being ** used to code a trigger that is really a "SET NULL" action belonging ** to trigger pFKey.

63812. Number of errors reported

63813. Subroutine for resetting the accumulator

63814. ** Set the value if the iSrcLine field for the previously coded instruction.

63815. ** Free any overflow pages associated with the given Cell. Write the ** local Cell size (the number of bytes on the original page, omitting ** overflow) into *pnSize.

63816. ** This function searches pList for an entry that matches the iCol-th column ** of index pIdx. *** If such an expression is found, its index in pList->a[] is returned. If ** no expression is found, -1 is returned.

63817. Usable size of mapped region

63818. compress=? parameter (or NULL)

63819. Initialization context

63820. The WHERE clause. May be NULL

63821. The flags specified at open()

63822. FTS5_AND node to advance

63823. Value returned by conflict handler

63824. Populate the coverage-checking heap for leaf pages

63825. SQLITE_OS_WIN_THREADS

63826. ***** Begin file where.c *****

63827. Insert an element into the hash table pH. The key is pKey ** and the data is "data". *** If no element exists with a matching key, then a new ** element is created and NULL is returned. *** If another element already exists with the same key, then the ** new data replaces the old data and the old data is returned. ** The key is not copied in this instance. If a malloc fails, then ** the new data is returned and the hash table is unchanged. *** If the "data" parameter to this function is NULL, then the ** element corresponding to "key" is removed from the hash table.

63828. Verifies FORDELETE and AUXDELETE flags

63829. First try to append the term to the current node. Return early if ** this is possible.

63830. ** Return a section of the pPage->aData to the freelist. ** The first byte of the new free block is pPage->aData[iStart] ** and the size of the block is iSize bytes. *** Adjacent freeblocks are coalesced. *** Note that even though the freeblock list was checked by btreeInitPage(), ** that routine will not detect overlap between cells or freeblocks. Nor ** does it detect cells or freeblocks that encroach into the reserved bytes ** at the end of the page. So do additional corruption checks inside this ** routine and return SQLITE_CORRUPT if any problems are found.

63831. Flags to sqlite3BtreeOpen()

63832. tcons ::= CONSTRAINT nm

63833. If we compiled with the SQLITE_NO_SYNC flag, then syncing is a ** no-op

63834. Find an existing tokenizer

63835. defined(SQLITE_WIN32_HAS_ANSI)

63836. EVIDENCE-OF: R-61275-35157 The SQLITE_CONFIG_MEMSTATUS option takes ** single argument of type int, interpreted as a boolean, which enables ** or disables the collection of memory allocation statistics.

63837. ** Read data from a blob handle.

63838. xSleep
63839. ** Return the maximum height of any expression tree referenced ** by the select statement passed as an argument.
63840. Info about locks on this inode
63841. The WhereClause to be initialized
63842. mem1.flags = 0; // Will be initialized by sqlite3VdbeSerialGet()
63843. Return code for this function
63844. Allocate and populate the context object.
63845. The i-th index expression
63846. **** Begin dot-file Locking
***** The dotfile locking implementation uses the existence of separate lock ** files (really a directory) to control access to the database. This works ** on just about every filesystem imaginable. But there are serious downsides: *** (1) There is zero concurrency. A single reader blocks all other ** connections from reading or writing the database. *** (2) An application crash or power loss can leave stale lock files ** sitting around that need to be cleared manually. *** Nevertheless, a dotlock is an appropriate locking mode for use if no ** other locking strategy is available. *** Dotfile locking works by creating a subdirectory in the same directory as ** the database and with the same name but with a ".lock" extension added. ** The existence of a lock directory implies an EXCLUSIVE lock. All other ** lock types (SHARED, RESERVED, PENDING) are mapped into EXCLUSIVE.
63847. Search the hash table or tables for an entry matching page number ** pgno. Each iteration of the following for() loop searches one ** hash table (each hash table indexes up to HASHTABLE_NPAGE frames). *** This code might run concurrently to the code in walIndexAppend() ** that adds entries to the wal-index (and possibly to this hash ** table). This means the value just read from the hash ** slot (aHash[iKey]) may have been added before or after the ** current read transaction was opened. Values added after the ** read transaction was opened may have been written incorrectly - ** i.e. these slots may contain garbage data. However, we assume ** that any slots written before the current read transaction was ** opened remain unmodified. *** For the reasons above, the if(..) condition featured in the inner ** loop of the following block is more stringent than would be required ** if we had exclusive access to the hash-table: *** (aPgno[iFrame]==pgno): ** This condition filters out normal hash-table collisions. *** (iFrame<=iLast): ** This condition filters out entries that were added to the hash ** table after the current read-transaction had started.
63848. Snippet iterator
63849. ** Return the number of dirty pages currently in the cache, as a percentage ** of the configured cache size.
63850. Enable recursive triggers
63851. Iterator used to gather data from index
63852. Don't do any authorization checks if the database is initialising ** or if the parser is being invoked from within sqlite3_declare_vtab.
63853. The parser's stack
63854. Absolute level number to work on
63855. ***** End of vacuum.c *****
63856. Content of this chunk
63857. Address of OP_Explain (or 0)
63858. ***** The following routines are the only publically visible identifiers in this ** file. Call the following routines in order to register the various SQL ** functions and the virtual table implemented by this file.*****
63859. Last field of the record
63860. Est. number of distinct keys less than this sample
63861. ** The following is guaranteed when this function is called: *** a) the WRITER lock is held, ** b) the entire log file has been checkpointed, and ** c) any existing readers are reading exclusively from the database ** file - there are no readers that may attempt to read a frame from ** the log file. *** This function updates the shared-memory structures so that the next ** client to write to the database (which may be this one) does so by ** writing frames into the start of the log file. *** The value of parameter salt1 is used as the aSalt[1] value in the ** new wal-index header. It should be passed a pseudo-random value (i.e. ** one obtained from sqlite3_randomness()).
63862. The WHERE clause to be searched
63863. Alternate addrEofA if B is uninitialized
63864. 750
63865. The following assert statements check that the binary search code ** above found the right answer. This block serves no purpose other ** than to invoke the asserts.
63866. failed to open/create the lock directory
63867. Index used to optimize scan, or NULL
63868. ** This function does all the work for both the xConnect and xCreate methods. ** These tables have no persistent representation of their own, so xConnect ** and xCreate are identical operations.
63869. If the child table is the same as the parent table, then add terms ** to the WHERE clause that prevent this entry from being scanned. ** The added WHERE clause terms are like this: *** \$current_rowid!=rowid ** NOT(\$current_a==a AND \$current_b==b AND ...) *** The first form is used for rowid tables. The second form is used ** for WITHOUT ROWID tables. In the second form, the primary key is ** (a,b,...)
63870. CREATE VIEW name AS... without an argument list. Construct ** the column names from the SELECT statement that defines the view.
63871. ** Sync the journal. In other words, make sure all the pages that have ** been written to the journal have actually reached the surface of the ** disk and can be restored in the event of a hot-journal rollback. *** If the Pager.noSync flag is set, then this function is a no-op. ** Otherwise, the actions required depend on the journal-mode and the ** device characteristics of the file-system, as follows: *** * If the journal file is an in-memory journal file, no action need ** be taken. ** * Otherwise, if the device does not support the SAFE_APPEND property, ** then the nRec field of the most recently written journal header ** is updated to contain the number of journal records that have ** been written following it. If the pager is operating in full-sync ** mode, then the journal file is synced before this field is updated. *** * If the device does not support the SEQUENTIAL property, then ** journal file is synced. ** * Or, in pseudo-code: *** if(NOT <in-memory journal>) { ** if(NOT SAFE_APPEND){ ** if(<full-sync mode>) xSync(<journal file>); ** <update nRec field> ** } ** if(NOT SEQUENTIAL) xSync(<journal file>); ** } *** If successful, this routine clears the PGHDR_NEED_SYNC flag of every ** page currently held in memory before returning SQLITE_OK. If an IO ** error is encountered, then the IO error code is returned to the caller.
63872. ** Obtain or release the mutex needed to access global data structures.
63873. ** Open a new database handle.
63874. Number of sub-vdbes seen so far
63875. The antipenultimate character of the master journal name must ** be "9" to avoid name collisions when using 8+3 filenames.
63876. If the blob handle is not open at this point, open it and seek ** to the requested entry.
63877. ** Convert a Microsoft Unicode string to UTF-8. ** Space to hold the returned string is obtained from sqlite3_malloc().
63878. Close the open database handle and VFS object.
63879. Fsync the rollback journal before ** writing this page to the database
63880. Opcode: SCopy P1 P2 * * * Synopsis: r[P2]=r[P1] ** Make a shallow copy of register P1 into register P2. ** This instruction makes a shallow copy of the value. If the value ** is a string or blob, then the copy is only a pointer to the ** original and hence if the original changes so will the copy. ** Worse, if the original is deallocated, the copy becomes invalid. ** Thus the program must guarantee that the original will not change ** during the lifetime of the copy. Use OP_Copy to make a complete ** copy.
63881. Code generator context
63882. If the log file is not already open, but does exist in the file-system, ** it may need to be checkpointed before the connection can switch to ** rollback mode. Open it now so this can happen.
63883. Used to build up table PK declaration
63884. Number of bytes written by seekAndWrite
63885. **comment:** If ORDER BY makes no difference in the output then neither does ** DISTINCT so it can be removed too.
label: code-design
63886. ** Convert OP_Column opcodes to OP_Copy in previously generated code. ** This routine runs over generated VDBE code and translates OP_Column ** opcodes into OP_Copy when the table is being accessed via co-routine ** instead of via table lookup. ** If the bIncrRowid parameter is 0, then any OP_Rowid instructions on ** cursor iTabCur are transformed into OP_Null. Or, if bIncrRowid is non-zero, ** then each OP_Rowid is transformed into an instruction to increment the ** value stored in its output register.

63887. This is an mmap page object
63888. Jump here to continue with next row
63889. ** Add an entry to the in-memory hash table. The key is the concatenation ** of bByte and (pToken/nToken). The value is (iRowid/iCol/iPos). *** (bByte || pToken) -> (iRowid,iCol,iPos) *** Or, if iCol is negative, then the value is a delete marker.
63890. Big-endian salt values
63891. The automatic index is partial
63892. ** Free an iterator allocated by walIteratorInit().
63893. orconf ::= OR resolvetype
63894. Move the page currently at pgnoRoot to pgnoMove.
63895. zJournal
63896. True to enable core mutexing
63897. ** Malloc and Free functions
63898. First leaf page number in segment
63899. Pointer to current cell in pPage
63900. Index to search
63901. ***** Begin file delete.c *****
63902. Most recent seek operation on this cursor
63903. Populated object for xBestIndex
63904. 4
63905. ** Configuration settings for an individual database connection
63906. OUT: Number of tokens in query
63907. ** This is a public wrapper for the winUnicodeToUtf8() function.
63908. 26
63909. This index is not unique over the IN RHS columns
63910. The current term of the first nMerge entries in the array ** of Fts3SegReader objects is the same. The doclists must be merged ** and a single term returned with the merged doclist.
63911. YYNOERRORRECOVERY
63912. Integer sum
63913. Special case: WHERE terms that do not refer to any tables in the join ** (constant expressions). Evaluate each such term, and jump over all the ** generated code if the result is not true. *** Do not do this if the expression contains non-deterministic functions ** that are not within a sub-select. This is not strictly required, but ** preserves SQLite's legacy behaviour in the following two cases: ** FROM ... WHERE random()>0; -- eval random() once per row ** FROM ... WHERE (SELECT random())>0; -- eval random() once overall
63914. ***** End of opcodes.c *****
63915. Pagesize after VACUUM if >0
63916. ** If the WAL file is currently larger than nMax bytes in size, truncate ** it to exactly nMax bytes. If an error occurs while doing so, ignore it.
63917. Set the output variables.
63918. blobSeekToRow() will initialize r[1] to the desired rowid
63919. input we are tokenizing
63920. **comment:** OUT: Error message (sqlite3_malloc)
 label: code-design
63921. Base register
63922. ** Resize the hash table by doubling the number of slots.
63923. This function is only available internally, it is not part of the ** external API. It works in a similar way to sqlite3_value_text(), ** except the data returned is in the encoding specified by the second ** parameter, which must be one of SQLITE_UTF16BE, SQLITE_UTF16LE or ** SQLITE_UTF8. *** (2006-02-16:) The enc value can be or-ed with SQLITE_UTF16_ALIGNED. ** If that is the case, then the result must be aligned on an even byte ** boundary.
63924. RHS is an integer
63925. ** Return the offset of frame iFrame in the write-ahead log file, ** assuming a database page size of szPage bytes. The offset returned ** is to the start of the write-ahead log frame-header.
63926. PageWriter object
63927. True for an UPDATE or DELETE
63928. This variable holds the process id (pid) from when the xRandomness() ** method was called. If xOpen() is called from a different process id, ** indicating that a fork() has occurred, the PRNG will be reset.
63929. The list is being searched and this trunk page is the page ** to allocate, regardless of whether it has leaves.
63930. synopsis: data=r[P3@P2]
63931. Array of column names for fts3 table
63932. ** CAPI3REF: Return The Filename For A Database Connection ** METHOD: sqlite3 *** ^The sqlite3_db_filename(D,N) interface returns a pointer to a filename ** associated with database N of connection D. ^The main database file ** has the name "main". If there is no attached database N on the database ** connection D, or if database N is a temporary or in-memory database, then ** a NULL pointer is returned. *** ^The filename returned by this function is the output of the ** xFullPathname method of the [VFS]. ^In other words, the filename ** will be an absolute pathname, even if the filename used ** to open the database originally was a URI or relative pathname.
63933. Time when query started - used for profiling
63934. if non zero missing dirs will be created
63935. ** If node pLeaf is not the root of the r-tree and its pParent pointer is ** still NULL, load all ancestor nodes of pLeaf into memory and populate ** the pLeaf->pParent chain all the way up to the root node. *** This operation is required when a row is deleted (or updated - an update ** is implemented as a delete followed by an insert). SQLite provides the ** rowid of the row to delete, which can be used to find the leaf on which ** the entry resides (argument pLeaf). Once the leaf is located, this ** function is called to determine its ancestry.
63936. **comment:** Make copies of constant WHERE-clause terms in the outer query down ** inside the subquery. This can help the subquery to run more efficiently.
 label: code-design
63937. SQL name of the function.
63938. Decoded structure object
63939. ** Attempt to transform a query of the form *** SELECT count(*) FROM (SELECT x FROM t1 UNION ALL SELECT y FROM t2) *** Into this: *** SELECT (SELECT count(*) FROM t1)+(SELECT count(*) FROM t2) *** The transformation only works if all of the following are true: *** * The subquery is a UNION ALL of two or more terms ** * There is no WHERE or GROUP BY or HAVING clauses on the subqueries *** * The outer query is a simple count(*) *** ** Return TRUE if the optimization is undertaken.
63940. Offset within aMemory of next record
63941. Add the new BtShared object to the linked list sharable BtShares.
63942. OUT: Bitmask of phrases covered
63943. Analyze the right-child page of internal pages
63944. Opcode: SqlExec * * * P4 * *** Run the SQL statement or statements specified in the P4 string.
63945. Flags passed through to sqlite3_vfs.xOpen()
63946. ** This routine is called to report the final ")" that terminates ** a CREATE TABLE statement. *** The table structure that other action routines have been building ** is added to the internal hash tables, assuming no errors have ** occurred. *** An entry for the table is made in the master table on disk, unless ** this is a temporary table or db->init.busy==1. When db->init.busy==1 ** it means we are reading the sqlite_master table because we just ** connected to the database or because the sqlite_master table has ** recently changed, so the entry for this table already exists in ** the sqlite_master table. We do not want to create it again. *** If the pSelect argument is not NULL, it means that this routine ** was called to create a table generated from a ** "CREATE TABLE ... AS SELECT ..." statement. The column names of ** the new table will match the result set of the SELECT.
63947. Reprepares only called for prepare_v2() statements
63948. The name-context to resolve expressions in

63949. Docid of the row pCsr points to
63950. exponent of real numbers
63951. True if some process holds a RESERVED lock
63952. Tables that might satisfy case 1
63953. Size of pCsr->pKeyInfo in bytes
63954. ** DO NOT EDIT THIS MACHINE GENERATED FILE.
63955. Verify condition (2): If cells are moving right, update iPg ** only after iPg+1 has already been updated.
63956. The current column index in pp1
63957. For this path res2 must be exactly 0 or 1
63958. If there are no other statements currently running, then ** reset the interrupt flag. This prevents a call to sqlite3_interrupt ** from interrupting a statement that has not yet started.
63959. ** Write a set of frames to the log. The caller must hold the write-lock ** on the log file (obtained using sqlite3WalBeginWriteTransaction()).
63960. SEMI => nothing
63961. Number of segments to merge (iLevel>=0)
63962. Extra prerequisites for using this table
63963. Does the table have a rowid
63964. OUT: malloc'd buffer containing str/bw
63965. If the database file is corrupt, it is possible for the value of idx ** to be invalid here. This can only occur if a second cursor modifies ** the page while cursor pCur is holding a reference to it. Which can ** only happen if the database is corrupt in such a way as to link the ** page into more than one b-tree structure.
63966. ** Check to see if pIndex uses the collating sequence pColl. Return ** true if it does and false if it does not.
63967. Size threshold above which reanalysis is needed
63968. Context for sqlite3_result_xxx() calls
63969. create_table_args ::= LP columnlist conslist_opt RP table_options
63970. ** The rollback journal is composed of a linked list of these structures. ** ** The zChunk array is always at least 8 bytes in size - usually much more. ** Its actual size is stored in the MemJournal.nChunkSize variable.
63971. ** When compiling the test fixture or with debugging enabled (on Win32), ** this variable being set to non-zero will cause OSTRACE macros to emit ** extra diagnostic information.
63972. The function definition object
63973. ** Find a tokenizer. This is the implementation of the ** fts5_api.xFindTokenizer() method.
63974. ** Pop the parser's stack once. ** ** If there is a destructor routine associated with the token which ** is popped from the stack, then call it.
63975. When opening a zero-size database, the findInodeInfo() procedure ** writes a single byte into that file in order to work around a bug ** in the OS-X msdos filesystem. In order to avoid problems with upper ** layers, we need to report this file size as zero even though it is ** really 1. Ticket #3260.
63976. This is the (easy) common case where the entire payload fits ** on the local page. No overflow is required.
63977. The HAVING clause
63978. **comment:** Try 3 times to get the pending lock. This is needed to work ** around problems caused by indexing and/or anti-virus software on ** Windows systems.
** If you are using this code as a model for alternative VFSes, do not ** copy this retry logic. It is a hack intended for Windows only.
label: code-design
63979. Database filenames are double-zero terminated if they are not ** URIs with parameters. Hence, they can always be passed into ** sqlite3_uri_parameter().
63980. Save the positions of all other cursors open on this table. This is ** required in case any of them are holding references to an xFetch ** version of the b-tree page modified by the accessPayload call below. ** ** Note that pCsr must be open on a INTKEY table and saveCursorPosition() ** and hence saveAllCursors() cannot fail on a BTREE_INTKEY table, hence ** saveAllCursors can only return SQLITE_OK.
63981. THEN
63982. ** Return the Btree pointer identified by zDbName. Return NULL if not found.
63983. expr ::= exprlist
63984. xFindMethod
63985. Number of hash collisions remaining
63986. wait 10 sec and try the lock again
63987. ** Translate a single UTF-8 character. Return the unicode value. ** ** During translation, assume that the byte that zTerm points ** is a 0x00. ** ** Write a pointer to the next unread byte back into *pzNext. ** ** Notes On Invalid UTF-8: ** ** * This routine never allows a 7-bit character (0x00 through 0x7f) to ** be encoded as a multi-byte character. Any multi-byte character that ** attempts to encode a value between 0x00 and 0x7f is rendered as 0xffffd. ** ** * This routine never allows a UTF16 surrogate value to be encoded. ** If a multi-byte character attempts to encode a value between ** 0xd800 and 0xe000 then it is rendered as 0xffffd. ** ** * Bytes in the range of 0x80 through 0xbf which occur as the first ** byte of a character are interpreted as single-byte characters ** and rendered as themselves even though they are technically ** invalid characters. ** ** * This routine accepts over-length UTF8 encodings ** for unicode values 0x80 and greater. It does not change over-length ** encodings to 0xffffd as some systems recommend.
63988. Device number
63989. Content to be written
63990. FTS virtual table handle
63991. Transfer the FROM clause terms from the subquery into the ** outer query.
63992. ** Invoke the Vdbe coverage callback, if that callback is defined. This ** feature is used for test suite validation only and does not appear in ** production builds. ** ** M is an integer, 2 or 3, that indices how many different ways the ** branch can go. It is usually 2. "I" is the direction the branch ** goes. 0 means falls through. 1 means branch is taken. 2 means the ** second alternative branch is taken. ** ** iSrcLine is the source code line (from the __LINE__ macro) that ** generated the Vdbe instruction. This instrumentation assumes that all ** source code is in a single file (the amalgamation). Special values 1 ** and 2 for the iSrcLine parameter mean that this particular branch is ** always taken or never taken, respectively.
63993. ** The following routines are substitutes for constants SQLITE_CORRUPT, ** SQLITE_MISUSE, SQLITE_CANTOPEN, SQLITE_NOMEM and possibly other error ** constants. They serve two purposes: ** ** 1. Serve as a convenient place to set a breakpoint in a debugger ** to detect when version error conditions occurs. ** ** 2. Invoke sqlite3_log() to provide the source code location where ** a low-level error is first detected.
63994. **comment:** ** Translate a string containing an fts5vocab table type to an ** FTS5_VOCAB_XXX constant. If successful, set *peType to the output ** value and return SQLITE_OK. Otherwise, set *pzErr to an error message ** and return SQLITE_ERROR.
label: code-design
63995. ***** Begin file os_unix.c *****
63996. ** If expression pExpr is of type TK_SELECT, generate code to evaluate ** it. Return the register in which the result is stored (or, if the ** sub-select returns more than one column, the first in an array ** of registers in which the result is stored). ** ** If pExpr is not a TK_SELECT expression, return 0.
63997. EVIDENCE-OF: R-63325-48378 The SQLITE_CONFIG_PCACHE2 option takes a ** single argument which is a pointer to an sqlite3_pcachemethods2 ** object. This object specifies the interface to a custom page cache ** implementation.
63998. Register for new.* values
63999. cmd ::= create_vtab LP vtabarglist RP
64000. The state of the parser is completely contained in an instance of ** the following structure
64001. 0x50 .. 0x5F
64002. The requested data is not all available in the in-memory buffer. ** In this case, allocate space at p->aAlloc[] to copy the requested ** range into. Then return a copy of pointer p->aAlloc to the caller.
64003. Prepared statement in use by the cursor
64004. The index to be created
64005. NULL or constants 0 or 1
64006. Delete any auxdata allocations made by the VM
64007. ** CAPI3REF: Extended Result Codes ** KEYWORDS: {extended result code definitions} ** ** In its default configuration, SQLite API routines return one of 30 integer ** [result codes]. However, experience has shown that many of ** these result codes are too coarse-grained. They do not provide as ** much information about problems as programmers might like. In an effort to ** address this, newer versions of SQLite (version 3.3.8 [dateof:3.3.8] ** and later) include ** support for additional result codes that provide more detailed information ** about errors. These [extended result codes] are enabled or disabled ** on a per

database connection basis using the ** [sqlite3_extended_result_codes()] API. Or, the extended code for ** the most recent error can be obtained using ** [sqlite3_extended_errcode()].

64008. Used to iterate through tokens
64009. Label for the start of the merge algorithm
64010. 0x0c
64011. **comment:** ** The type for a callback function. ** This is legacy and deprecated. It is included for historical ** compatibility and is not documented.
 label: documentation
64012. Current row as a Stat4Sample
64013. OUT: The page number of the next page
64014. Return open Btree* here
64015. 1520
64016. Variable tname now contains the token that is the old table-name ** in the CREATE TRIGGER statement.
64017. Invoked by sqlite3FaultSim()
64018. ** Given 1 to 3 identifiers preceding the JOIN keyword, determine the ** type of join. Return an integer constant that expresses that type ** in terms of the following bit values: ** ** JT_INNER ** JT_CROSS ** JT_OUTER ** JT_NATURAL ** JT_LEFT ** JT_RIGHT ** ** A full outer join is the combination of JT_LEFT and JT_RIGHT. ** ** If an illegal or unsupported join type is seen, then still return ** a join type, but put an error in the pParse structure.
64019. ***** Non-locking sqlite3_file methods
***** The next division contains implementations for all methods of the ** sqlite3_file object other than the locking methods.
The locking ** methods were defined in divisions above (one locking method per ** division). Those methods that are common to all locking modes ** are gather together into this division.
64020. For each expression in pEList that is a copy of an expression in ** the ORDER BY clause (pSort->pOrderBy), set the associated ** iOrderByCol value to one more than the index of the ORDER BY ** expression within the sort-key that pushOntoSorter() will generate. ** This allows the pEList field to be omitted from the sorted record, ** saving space and CPU cycles.
64021. Code the current SELECT into temporary table "tab2"
64022. Pad transactions out to the next sector
64023. ** We already know that pExpr is a binary operator where both operands are ** column references. This routine checks to see if pExpr is an equivalence ** relation: ** 1. The SQLITE_Transitive optimization must be enabled ** 2. Must be either an == or an IS operator ** 3. Not originating in the ON clause of an OUTER JOIN ** 4. The affinities of A and B must be compatible ** 5a. Both operands use the same collating sequence OR ** 5b. The overall collating sequence is BINARY ** If this routine returns TRUE, that means that the RHS can be substituted ** for the LHS anywhere else in the WHERE clause where the LHS column occurs. ** This is an optimization. No harm comes from returning 0. But if 1 is ** returned when it should not be, then incorrect answers might result.
64024. If we cannot open the rollback journal file in order to see if ** it has a zero header, that might be due to an I/O error, or ** it might be due to the race condition described above and in ** ticket #3883. Either way, assume that the journal is hot. ** This might be a false positive. But if it is, then the ** automatic journal playback and recovery mechanism will deal ** with it under an EXCLUSIVE lock where we do not need to ** worry so much with race conditions.
64025. ** Get the RTreeNode for the search point with the lowest score.
64026. ** Implementation of the json_object(NAME,VALUE,...) function. Return a JSON ** object that contains all name/value given in arguments. Or if any name ** is not a string or if any value is a BLOB, throw an error.
64027. Smallest sample larger than or equal to pRec
64028. ** A cell with the same content as pCell has just been inserted into ** the node pNode. This function updates the bounding box cells in ** all ancestor elements.
64029. **comment:** TODO: Could use temporary registers here. Also could attempt to ** avoid copying the contents of the rowid register.
 label: code-design
64030. ** The default number of frames to accumulate in the log file before ** checkpointing the database in WAL mode.
64031. ** Initialize a preallocated WhereClause structure.
64032. Error if integer primary key greater than this
64033. Digits
64034. Reparse everything to update our internal data structures
64035. Conflict resolution algorithm
64036. SAVEPOINT
64037. 325
64038. 1030
64039. value of function parameters
64040. OUT: Database page number for frame
64041. Obtain a reference to the leaf node that contains the entry ** about to be deleted.
64042. aMI[iCol*3 + 1] = Number of occurrences ** aMI[iCol*3 + 2] = Number of rows containing at least one instance
64043. Stop accumulating errors when this reaches zero
64044. ** CAPI3REF: Free Memory Used By A Database Connection ** METHOD: sqlite3 *** ^The sqlite3_db_release_memory(D) interface attempts to free as much heap ** memory as possible from database connection D. Unlike the ** [sqlite3_release_memory()] interface, this interface is in effect even ** when the [SQLITE_ENABLE_MEMORY_MANAGEMENT] compile-time option is ** omitted. ** ** See also: [sqlite3_release_memory()]
64045. **comment:** ** Append a term and (optionally) doclist to the FTS segment node currently ** stored in blob *pNode. The node need not contain any terms, but the ** header must be written before this function is called. ** ** A node header is a single 0x00 byte for a leaf node, or a height varint ** followed by the left-hand-child varint for an internal node. ** ** The term to be appended is passed via arguments zTerm/nTerm. For a ** leaf node, the doclist is passed as aDoclist/nDoclist. For an internal ** node, both aDoclist and nDoclist must be passed 0. ** ** If the size of the value in blob pPrev is zero, then this is the first ** term written to the node. Otherwise, pPrev contains a copy of the ** previous term. Before this function returns, it is updated to contain a ** copy of zTerm/nTerm. ** ** It is assumed that the buffer associated with pNode is already large ** enough to accommodate the new entry. The buffer associated with pPrev ** is extended by this function if required. ** ** If an error (i.e. OOM condition) occurs, an SQLite error code is ** returned. Otherwise, SQLITE_OK.
 label: code-design
64046. ** Return a "position-list blob" corresponding to the current position of ** cursor pCsr via sqlite3_result_blob(). A position-list blob contains ** the current position-list for each phrase in the query associated with ** cursor pCsr. ** ** A position-list blob begins with (nPhrase-1) varints, where nPhrase is ** the number of phrases in the query. Following the varints are the ** concatenated position lists for each phrase, in order. ** ** The first varint (if it exists) contains the size of the position list ** for phrase 0. The second (same disclaimer) contains the size of position ** list 1. And so on. There is no size field for the final position list, ** as it can be derived from the total size of the blob.
64047. The maximum supported page-size is 65536 bytes. This means that ** the maximum number of record bytes stored on an index B-Tree ** page is less than 16384 bytes and may be stored as a 2-byte ** varint. This information is used to attempt to avoid parsing ** the entire cell by checking for the cases where the record is ** stored entirely within the b-tree page by inspecting the first ** 2 bytes of the cell.
64048. expr ::= PLUS expr
64049. expr ::= colset COLON LP expr RP
64050. 208
64051. 0x17
64052. Use sqlite3EndTable() to add the view to the SQLITE_MASTER table
64053. ***** Begin file rtree.h *****
64054. Number of values to buffer
64055. Source registers if SQLITE_ECEL_REF
64056. Caused by OOM in sqlite3KeyInfoAlloc()
64057. Return the sqlite3_file object for the WAL file
64058. Pointer into the body of aDoclist
64059. ** Open a new iterator to iterate though all rowids that match the ** specified token or token prefix.
64060. synopsis: accum=r[P1] N=P2
64061. xCommit
64062. Combined size of cells placed on i-th page

64063. ** A single object of this type is allocated when the FTS5 module is ** registered with a database handle. It is used to store pointers to ** all registered FTS5 extensions - tokenizers and auxiliary functions.

64064. Control reaches here if best-so-far path pTo=aTo[jj] covers the ** same set of loops and has the same isOrdered setting as the ** candidate path. Check to see if the candidate should replace ** pTo or if the candidate should be skipped. *** The conditional is an expanded vector comparison equivalent to: ** (pTo->rCost,pTo->nRow,pTo->rUnsorted) <= (rCost,nOut,rUnsorted)

64065. Persistent WAL mode

64066. Fields below are only available in SQLite 3.8.2 and later

64067. case_else

64068. content

64069. MJ checksum value read from journal

64070. Promote anything this size or smaller

64071. Callback information

64072. ***** End of vdbeTrace.c *****

64073. OP_Delete: index in a DELETE op

64074. Count the tokens in this AND/NEAR cluster. If none of the doclists ** associated with the tokens spill onto overflow pages, or if there is ** only 1 token, exit early. No tokens to defer in this case.

64075. In direct mode, reference the sorting index rather ** than the source table

64076. Write the aPgn[] array entry and the hash-table slot.

64077. OUTPUT: Collation sequence name

64078. endif pGroupBy. Begin aggregate queries without GROUP BY:

64079. Name of the WAL file

64080. Indexed columns affinity

64081. This iterator is already at EOF for this column.

64082. Opcode: VOpen P1 * * P4 * *** P4 is a pointer to a virtual table object, an sqlite3_vtab structure. ** P1 is a cursor number. This opcode opens a cursor to the virtual ** table and stores that cursor in P1.

64083. 810

64084. ** Return the size of a BtCursor object in bytes. *** This interfaces is needed so that users of cursors can preallocate ** sufficient storage to hold a cursor. The BtCursor object is opaque ** to users so they cannot do the sizeof() themselves - they must call ** this routine.

64085. Link the new Index structure to its table and to the other ** in-memory database structures.

64086. Index in b.paCell[] of cell after i-th page

64087. Set up the new page-index array

64088. NB: zFilename exists and remains valid until the file is closed ** according to requirement F11141. So we do not need to make a ** copy of the filename.

64089. ** End of interface to code in fts5.c. *****

64090. non-decimal integer types. %x %o

64091. ** Added for 3.5.0

64092. If non-zero, increment the row change counter

64093. Step 5a.

64094. Close a pragma virtual table cursor

64095. same as TK_BITAND, synopsis: r[P3]=r[P1]&r[P2]

64096. **comment:** The xBestIndex method of this virtual table requires an estimate of ** the number of rows in the virtual table to calculate the costs of ** various strategies. If possible, this estimate is loaded from the ** sqlite_stat1 table (with RTREE_MIN_ROWEST as a hard-coded minimum). ** Otherwise, if no sqlite_stat1 entry is available, use ** RTREE_DEFAULT_ROWEST.

label: code-design

64097. explain ::= EXPLAIN

64098. The prepared statement

64099. This function (for internal use only) locates an element in an ** hash table that matches the given key. If no element is found, ** a pointer to a static null element with HashElem.data==0 is returned. ** If pH is not NULL, then the hash for this key is written to *pH.

64100. ** Field iChng of the index being scanned has changed. So at this point ** p->current contains a sample that reflects the previous row of the ** index. The value of anEq[iChng] and subsequent anEq[] elements are ** correct at this point.

64101. An integer primary key. If the table has an explicit IPK, use ** its name. Otherwise, use "rbu_rowid".

64102. Database

64103. **comment:** Total of all malloc calls - includes internal frag

label: code-design

64104. Number of sorting key columns, including OP_Sequence

64105. A hidden column in a virtual table

64106. If the "last page" field of the wal-index header snapshot is 0, then ** no data will be read from the wal under any circumstances. Return early ** in this case as an optimization. Likewise, if pWal->readLock==0, ** then the WAL is ignored by the reader so return early, as if the ** WAL were empty.

64107. TK_GT

64108. 281

64109. Associated database handle

64110. Allocate and populate the array to return.

64111. CURRENT_TIMESTAMP

64112. Register for column value

64113. ** Set both the "read version" (single byte at byte offset 18) and ** "write version" (single byte at byte offset 19) fields in the database ** header to iVersion.

64114. Number of references to columns in pSrcList

64115. 0: (end_constraints && !bRev && !endEq)

64116. ** Estimated quantities used for query planning are stored as 16-bit ** logarithms. For quantity X, the value stored is $10 \log_2(X)$. This ** gives a possible range of values of approximately 1.0e986 to 1e-986. ** But the allowed values are "grainy". Not every value is representable. ** For example, quantities 16 and 17 are both represented by a LogEst ** of 40. However, since LogEst quantities are suppose to be estimates, ** not exact values, this imprecision is not a problem. *** "LogEst" is short for "Logarithmic Estimate". *** Examples: ** 1 -> 0 20 -> 43 10000 -> 132 ** 2 -> 10 25 -> 46 25000 -> 146 ** 3 -> 16 100 -> 66 1000000 -> 199 ** 4 -> 20 1000 -> 99 1048576 -> 200 ** 10 -> 33 1024 -> 100 4294967296 -> 320 ** ** The LogEst can be negative to indicate fractional values. ** Examples: ** ** 0.5 -> -10 0.1 -> -33 0.0625 -> -40

64117. fall-through

64118. 315

64119. Double-check that the aSyscall[] array has been constructed ** correctly. See ticket [bb3a86e890c8e96ab]

64120. Set useTempTable to TRUE if the result of the SELECT statement ** should be written into a temporary table (template 4). Set to ** FALSE if each output row of the SELECT can be written directly into ** the destination table (template 3). *** A temp table must be used if the table being updated is also one ** of the tables being read by the SELECT statement. Also use a ** temp table in the case of row triggers.

64121. 249

64122. 0x00 .. 0xF

64123. ** This is the top-level implementation of sqlite3_step(). Call ** sqlite3Step() to do most of the work. If a schema error occurs, ** call sqlite3Reprepare() and try again.

64124. Bind values to this statement

64125. When the number of output rows reaches nRow, that means the ** listing has finished and sqlite3_step() should return SQLITE_DONE. ** nRow is the sum of the number of rows in the main program, plus ** the sum of the number of rows in all trigger subprograms encountered ** so far. The nRow value will increase as new trigger subprograms are ** encountered, but p->pc will eventually catch up to nRow.

64126. Opcode: OpenWrite P1 P2 P3 P4 P5 ** Synopsis: root=P2 iDb=P3 ** ** Open a read/write cursor named P1 on the table or index whose root ** page is P2. Or if P5!=0 use the content of register P2 to find the ** root page. *** The P4 value may be either an integer (P4_INT32) or a pointer to ** a KeyInfo structure

(P4_KEYINFO). If it is a pointer to a KeyInfo ** structure, then said structure defines the content and collating ** sequence of the index being opened. Otherwise, if P4 is an integer ** value, it is set to the number of columns in the table, or to the ** largest index of any column of the table that is actually used. *** This instruction works just like OpenRead except that it opens the cursor ** in read/write mode. For a given table, there can be one or more read-only ** cursors or a single read/write cursor but not both. *** See also OpenRead.

64127. ** This routine walks an expression tree and resolves references to ** table columns and result-set columns. At the same time, do error ** checking on function usage and set a flag if any aggregate functions ** are seen. *** To resolve table columns references we look for nodes (or subtrees) of the ** form X.Y.Z or Y.Z or just Z where ** ** X: The name of a database. Ex: "main" or "temp" or ** the symbolic name assigned to an ATTACH-ed database. *** ** Y: The name of a table in a FROM clause. Or in a trigger ** one of the special names "old" or "new". *** Z: The name of a column in table Y. *** The node at the root of the subtree is modified as follows: *** Expr.op Changed to TK_COLUMN ** Expr.pTab Points to the Table object for X.Y ** Expr.iColumn The column index in X.Y. -1 for the rowid. ** Expr.iTable The VDBE cursor number for X.Y *** ** To resolve result-set references, look for expression nodes of the ** form Z (with no X and Y prefix) where the Z matches the right-hand ** size of an AS clause in the result-set of a SELECT. The Z expression ** is replaced by a copy of the left-hand side of the result-set expression. ** Table-name and function resolution occurs on the substituted expression ** tree. For example, in: *** SELECT a+b AS x, c+d AS y FROM t1 ORDER BY x; *** The "x" term of the order by is replaced by "a+b" to render: *** SELECT a+b AS x, c+d AS y FROM t1 ORDER BY a+b; *** Function calls are checked to make sure that the function is ** defined and that the correct number of arguments are specified. ** If the function is an aggregate function, then the NC_HasAgg flag is ** set and the opcode is changed from TK_FUNCTION to TK_AGG_FUNCTION. ** If an expression contains aggregate functions then the EP_Agg ** property on the expression is set. *** An error message is left in pParse if anything is amiss. The number ** if errors is returned.

64128. Table into which to insert

64129. All pages of the b-tree have been visited. Return successfully.

64130. EVIDENCE-OF: R-43737-39999 Every valid SQLite database file begins ** with the following 16 bytes (in hex): 53 51 4c 69 74 65 20 66 6f 72 6d ** 61 74 20 33 00.

64131. ** Generate the end of the WHERE loop. See comments on ** sqlite3WhereBegin() for additional information.

64132. new.* record for first change

64133. Size of each page in pPage[]

64134. ** Allocate the index structure.

64135. bCoreMutex

64136. Number of entries in azCol[]

64137. True if one or more PMAs created

64138. table leaf

64139. Used when p4type is P4_REAL

64140. ** The fsync() system call does not work as advertised on many ** unix systems. The following procedure is an attempt to make ** it work better. *** The SQLITE_NO_SYNC macro disables all fsync()'s. This is useful ** for testing when we want to run through the test suite quickly. ** You are strongly advised *not* to deploy with SQLITE_NO_SYNC ** enabled, however, since with SQLITE_NO_SYNC enabled, an OS crash ** or power failure will likely corrupt the database file. *** SQLite sets the dataOnly flag if the size of the file is unchanged. ** The idea behind dataOnly is that it should only write the file content ** to disk, not the inode. We only set dataOnly if the file size is ** unchanged since the file size is part of the inode. However, ** Ted Ts'o tells us that fdatasync() will also write the inode if the ** file size has changed. The only real difference between fdatasync() ** and fsync(), Ted tells us, is that fdatasync() will not flush the ** inode if the mtime or owner or other inode attributes have changed. ** We only care about the file size, not the other file attributes, so ** as far as SQLite is concerned, an fdatasync() is always adequate. ** So, we always use fdatasync() if it is available, regardless of ** the value of the dataOnly flag.

64141. IMP: R-33038-09382

64142. Fts5Storage object

64143. ** This function is the implementation of both the xConnect and xCreate ** methods of the FTS3 virtual table. *** The argv[] array contains the following: ** ** argv[0] -> module name ("fts5vocab") ** argv[1] -> database name ** argv[2] -> table name *** then: *** argv[3] -> name of fts5 table ** argv[4] -> type of fts5vocab table *** or, for tables in the TEMP schema only. *** argv[3] -> name of fts5 tables database ** argv[4] -> name of fts5 table ** argv[5] -> type of fts5vocab table

64144. synopsis: key=r[P2@P3]

64145. Size of the header

64146. ** Add term zTerm to the SegmentNode. It is guaranteed that zTerm is larger ** (according to memcmp) than the previous term.

64147. ** Reset the PRNG back to its uninitialized state. The next call ** to sqlite3_randomness() will reseed the PRNG using a single call ** to the xRandomness method of the default VFS.

64148. Comparison result

64149. **comment:** Special Comments: *** Some comments have special meaning to the tools that measure test ** coverage: *** NO_TEST - The branches on this line are not ** measured by branch coverage. This is ** used on lines of code that actually ** implement parts of coverage testing. *** OPTIMIZATION-IF-TRUE - This branch is allowed to always be false ** and the correct answer is still obtained, ** though perhaps more slowly. *** OPTIMIZATION-IF-FALSE - This branch is allowed to always be true ** and the correct answer is still obtained, ** though perhaps more slowly. *** PREVENTS-HARMLESS-OVERREAD - This branch prevents a buffer overread ** that would be harmless and undetectable ** if it did occur. *** In all cases, the special comment must be enclosed in the usual ** slash-asterisk...asterisk-slash comment marks, with no spaces between the ** asterisks and the comment text.

label: test

64150. **comment:** ** The following parameters determine how many adjacent pages get involved ** in a balancing operation. NN is the number of neighbors on either side ** of the page that participate in the balancing operation. NB is the ** total number of pages that participate, including the target page and ** NN neighbors on either side. *** The minimum value of NN is 1 (of course). Increasing NN above 1 ** (to 2 or 3) gives a modest improvement in SELECT and DELETE performance ** in exchange for a larger degradation in INSERT and UPDATE performance. ** The value of NN appears to give the best results overall.

label: code-design

64151. Size of zName in bytes

64152. synopsis: r[P3]=vcolumn(P2)

64153. Unpacked record to return

64154. Append cells to the end of the page

64155. ** Include the primary Windows SDK header file.

64156. One past the last escaped input char

64157. ** Streaming version of sqlite3changeset_invert().

64158. Num. bytes allocated for WhereInfo struct

64159. If pVtab is already in the aVTrans array, return early

64160. Top-level operator. pExpr->op

64161. Maximum PMA size, in bytes. 0==no limit

64162. True at EOF

64163. GROUP

64164. Iterator for looping through Pager.aSavepoint

64165. Original value of pNew->nSkip

64166. Any prepared statement that invokes this opcode will hold mutexes ** on every btree. This is a prerequisite for invoking ** sqlite3InitCallback().

64167. Step 1b

64168. Position list to iterate through

64169. Column containing best snippet

64170. Copy of pCsr->pKeyInfo with db==0

64171. KEY

64172. Temp file for level-0 PMAs

64173. Absolute level just updated

64174. ** Clear the YMD and HMS and the TZ

64175. uniqueflag :=

64176. Acquire a shared lock

64177. The IN expression
 64178. Mask of FTSS5_TOKEN_* flags
 64179. Size of buffer nBuf
 64180. Bytes of payload
 64181. Type of top-level element
 64182. ** Potential values for BtCursor.eState. *** CURSOR_INVALID: ** Cursor does not point to a valid entry. This can happen (for example) ** because the table is empty or because BtreeCursorFirst() has not been ** called. *** CURSOR_VALID: ** Cursor points to a valid entry. getPayload() etc. may be called. *** CURSOR_SKIPNEXT: ** Cursor is valid except that the Cursor.skipNext field is non-zero ** indicating that the next sqlite3BtreeNext() or sqlite3BtreePrevious() ** operation should be a no-op. *** CURSOR_REQUIRESEEK: ** The table that this cursor was opened on still exists, but has been ** modified since the cursor was last used. The cursor position is saved ** in variables BtCursor.pKey and BtCursor.nKey. When a cursor is in ** this state, restoreCursorPosition() can be called to attempt to ** seek the cursor to the saved position. *** CURSORFAULT: ** An unrecoverable error (an I/O error or a malloc failure) has occurred ** on a different connection that shares the BtShared cache with this ** cursor. The error has left the cache in an inconsistent state. ** Do nothing else with this cursor. Any attempt to use the cursor ** should return the error code stored in BtCursor.skipNext
 64183. ** Rewind the VDBE back to the beginning in preparation for ** running it.
 64184. Label to jump to if WHEN is false
 64185. **comment:** ** Deferred and Immediate FKs ** ----- ** Foreign keys in SQLite come in two flavours: deferred and immediate. ** If an immediate foreign key constraint is violated, ** SQLITE_CONSTRAINT_FOREIGNKEY is returned and the current ** statement transaction rolled back. If a ** deferred foreign key constraint is violated, no action is taken ** immediately. However if the application attempts to commit the ** transaction before fixing the constraint violation, the attempt fails. *** Deferred constraints are implemented using a simple counter associated ** with the database handle. The counter is set to zero each time a ** database transaction is opened. Each time a statement is executed ** that causes a foreign key violation, the counter is incremented. Each ** time a statement is executed that removes an existing violation from ** the database, the counter is decremented. When the transaction is ** committed, the commit fails if the current value of the counter is ** greater than zero. This scheme has two big drawbacks: *** * When a commit fails due to a deferred foreign key constraint, ** there is no way to tell which foreign constraint is not satisfied, ** or which row it is not satisfied for. *** * If the database contains foreign key violations when the ** transaction is opened, this may cause the mechanism to malfunction. *** Despite these problems, this approach is adopted as it seems simpler ** than the alternatives. *** INSERT operations: *** I.1) For each FK for which the table is the child table, search ** the parent table for a match. If none is found increment the ** constraint counter. *** I.2) For each FK for which the table is the parent table, ** search the child table for rows that correspond to the new ** row in the parent table. Decrement the counter for each row ** found (as the constraint is now satisfied). *** DELETE operations: *** D.1) For each FK for which the table is the child table, ** search the parent table for a row that corresponds to the ** deleted row in the child table. If such a row is not found, ** decrement the counter. *** D.2) For each FK for which the table is the parent table, search ** the child table for rows that correspond to the deleted row ** in the parent table. For each found increment the counter. *** UPDATE operations: *** An UPDATE command requires that all 4 steps above are taken, but only ** for FK constraints for which the affected columns are actually ** modified (values must be compared at runtime). *** Note that I.1 and D.1 are very similar operations, as are I.2 and D.2. ** This simplifies the implementation a bit. *** For the purposes of immediate FK constraints, the OR REPLACE conflict ** resolution is considered to delete rows before the new row is inserted. ** If a delete caused by OR REPLACE violates an FK constraint, an exception ** is thrown, even if the FK constraint would be satisfied after the new ** row is inserted. *** Immediate constraints are usually handled similarly. The only difference ** is that the counter used is stored as part of each individual statement ** object (struct Vdbe). If, after the statement has run, its immediate ** constraint counter is greater than zero, ** it returns SQLITE_CONSTRAINT_FOREIGNKEY ** and the statement transaction is rolled back. An exception is an INSERT ** statement that inserts a single row only (no triggers). In this case, ** instead of using a counter, an exception is thrown immediately if the ** INSERT violates a foreign key constraint. This is necessary as such ** an INSERT does not open a statement transaction. *** TODO: How should dropping a table be handled? How should renaming a ** table be handled? *** Query API Notes ** ----- *** Before coding an UPDATE or DELETE row operation, the code-generator ** for those two operations needs to know whether or not the operation ** requires any FK processing and, if so, which columns of the original ** row are required by the FK processing VDBE code (i.e. if FKS were ** implemented using triggers, which of the old.* columns would be ** accessed). No information is required by the code-generator before ** coding an INSERT operation. The functions used by the UPDATE/DELETE ** generation code to query for this information are: *** sqlite3FkRequired() - Test to see if FK processing is required. ** sqlite3FkOldmask() - Query for the set of required old.* columns. *** Externally accessible module functions ** ----- *** sqlite3FkCheck() - Check for foreign key violations. **
 sqlite3FkActions() - Code triggers for ON UPDATE/ON DELETE actions. ** sqlite3FkDelete() - Delete an FKey structure.
label: code-design
 64186. OUT: Total number of tokens in phrase.
 64187. ** Handle a 'special' INSERT of the form: *** "INSERT INTO tbl(tbl) VALUES(<expr>)" *** Argument pVal contains the result of <expr>. Currently the only ** meaningful value to insert is the text 'optimize'.
 64188. Bytes written by each WriteFile() call
 64189. Jump instructions that get retargetted
 64190. IcuCursor
 64191. Parent page of siblings being balanced
 64192. Non-zero if the rowid or PRIMARY KEY changed
 64193. Current rolling back, so do not spill
 64194. Resize the hash table so that it contains "new_size" buckets. ** "new_size" must be a power of 2. The hash table might fail ** to resize if sqliteMalloc() fails. **
 ** Return non-zero if a memory allocation error occurs.
 64195. Foreign key index for pFKey
 64196. Restore the clients cache of the wal-index header to the state it ** was in before the client began writing to the database.
 64197. Flags this file was opened with
 64198. If the azData[] array contains more than one element, elements ** (azData[2]..azData[argc-1]) contain a new record to insert into ** the r-tree structure.
 64199. Dynamically allocated strings. %z
 64200. cmd ::= ANALYZE
 64201. ** This vector defines all the methods that can operate on an ** sqlite3_file for win32 without performing any locking.
 64202. Guard word for sanity
 64203. Create a new entry if true
 64204. All keys of a UNIQUE index used
 64205. Current authentication level
 64206. Next on a list of sharable BtShared structs
 64207. Required space after adding zTerm
 64208. PRAGMA integrity_check ** PRAGMA integrity_check(N) ** PRAGMA quick_check ** PRAGMA quick_check(N) *** Verify the integrity of the database.
 *** The "quick_check" is reduced version of ** integrity_check designed to detect most database corruption ** without the overhead of cross-checking indexes.
 Quick_check ** is linear time whereas integrity_check is O(NlogN).
 64209. INSTEAD of triggers are only for views and views only support INSTEAD ** of triggers.
 64210. xClose
 64211. True for toupper(), false for tolower()
 64212. Result of special query
 64213. xFetch
 64214. same as TK_NOTNULL, synopsis: if r[P1]!=NULL goto P2
 64215. **comment:** ***** This file is an amalgamation of many separate C source files from SQLite ** version 3.20.0. By combining all the individual C code files into this ** single large file, the entire code can be compiled as a single translation ** unit. This allows many compilers to do optimizations that would not be ** possible if the files were compiled separately. Performance improvements ** of 5% or more are commonly seen when SQLite is compiled as a single ** translation unit. *** This file is all you need to compile SQLite. To use SQLite in other ** programs, you need this file and the "sqlite3.h" header file that defines ** the programming interface to the SQLite library. (If you do not have ** the "sqlite3.h" header file at hand, you will find a copy embedded within ** the text of this file. Search for "Begin file sqlite3.h" to find the start ** of the embedded sqlite3.h header file.) Additional code files may be needed ** if you want a wrapper to interface SQLite with your choice of programming ** language. The code for the "sqlite3" command-line shell is also in a ** separate file. This file contains only code for the core SQLite library.
label: code-design

64216. SQLITE OMIT BETWEEN OPTIMIZATION
64217. ** Implementation of the upper() and lower() SQL functions.
64218. Total record (payload) size
64219. True if outer SELECT uses aggregate functions
64220. A table interior node. nPayload==0.
64221. 0x00
64222. Destructor for the string
64223. **** Fields above must be initialized to zero. The fields that follow, ** down to the beginning of the recursive section, do not need to be ** initialized as they will be set before being used. The boundary is ** determined by offsetof(Parse,aColCache). ****
64224. Next available select ID for EXPLAIN output
64225. Maximum unsorted cost of a set of path
64226. ** The input pBlob is guaranteed to be a Blob that is not marked ** with MEM_Zero. Return true if it could be a zero-blob.
64227. sqlite3OsRead() return code
64228. Ephemeral table used to enforce DISTINCT
64229. Buffer containing next term
64230. '<'. Part of < or <= or <>
64231. Maximum size of zOut
64232. ** Generate code that checks the left-most column of index table iCur to see if ** it contains any NULL entries. Cause the register at regHasNull to be set ** to a non-NUL value if iCur contains no NULLs. Cause register regHasNull ** to be set to NULL if iCur contains one or more NULL values.
64233. SQLITE OMIT ANALYZE
64234. expr ::= LP nexplist COMMA expr RP
64235. Single term in FROM clause
64236. ** Macros to compute minimum and maximum of two numbers.
64237. The subexpression broken out
64238. ** Shutdown the page cache. Free all memory and close all files. ** ** If a transaction was in progress when this routine is called, that ** transaction is rolled back. All outstanding pages are invalidated ** and their memory is freed. Any attempt to use a page associated ** with this page cache after this function returns will likely ** result in a core dump. ** ** This function always succeeds. If a transaction is active an attempt ** is made to roll it back. If an error occurs during the rollback ** a hot journal may be left in the filesystem but no error is returned ** to the caller.
64239. b.szCell
64240. Index of term<=? value in apVal
64241. ***** Begin file vdbeapi.c *****
64242. Collating sequence. If NULL, use the default
64243. If the bit corresponding to this variable in Vdbe.expmask is set, then ** binding a new value to this variable invalidates the current query plan. ** **
IMPLEMENTATION-OF: R-48440-37595 If the specific value bound to host ** parameter in the WHERE clause might influence the choice of query plan ** for a statement, then the statement will be automatically recompiled, ** as if there had been a schema change, on the first sqlite3_step() call ** following any change to the bindings of that parameter.
64244. Number of columns in parent key
64245. Current offset into pData
64246. ** Generate code for scalar subqueries used as a subquery expression, EXISTS, ** or IN operators. Examples: ** ** (SELECT a FROM b) -- subquery **
EXISTS (SELECT a FROM b) -- EXISTS subquery ** x IN (4,5,11) -- IN operator with list on right-hand side ** x IN (SELECT a FROM b) -- IN operator with subquery on the right ** ** The pExpr parameter describes the expression that contains the IN ** operator or subquery. ** ** If parameter isRowid is non-zero, then expression pExpr is guaranteed ** to be of the form "<rowid> IN (?, ?, ?)", where <rowid> is a reference ** to some integer key column of a table B-Tree. In this case, use an ** intkey B-Tree to store the set of IN(...) values instead of the usual ** (slower) variable length keys B-Tree. ** ** If rMayBeNull is non-zero, that means that the operation is an IN ** (not a SELECT or EXISTS) and that the RHS might contain NULLs. ** All this routine does is initialize the register given by rMayBeNull ** to NULL. Calling routines will take care of changing this register ** value to non-NUL if the RHS is NULL-free. ** ** For a SELECT or EXISTS operator, return the register that holds the ** result. For a multi-column SELECT, the result is stored in a contiguous ** array of registers and the return value is the register of the left-most ** result column. Return 0 for IN operators or if an error occurs.
64247. **comment:** ** Do a deep comparison of two expression trees. Return 0 if the two ** expressions are completely identical. Return 1 if they differ only ** by a COLLATE operator at the top level. Return 2 if there are differences ** other than the top-level COLLATE operator. ** ** If any subelement of pB has Expr.iTable==(-1) then it is allowed ** to compare equal to an equivalent element in pA with Expr.iTable==iTab. ** ** The pA side might be using TK_REGISTER. If that is the case and pB is ** not using TK_REGISTER but is otherwise equivalent, then still return 0. ** ** Sometimes this routine will return 2 even if the two expressions ** really are equivalent. If we cannot prove that the expressions are ** identical, we return 2 just to be safe. So if this routine ** returns 2, then you do not really know for certain if the two ** expressions are the same. But if you get a 0 or 1 return, then you ** can be sure the expressions are the same. In the places where ** this routine is used, it does not hurt to get an extra 2 - that ** just might result in some slightly slower code. But returning ** an incorrect 0 or 1 could lead to a malfunction. ** ** If pParse is not NULL then TK_VARIABLE terms in pA with bindings in ** pParse->pReprepare can be matched against literals in pB. The ** pParse->pVdbe->expmask bitmask is updated for each variable referenced. ** If pParse is NULL (the normal case) then any TK_VARIABLE term in ** Argument pParse should normally be NULL. If it is not NULL and pA or ** pB causes a return value of 2.
label: code-design
64248. Info on how to code the DISTINCT keyword
64249. Loop counter: Field of the foreign key
64250. ** All default VFSes for unix are contained in the following array. ** ** Note that the sqlite3_vfs.pNext field of the VFS object is modified ** by the SQLite core when the VFS is registered. So the following ** array cannot be const.
64251. Minimum amount of incr-merge work to do
64252. Schema to which this item is fixed
64253. The x subexpression
64254. Inputs
64255. Address of ptr to next freeblock
64256. Top of the loop for inserting rows
64257. Vdbe is guaranteed to have been allocated by this stage.
64258. Number of segment-iters in use
64259. ** Access routines. To delete, insert a NULL pointer.
64260. Mark non-alphanumeric ASCII characters as delimiters
64261. ** Free any cursor components allocated by sqlite3VdbeSorterXXX routines.
64262. ** Check to see if the given expression is of the form ** ** column OP expr ** ** where OP is one of MATCH, GLOB, LIKE or REGEXP and "column" is a ** column of a virtual table. ** ** If it is then return TRUE. If not, return FALSE.
64263. Opcode: Program P1 P2 P3 P4 P5 ** ** Execute the trigger program passed as P4 (type P4_SUBPROGRAM). ** ** P1 contains the address of the memory cell that contains the first memory ** cell in an array of values used as arguments to the sub-program. P2 ** contains the address to jump to if the sub-program throws an IGNORE ** exception using the RAISE() function. Register P3 contains the address ** of a memory cell in this (the parent) VM that is used to allocate the ** memory required by the sub-vdbe at runtime. ** ** P4 is a pointer to the VM containing the trigger program. ** ** If P5 is non-zero, then recursive program invocation is enabled.
64264. Opcode: MemMax P1 P2 * * * * Synopsis: r[P1]=max(r[P1],r[P2]) ** ** P1 is a register in the root frame of this VM (the root frame is ** different from the current frame if this instruction is being executed ** within a sub-program). Set the value of register P1 to the maximum of ** its current value and the value in register P2. ** ** This instruction throws an error if the memory cell is not initially ** an integer.
64265. The third parameter
64266. FTS5 backend to iterate within
64267. Journal file must be open.
64268. Search for an existing table

64269. Cursor returned by porterOpen
64270. Figure out the node size to use.
64271. When an OP_Program opcode is encounter (the only opcode that has ** a P4_SUBPROGRAM argument), expand the size of the array of subprograms ** kept in p->aMem[9].z to hold the new program - assuming this subprogram ** has not already been seen.
64272. Depth of tree according to TreeDepth()
64273. Write number of bytes if zBuff[] used here
64274. For sizes 2 through MX_SMALL, inclusive
64275. Constant "k1" from BM25 formula
64276. Maximum number of prepared UPDATE statements held by this module
64277. ** CAPI3REF: Delete A Session Object *** Delete a session object previously allocated using ** [sqlite3session_create()]. Once a session object has been deleted, the ** results of attempting to use pSession with any other session module ** function are undefined. *** Session objects must be deleted before the database handle to which they ** are attached is closed. Refer to the documentation for ** [sqlite3session_create()] for details.
64278. Right input list
64279. Size of leaf without page-index
64280. ** Enter the mutex on every Btree associated with a database ** connection. This is needed (for example) prior to parsing ** a statement since we will be comparing table and column names ** against all schemas and we do not want those schemas being ** reset out from under us. *** There is a corresponding leave-all procedures. *** Enter the mutexes in ascending order by BtShared pointer address ** to avoid the possibility of deadlock when two threads with ** two or more btrees in common both try to lock all their btrees ** at the same instant.
64281. **comment:** ** CAPI3REF: Deprecated Functions ** DEPRECATED *** These functions are [deprecated]. In order to maintain ** backwards compatibility with older code, these functions continue ** to be supported. However, new applications should avoid ** the use of these functions. To encourage programmers to avoid ** these functions, we will not explain what they do.
label: code-design
64282. Pending terms table for this index
64283. If pBuilder->pOrSet is defined, then only keep track of the costs ** and prereqs.
64284. Array of Fts5DlidxWriter objects
64285. Highwater mark for nCurrent
64286. ** Do a node lookup using zPath. Return a pointer to the node on success. ** Return NULL if not found or if there is an error. *** On an error, write an error message into pCtx and increment the ** pParse->nErr counter. *** If pApnd!=NULL then try to append missing nodes and set *pApnd = 1 if ** nodes are appended.
64287. ** Insert the sizes (in tokens) for each column of the document ** with docid equal to p->iPrevDocid. The sizes are encoded as ** a blob of varints.
64288. (289) typetoken ::= typename
64289. Truncate very long tokens to this many bytes. Hard limit is ** (65536-1-1-4-9)==65521 bytes. The limiting factor is the 16-bit offset ** field that occurs at the start of each leaf page (see fts5_index.c).
64290. 0: module 1: schema 2: vtab name 3...: args
64291. Name of the table or index to be reindexed
64292. **comment:** The following are used internally by the fts5_index.c module. They are ** defined here only to make it easier to avoid clashes with the flags ** above.
label: code-design
64293. With this option, sqlite3VtabSync() is defined to be simply ** SQLITE_OK so p is not used.
64294. This block calculates the checksum according to the %_content table
64295. ** This function is similar to sqlite3Fts3ExprParse(), with the following ** differences: *** 1. It does not do expression rebalancing. *** 2. It does not check that the expression does not exceed the ** maximum allowable depth. *** 3. Even if it fails, *ppExpr may still be set to point to an ** expression tree. It should be deleted using sqlite3Fts3ExprFree() ** in this case.
64296. New value for iStartBlock
64297. If this loop satisfies a sort order (pOrderBy) request that ** was passed to this function to implement a "SELECT min(x) ..." ** query, then the caller will only allow the loop to run for ** a single iteration. This means that the first row returned ** should not have a NULL value stored in 'x'. If column 'x' is ** the first one after the nEq equality constraints in the index, ** this requires some special handling.
64298. trigger_event
64299. ***** End of wherecode.c *****
64300. ** Allocate a private rbu VFS for the rbu handle passed as the only ** argument. This VFS will be used unless the call to sqlite3rbu_open() ** specified a URI with a vfs=? option in place of a target database ** file name.
64301. BY
64302. Figure out how many bytes of space are required to store explicitly ** specified collation sequence names.
64303. The glob pattern
64304. Source database connection
64305. ERROR: copy command not terminated by ','
64306. aFile[0] for reading, [1] for writing
64307. Advance each iterator that currently points to iRowid. Or, if iFrom ** is valid - each iterator that points to a rowid before iFrom.
64308. Number of opcodes to add
64309. ** This is the function signature used for all extension entry points. It ** is also defined in the file "loadext.c".
64310. Since nOpt is normally in single digits, a linear search is ** adequate. No need for a binary search.
64311. Write the first index cursor number here
64312. The minor token to shift in
64313. ** An instance of the following structure holds all information about a ** WHERE clause. Mostly this is a container for one or more WhereTerms. ***
Explanation of pOuter: For a WHERE clause of the form ** ** a AND ((b AND c) OR (d AND e)) AND f *** There are separate WhereClause objects for the whole clause and for ** the subclauses "(b AND c)" and "(d AND e)". The pOuter field of the ** subclauses points to the WhereClause object for the whole clause.
64314. Analyze the list of expressions that form the terms of the index and ** report any errors. In the common case where the expression is exactly ** a table column, store that column in aiColumn[]. For general expressions, ** populate pIndex->aColExpr and store XN_EXPR (-2) in aiColumn[]. *** TODO: Issue a warning if two or more columns of the index are identical. ** TODO: Issue a warning if the table primary key is used as part of the ** index key.
64315. OUT: Adjusted offset
64316. Causes schema cookie check after an error
64317. #include <assert.h>
64318. ***** Begin file bitvec.c *****
64319. Make sure the buffer contains at least 10 bytes of input data, or all ** remaining data if there are less than 10 bytes available. This is ** sufficient either for the 'T' or 'P' byte and the varint that follows ** it, or for the two single byte values otherwise.
64320. Address of jump operation
64321. Number of bytes in token
64322. Opcode for update hook: SQLITE_UPDATE or SQLITE_INSERT
64323. ** Open an rbu file handle.
64324. Set \$p to point to the left-most leaf in the tree of eType nodes.
64325. nearest
64326. Max dlidx tree height of 32
64327. Must be so if pLimit==0
64328. Which coordinate to extract
64329. ** As part of a tokenchars= or separators= option, the CREATE VIRTUAL TABLE ** statement has specified that the tokenizer for this table shall consider ** all characters in string zIn/nIn to be separators (if bAlnum==0) or ** token characters (if bAlnum==1). *** For each codepoint in the zIn/nIn string, this function checks if the ** sqlite3FtsUnicodeIsalnum() function already returns the desired result. ** If so, no action is taken. Otherwise, the codepoint is added to the ** unicode_tokenizer.aiException[] array. For the purposes of tokenization, ** the return value of sqlite3FtsUnicodeIsalnum() is inverted for all ** codepoints in the

aiException[] array. *** If a standalone diacritic mark (one that sqlite3FtsUnicodeIsdiacritic() ** identifies as a diacritic) occurs in the zIn/nIn string it is ignored. ** It is not possible to change the behavior of the tokenizer with respect ** to these codepoints.

64330. ** Ordinarily, if no value is explicitly provided, SQLite creates databases ** with page size SQLITE_DEFAULT_PAGE_SIZE. However, based on certain ** device characteristics (sector-size and atomic write() support), ** SQLite may choose a larger value. This constant is the maximum value ** SQLite will choose on its own.

64331. 820

64332. Number of overflow pages to load doclist

64333. ** The number of levels of backtrace to save in new allocations.

64334. ASC or DESC

64335. We can reuse a temporary table generated by a SELECT to our ** right.

64336. Current page of the b-tree

64337. **comment:** *** NOTE: Windows CE is handled differently here due its lack of the Win32 ** API LockFile.

label: test

64338. Compressed expressions only appear when parsing the DEFAULT clause ** on a table column definition, and hence only when pCtx==0. This ** check ensures that an EP_TokenOnly expression is never passed down ** into valueFromFunction().

64339. ** Alternative compound select code generator for cases when there ** is an ORDER BY clause. *** We assume a query of the following form: *** <selectA> <operator> <selectB> ORDER BY <orderbylist> *** <operator> is one of UNION ALL, UNION, EXCEPT, or INTERSECT. The idea ** is to code both <selectA> and <selectB> with the ORDER BY clause as ** co-routines. Then run the co-routines in parallel and merge the results ** into the output. In addition to the two coroutines (called selectA and ** selectB) there are 7 subroutines: *** outA: Move the output of the selectA coroutine into the output ** of the compound query. *** outB: Move the output of the selectB coroutine into the output ** of the compound query. (Only generated for UNION and ** UNION ALL. EXCEPT and INSERTSECT never output a row that ** appears only in B.) *** AltB: Called when there is data from both routines and A<B. *** AeqB: Called when there is data from both routines and A==B. *** AgtB: Called when there is data from both routines and A>B. *** EofA: Called when data is exhausted from selectA. *** EofB: Called when data is exhausted from selectB. *** The implementation of the latter five subroutines depend on which ** <operator> is used: *** UNION ALL UNION EXCEPT INTERSECT ** ----- ** AltB: outA, nextA outA, nextA nextA *** AeqB: outA, nextA nextA nextA outA, nextA *** AgtB: outB, nextB outB, nextB nextB *** EofA: outB, nextB outB, nextB halt *** EofB: outA, nextA outA, nextA outA, nextA halt *** In the AltB, AeqB, and AgtB subroutines, an EOF on A following nextA ** causes an immediate jump to EofA and an EOF on B following nextB causes ** an immediate jump to EofB. Within EofA and EofB, and EOF on entry or ** following nextX causes a jump to the end of the select processing. *** Duplicate removal in the UNION, EXCEPT, and INTERSECT cases is handled ** within the output subroutine. The regPrev register set holds the previously ** output value. A comparison is made against this value and the output ** is skipped if the next results would be the same as the previous. *** The implementation plan is to implement the two routines and seven ** subroutines first, then put the control logic at the bottom. Like this: *** goto Init ** coA: coroutine for left query (A) ** coB: coroutine for right query (B) ** outA: output one row of A ** outB: output one row of B (UNION and UNION ALL only) ** EofA: ... ** EofB: ... ** AltB: ... ** AeqB: ... ** AgtB: ... ** Init: initialize coroutine registers ** yield coA ** if eof(A) goto EofA ** yield coB ** if eof(B) goto EofB ** CmpR: Compare A, B ** Jump AltB, AeqB, AgtB ** End: ... *** We call AltB, AeqB, AgtB, EofA, and EofB "subroutines" but they are not ** actually called using Gosub and they do not Return. EofA and EofB loop ** until all data is exhausted then jump to the "end" labe. AltB, AeqB, ** and AgtB jump to either L2 or to one of EofA or EofB.

64340. Now that the shift has been done, check if the initial "..." are ** required. They are required if (a) this is not the first fragment, ** or (b) this fragment does not begin at position 0 of its column.

64341. The master-journal page number must never be used as a pointer map page

64342. 150

64343. ** Unlink the chunk at mem5.aPool[i] from list it is currently ** on. It should be found on mem5.aiFreelist[iLogsize].

64344. These two variables are set by every call to backup_step0(). They are ** read by calls to backup_remaining() and backup_pagecount().

64345. Fill the sorter until it contains LIMIT+OFFSET entries. (The iLimit ** register is initialized with value of LIMIT+OFFSET.) After the sorter ** fills up, delete the least entry in the sorter after each insert. ** Thus we never hold more than the LIMIT+OFFSET rows in memory at once

64346. Table this info block refers to

64347. WAL to write to

64348. **comment:** WHERE clause term to check

label: code-design

64349. ** Reclaim the memory used by an index

64350. P4 is a 64-bit floating point value

64351. Number of names resolved by this context

64352. ** Report the current state of file logs for all databases

64353. #ifdef SQLITE_DEBUG

64354. Number of locks to hold

64355. ** Return the peak depth of the stack for a parser.

64356. True to update the change-counter

64357. path

64358. Array of offset to rowid fields

64359. Copy the result of the function into register P3

64360. True if SQLITE_BUSY has been encountered

64361. sqlite3SelectExpand() called on this

64362. There exists an unusable MATCH constraint. This means that if ** the planner does elect to use the results of this call as part ** of the overall query plan the user will see an "unable to use ** function MATCH in the requested context" error. To discourage ** this, return a very high cost here.

64363. 106

64364. Increment constraint counter by this

64365. 221

64366. Statement used to insert the encoding

64367. The flags parameter to sqlite3BtreeCreateTable can be the bitwise OR ** of the flags shown below. *** Every SQLite table must have either BTREE_INTKEY or BTREE_BLOBKEY set. ** With BTREE_INTKEY, the table key is a 64-bit integer and arbitrary data ** is stored in the leaves. (BTREE_INTKEY is used for SQL tables.) With ** BTREE_BLOBKEY, the key is an arbitrary BLOB and no content is stored ** anywhere - the key is the content. (BTREE_BLOBKEY is used for SQL ** indices.)

64368. ** Return a pointer to the appropriate hash function given the key class. *** For help in interpreted the obscure C code in the function definition, ** see the header comment on the previous function.

64369. TUNING: If there is no likelihood() value, assume that a ** "col IS NULL" expression matches twice as many rows ** as (col=?).

64370. True if integer overflow seen

64371. The bytes to be written

64372. Last byte of buffer to write

64373. No progress was made on the last round.

64374. Expression to iterate phrases of

64375. ***** End of mem0.c *****

64376. Newly allocated page

64377. Current token number of document

64378. IMP: R-04780-55815

64379. **comment:** ** The following global variables hold counters used for ** testing purposes only. These variables do not exist in ** a non-testing build. These variables are not thread-safe.

label: requirement

64380. Node to return

64381. ** Implementation of the stat_init(N,K,C) SQL function. The three parameters ** are: ** N: The number of columns in the index including the rowid/pk (note 1) ** K: The number of columns in the index excluding the rowid/pk. ** C: The number of rows in the index (note 2) *** Note 1: In the special case of the

covering index that implements a ** WITHOUT ROWID table, N is the number of PRIMARY KEY columns, not the ** total number of columns in the table. ** Note 2: C is only used for STAT3 and STAT4. ** For indexes on ordinary rowid tables, N==K+1. But for indexes on ** WITHOUT ROWID tables, N=K+P where P is the number of columns in the ** PRIMARY KEY of the table. The covering index that implements the ** original WITHOUT ROWID table as N==K as a special case. ** This routine allocates the Stat4Accum object in heap memory. The return ** value is a pointer to the Stat4Accum object. The datatype of the ** return value is BLOB, but it is really just a pointer to the Stat4Accum ** object.

64382. The length of the serialized data for the column

64383. Log2 Size of this block

64384. ** Parameter zIn contains a rank() function specification. The format of ** this is: ** ** + Bareword (function name) ** + Open parenthesis - "(" ** + Zero or more SQL literals in a comma separated list ** + Close parenthesis - ")"

64385. **comment:** ** Return a pointer to a human readable string in a static buffer ** containing the state of the Pager object passed as an argument. This ** is intended to be used within debuggers. For example, as an alternative ** to "print *pPager" in gdb: ** ** (gdb) printf "%s", print_pager_state(pPager)

label: code-design

64386. Return value for this function

64387. Write head of the output list here

64388. If true, fsync() directory after deleting file

64389. Figure out the page-size for the database. This is required in order to ** estimate the cost of loading large doclists from the database.

64390. This condition occurs when an earlier OOM in a call to ** sqlite3_value_text() or sqlite3_value_blob() (perhaps from within ** a conflict-handler) has zeroed the pVal->z pointer. Return NOMEM.

64391. Database holding the object

64392. For descending pending seg-readers only

64393. New aiException[] array

64394. List to which to add the span.

64395. Reset schema after an error if positive

64396. Connection mutex

64397. ** group_concat(EXPR, ?SEPARATOR?)

64398. FTS table handle

64399. The Parse structure

64400. ** Round up a number to the next larger multiple of 8. This is used ** to force 8-byte alignment on 64-bit architectures.

64401. ** Check invariants on a Mem object. ** This routine is intended for use inside of assert() statements, like ** this: assert(sqlite3VdbeCheckMemInvariants(pMem));

64402. Disable mutex on core

64403. Null-terminated column definition

64404. **comment:** ** CAPI3REF: Application Defined Page Cache. ** KEYWORDS: {page cache} ** ** ^{The [sqlite3_config]([SQLITE_CONFIG_PCACHE2], ...)} interface can ** register an alternative page cache implementation by passing in an ** instance of the sqlite3_pcache_methods2 structure.^ ** In many applications, most of the heap memory allocated by ** SQLite is used for the page cache. ** By implementing a ** custom page cache using this API, an application can better control ** the amount of memory consumed by SQLite, the way in which ** that memory is allocated and released, and the policies used to ** determine exactly which parts of a database file are cached and for ** how long. ** The alternative page cache mechanism is an ** extreme measure that is only needed by the most demanding applications. ** The built-in page cache is recommended for most uses. ** ^{The contents of the sqlite3_pcache_methods2 structure are copied to an ** internal buffer by SQLite within the call to [sqlite3_config]. Hence ** the application may discard the parameter after the call to ** [sqlite3_config] returns.^ ** [[the xInit() page cache method]] ** ^{The xInit() method is called once for each effective ** call to [sqlite3_initialize()].^ ** (usually only once during the lifetime of the process).^ ** [[the xShutdown() page cache method]] ** ^{The xShutdown() method is called by [sqlite3_shutdown()]. ** It can be used to clean up ** any outstanding resources before process shutdown, if required. ** ^{The xShutdown() method may be NULL. ** ^{SQLite automatically serializes calls to the xInit method, ** so the xInit method need not be threadsafe. ^{The ** xShutdown method is only called from [sqlite3_shutdown()] so it does ** not need to be threadsafe either. All other methods must be threadsafe ** in multithreaded applications. ** ^{SQLite will never invoke xInit() more than once without an intervening ** call to xShutdown(). ** ^{[[the xCreate() page cache methods]] ** ^{SQLite invokes the xCreate() method to construct a new cache instance. ** SQLite will typically create one cache instance for each open database file, ** though this is not guaranteed. ^{The ** first parameter, szPage, is the size in bytes of the pages that must ** be allocated by the cache. ^{szPage will always a power of two. ^{The ** second parameter szExtra is a number of bytes of extra storage ** associated with each page cache entry. ^{The szExtra parameter will ** a number less than 250. SQLite will use the ** extra szExtra bytes on each page to store metadata about the underlying ** database page on disk. The value passed into szExtra depends ** on the SQLite version, the target platform, and how SQLite was compiled. ** ^{The third argument to xCreate(), bPurgeable, is true if the cache being ** created will be used to cache database pages of a file stored on disk, or ** false if it is used for an in-memory database. The cache implementation ** does not have to do anything special based with the value of bPurgeable; ** it is purely advisory. ^{On a cache where bPurgeable is false, SQLite will ** never invoke xUnpin() except to deliberately delete a page. ** ^{In other words, calls to xUnpin() on a cache with bPurgeable set to ** false will always have the "discard" flag set to true. ** ^{Hence, a cache created with bPurgeable false will ** never contain any unpinned pages. ** ^{[[the xCachesize() page cache method]] ** ^{The xCachesize() method may be called at any time by SQLite to set the ** suggested maximum cache-size (number of pages stored by) the cache ** instance passed as the first argument. This is the value configured using ** the SQLite "[PRAGMA cache_size]" command.)^ As with the bPurgeable ** parameter, the implementation is not required to do anything with this ** value; it is advisory only. ** ^{[[the xPagecount() page cache methods]] ** ^{The xPagecount() method must return the number of pages currently ** stored in the cache, both pinned and unpinned. ** ^{[[the xFetch() page cache methods]] ** ^{The xFetch() method locates a page in the cache and returns a pointer to ** an sqlite3_pcache_page object associated with that page, or a NULL pointer. ** ^{The pBuf element of the returned sqlite3_pcache_page object will be a ** pointer to a buffer of szPage bytes used to store the content of a ** single database page. The pExtra element of sqlite3_pcache_page will be ** a pointer to the szExtra bytes of extra storage that SQLite has requested ** for each entry in the page cache. ** ^{The page to be fetched is determined by the key. ^{The minimum key value ** is 1. After it has been retrieved using xFetch, the page is considered ** to be "pinned". ** ^{If the requested page is already in the page cache, then the page cache ** implementation must return a pointer to the page buffer with its content ** intact. If the requested page is not already in the cache, then the ** cache implementation should use the value of the createFlag ** parameter to help it determined what action to take: ** ^{<table border=1 width=85% align=center> ** <tr><th> createFlag </th> Behavior when page is not already in cache ** <tr><td> 0 <td> Do not allocate a new page. Return NULL. ** <tr><td> 1 <td> Allocate a new page if it easy and convenient to do so. ** ^{Otherwise return NULL. ** <tr><td> 2 <td> Make every effort to allocate a new page. Only return ** NULL if allocating a new page is effectively impossible. ** </table> ** ^{SQLite will normally invoke xFetch() with a createFlag of 0 or 1. SQLite ** will only use a createFlag of 2 after a prior call with a createFlag of 1 ** failed.)^ In between the to xFetch() calls, SQLite may ** attempt to unpin one or more cache pages by spilling the content of ** pinned pages to disk and synching the operating system disk cache. ** ^{[[the xUnpin() page cache method]] ** ^{xUnpin() is called by SQLite with a pointer to a currently pinned page ** as its second argument. If the third parameter, discard, is non-zero, ** then the page must be evicted from the cache. ** ^{If the discard parameter is ** zero, then the page may be discarded or retained at the discretion of ** page cache implementation. ^{The page cache implementation ** may choose to evict unpinned pages at any time. ** ^{The cache must not perform any reference counting. A single ** call to xUnpin() unpins the page regardless of the number of prior calls ** to xFetch(). ** ^{[[the xRekey() page cache methods]] ** ^{The xRekey() method is used to change the key value associated with the ** page passed as the second argument. If the cache ** previously contains an entry associated with newKey, it must be ** discarded. ^{Any prior cache entry associated with newKey is guaranteed not ** to be pinned. ** ^{When SQLite calls the xTruncate() method, the cache must discard all ** existing cache entries with page numbers (keys) greater than or equal ** to the value of the iLimit parameter passed to xTruncate(). If any ** of these pages are pinned, they are implicitly unpinned, meaning that ** they can be safely discarded. ** ^{[[the xDestroy() page cache method]] ** ^{The xDestroy() method is used to delete a cache allocated by xCreate(). ** All resources associated with the specified cache should be freed. ^{After ** calling the xDestroy() method, SQLite considers the [sqlite3_pcache*] ** handle invalid, and will not use it with any other sqlite3_pcache_methods2 ** functions. ** ^{[[the xShrink() page cache method]] ** ^{SQLite invokes the xShrink() method when it wants the page cache to ** free up as much of heap memory as possible. The page cache implementation ** is not obligated to free any memory, but well-behaved implementations should ** do their best.

label: code-design

64405. cmd ::= PRAGMA nm dbnm EQ minus_num

64406. ** Free all resources owned by the object indicated by argument pTask. All ** fields of *pTask are zeroed before returning.

64407. Extract the rowid or primary key for the current row

64408. ** Implementation of the snippet() function for FTS3

64409. Open proxy lock file
64410. Parsing context. Leave error messages here
64411. Invoke the tokenizer destructor to free the tokenizer.
64412. We have successfully halted and closed the VM. Record this fact.
64413. ** The "pPager->journalHdr+JOURNAL_HDR_SZ(pPager)==pPager->journalOff" ** test is related to ticket #2565. See the discussion in the ** pager_playback() function for additional information.
64414. ** The default size of a database page.
64415. Decrement the shared lock counter. Release the lock using an ** OS call only when all threads in this same process have released ** the lock.
64416. Phrase number
64417. Can be null despite NOT NULL constraint
64418. The date/time value to be modified
64419. Argument to xStress
64420. eCreate value for for xFetch()
64421. Advance past the POS_END terminator byte
64422. Access the row with blockid=\$iBlockid
64423. The schema of the expression
64424. Check for the following errors: ** *** Too many attached databases, ** * Transaction currently open ** * Specified database name already being used.
64425. Anything before the first 0x01 is col 0
64426. Value for "start_block" field
64427. Populate the OLD.* pseudo-table register array. These values will be ** used by any BEFORE and AFTER triggers that exist.
64428. Bytes of data to read
64429. Malloc'd output buffers
64430. Text of MATCH query
64431. State: ** SEMI WS OTHER
64432. REINDEX => ID
64433. Total bytes that could not be allocated
64434. Check for immediate foreign key violations.
64435. ** The RBU handle passed as the only argument has just been opened and ** the state database is empty. If this RBU handle was opened for an ** RBU vacuum operation, create the schema in the target db.
64436. 300
64437. 780
64438. porter rule condition: (*o)
64439. 113
64440. Restrictions (8)(9)
64441. Number of columns in the FTS table
64442. ***** End of mem3.c *****
64443. ** Each database connection is an instance of the following structure.
64444. Index of database to analyze
64445. ** Implementation of fts5_source_id() function.
64446. ** if 2014 May 31 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

64447. Generate a subroutine to run when the results from select A ** are exhausted and only data in select B remains.
64448. skip non-significant significant digits ** (increase exponent by d to shift decimal left)
64449. 640
64450. A NOTFOUND or DATA error. Search the table to see if it contains ** a row with a matching primary key. If so, this is a DATA conflict. ** Otherwise, if there is no primary key match, it is a NOTFOUND.
64451. Opcode: EndCoroutine P1 * * * * * The instruction at the address in register P1 is a Yield. ** Jump to the P2 parameter of that Yield. ** After the jump, register P1 becomes undefined. ** * See also: InitCoroutine
64452. Number of columns
64453. True if RTree holds integer coordinates
64454. 5th argument to snippet()
64455. Search through to find the first varint with value 1. This is the ** start of the next columns hits.
64456. Set up the transaction-counter change checking flags when ** transitioning from a SHARED to a RESERVED lock. The change ** from SHARED to RESERVED marks the beginning of a normal ** write operation (not a hot journal rollback).
64457. ** Read data from an rbuVfs-file.
64458. We are currently listing subprograms. Figure out which one and ** pick up the appropriate opcode.
64459. Search the %_segdir table for the absolute level with the smallest ** relative level number that contains at least nMin segments, if any. ** If one is found, set iAbsLevel to the absolute level number and ** nSeg to nMin. If no level with at least nMin segments can be found, ** set nSeg to -1.
64460. True if this is an UPDATE operation
64461. The "NEAR" node (FTS5_STRING)
64462. ** For debugging purposes, record when the mutex subsystem is initialized ** and uninitialized so that we can assert() if there is an attempt to ** allocate a mutex while the system is uninitialized.
64463. Opcode: Vacuum P1 * * * * * Vacuum the entire database P1. P1 is 0 for "main", and 2 or more ** for an attached database. The "temp" database may not be vacuumed.
64464. Overflow payload available for local storage
64465. ***** Aggregate SQL function implementations *****
64466. Register in which Expr value is cached
64467. If the header is read successfully, check the version number to make ** sure the wal-index was not constructed with some future format that ** this version of SQLite cannot understand.
64468. ** Make the JsonNode the return value of the function.
64469. changeset_apply() context object
64470. Successful result
64471. Free any prepared statements held
64472. Opcode: AggStep0 * P2 P3 P4 P5 ** Synopsis: accum=r[P3] step(r[P2@P5]) ** ** Execute the step function for an aggregate. The ** function has P5 arguments. P4 is a pointer to the FuncDef ** structure that specifies the function. Register P3 is the ** accumulator. ** ** The P5 arguments are taken from register P2 and its ** successors.
64473. fts5YYFallback
64474. True if foreign key processing is required
64475. Set to 1 after new rowid is determined
64476. Iterator to read input data
64477. ** The start of buffer (a/n) contains the start of a doclist. The doclist ** may or may not finish within the buffer. This function appends a text ** representation of the part of the doclist that is present to buffer ** pBuf. ** ** The return value is the number of bytes read from the input buffer.
64478. This object holds the prerequisites and the cost of running a ** subquery on one operand of an OR operator in the WHERE clause. ** See WhereOrSet for additional information
64479. **comment:** Allocate a range of temporary registers and the KeyInfo needed ** for the logic that removes duplicate result rows when the ** operator is UNION, EXCEPT, or INTERSECT (but not UNION ALL).

label: code-design

64480. IMMEDIATEJOININSERTMATCHPLANALYZEPRAGMABORTVALUESVIRTUALIMITWHEN
64481. pcache2
64482. Aggregate functions are allowed here
64483. ***** End of opcodes.h *****
64484. Total number of arguments
64485. Output iterator
64486. 760
64487. Information about how to do the compare
64488. isMainJrnl is 0 or 1
64489. Defer all FK constraints
64490. Subfunction return code
64491. (327) anylist ::= anylist LP anylist RP
64492. ** If compiling for a processor that lacks floating point support, ** substitute integer for floating-point.
64493. ** Undo a readlock
64494. Sorter object
64495. Number of arguments
64496. ** Change the value of a limit. Report the old value. ** If an invalid limit index is supplied, report -1. ** Make no changes but still report the old value if the ** new limit is negative. ** ** A new lower limit does not shrink existing constructs. ** It merely prevents new constructs that exceed the limit ** from forming.
64497. A subquery used in place of a table name
64498. For all pages in the cache that are currently dirty or have already ** been written (but not committed) to the log file, do one of the ** following: ** ** + Discard the cached page (if refcount==0), or ** + Reload page content from the database (if refcount>0).
64499. Offset of free space in list.aMemory
64500. Rowid of final leaf block to traverse
64501. ORDER BY or GROUP BY or DISTINCT clause to check
64502. ** xClose - Close a cursor.
64503. Default column to query
64504. Address of OP_CreateTable opcode on CREATE TABLE
64505. ** Advance the changeset iterator to the next change. ** ** If both paRec and pnRec are NULL, then this function works like the public ** API sqlite3changeset_next(). If SQLITE_ROW is returned, then the ** sqlite3changeset_new() and old() APIs may be used to query for values. ** ** Otherwise, if paRec and pnRec are not NULL, then a pointer to the change ** record is written to *paRec before returning and the number of bytes in ** the record to *pnRec. ** ** Either way, this function returns SQLITE_ROW if the iterator is ** successfully advanced to the next change in the changeset, an SQLite ** error code if an error occurs, or SQLITE_DONE if there are no further ** changes in the changeset.
64506. ** Reset the automatic extension loading mechanism.
64507. ROW => ID
64508. Step 3
64509. Undo any frames written (but not committed) to the log
64510. Return 1 or 0.
64511. Offset from the start of data to return bytes from.
64512. ** Free the space allocated for aOp and any p4 values allocated for the ** opcodes contained within. If aOp is not NULL it is assumed to contain ** nOp entries.
64513. Free page blocks
64514. The i-th cell pointer
64515. 256
64516. ** Obtain a complete parse of the JSON found in the first argument ** of the argv array. Use the sqlite3_get_auxdata() cache for this ** parse if it is available. If the cache is not available or if it ** is no longer valid, parse the JSON again and return the new parse, ** and also register the new parse so that it will be available for ** future sqlite3_get_auxdata() calls.
64517. This variation on cellSizePtr() is used inside of assert() statements ** only.
64518. Mask of locks held by siblings
64519. ** This structure is used to build up buffers containing segment b-tree ** nodes (blocks).
64520. Register containing table name
64521. The upper or lower bound just coded
64522. This function (for internal use only) locates an element in an ** hash table that matches the given key. The hash for this key has ** already been computed and is passed as the 4th parameter.
64523. Pointer to page data
64524. If the SQLITE_TESTCTRL_FAULT_INSTALL callback is registered to a ** function that returns SQLITE_ERROR when passed the argument 200, that ** forces worker threads to run sequentially and deterministically ** (via the sqlite3FaultSim() term of the conditional) for testing ** purposes.
64525. Advance the cursor to the next element for json_tree()
64526. Bytes of data in aData
64527. True for without rowid
64528. Success code from a subroutine
64529. xDlSym
64530. ** Each entry in the hash table is represented by an object of the ** following type. Each object, its key (a nul-terminated string) and ** its current data are stored in a single memory allocation. The ** key immediately follows the object in memory. The position list ** data immediately follows the key data in memory. ** ** The data that follows the key is in a similar, but not identical format ** to the doclist data stored in the database. It is: ** ** * Rowid, as a varint ** * Position list, without 0x00 terminator. ** * Size of previous position list and rowid, as a 4 byte ** big-endian integer. ** ** iRowidOff: ** Offset of last rowid written to data area. Relative to first byte of ** structure. ** ** nData: ** Bytes of data written since iRowidOff.
64531. ** Serialize a changeset (or patchset) based on all changesets (or patchsets) ** added to the changegroup object passed as the first argument. ** ** If xOutput is not NULL, then the changeset/patchset is returned to the ** user via one or more calls to xOutput, as with the other streaming ** interfaces. ** ** Or, if xOutput is NULL, then (*ppOut) is populated with a pointer to a ** buffer containing the output changeset before this function returns. In ** this case (*pnOut) is set to the size of the output buffer in bytes. It ** is the responsibility of the caller to free the output buffer using ** sqlite3_free() when it is no longer required. ** ** If successful, SQLITE_OK is returned. Or, if an error occurs, an SQLite ** error code. If an error occurs and xOutput is NULL, (*ppOut) and (*pnOut) ** are both set to 0 before returning.
64532. ** Return the serial-type for the value stored in pMem.
64533. ** A coordinate can be either a floating point number or a integer. All ** coordinates within a single R-Tree are always of the same time.
64534. same as TK_MINUS, synopsis: r[P3]=[P2]-r[P1]
64535. ** Allocate and return a new IncrMerger object to read data from pMerger. ** ** If an OOM condition is encountered, return NULL. In this case free the ** pMerger argument before returning.
64536. If zTerm is not NULL, and this segment is not stored entirely on its ** root node, the range of leaves scanned can be reduced. Do this.
64537. First sorted list to be merged
64538. ** Return a vector of device characteristics.
64539. OUT: Write the thread object here
64540. VIRTUAL
64541. Pointer value
64542. BEFORE => ID
64543. This branch is taken when "%00" appears within the URI. In this ** case we ignore all text in the remainder of the path, name or ** value currently being parsed. So ignore the current character ** and skip to the next "?", "=" or "&", as appropriate.
64544. Hint number of segments
64545. Size of write ahead log header, including checksum.

64546. fts5WriteInit() should have initialized the buffers to (most likely) ** the maximum space required.
64547. Use this index for == or IN queries only
64548. Iterator to return
64549. ** This function is used to add terms implied by JOIN syntax to the ** WHERE clause expression of a SELECT statement. The new term, which ** is ANDed with the existing WHERE clause, is of the form: ** ** (tab1.col1 = tab2.col2) ** ** where tab1 is the iSrc'th table in SrcList pSrc and tab2 is the ** (iSrc+1)'th. Column col1 is column iColLeft of tab1, and col2 is ** column iColRight of tab2.
64550. Expression encoding the function
64551. autoinc ::=
64552. ** An instance of the virtual machine. This structure contains the complete ** state of the virtual machine. ** ** The "sqlite3_stmt" structure pointer that is returned by sqlite3_prepare() ** is really a pointer to an instance of this structure.
64553. Open the journal file if it is not already open.
64554. ** Create the backing store tables (%_content, %_segments and %_segdir) ** required by the FTS3 table passed as the only argument. This is done ** as part of the vtab xCreate() method. ** ** If the p->bHasDocsize boolean is true (indicating that this is an ** FTS4 table, not an FTS3 table) then also create the %_docsize and ** %_stat tables required by FTS4.
64555. **comment:** Database connection (for malloc())
label: code-design
64556. VACUUMVIEWINITIALLY
64557. Name of file to delete
64558. SQLITE_ABORT
64559. ** Register the fts3aux module with database connection db. Return SQLITE_OK ** if successful or an error code if sqlite3_create_module() fails.
64560. The permutation
64561. End of the ephemeral table scan. Or, if using the onepass strategy, ** jump to here if the scan visited zero rows.
64562. sqlite3_test_control(SQLITE_TESTCTRL_NEVER_CORRUPT, int); ** ** Set or clear a flag that indicates that the database file is always well- ** formed and never corrupt. This flag is clear by default, indicating that ** database files might have arbitrary corruption. Setting the flag during ** testing causes certain assert() statements in the code to be activated ** that demonstrate invariants on well-formed database files.
64563. Creates a new file, only if it does not already exist.
64564. Opcode: String P1 P2 P3 P4 P5 ** Synopsis: r[P2]=P4' (len=P1) ** ** The string value P4 of length P1 (bytes) is stored in register P2. ** ** If P3 is not zero and the content of register P3 is equal to P5, then ** the datatype of the register P2 is converted to BLOB. The content is ** the same sequence of bytes, it is merely interpreted as a BLOB instead ** of a string, as if it had been CAST. In other words: ** ** if(P3!=0 and reg[P3]==P5) reg[P2] := CAST(reg[P2] as BLOB)
64565. Value to return via *pnPage
64566. No error has occurred. Free the in-memory buffers.
64567. ** The following variable, if set to a non-zero value, is interpreted as ** the number of seconds since 1970 and is used to set the result of ** sqlite3OsCurrentTime() during testing.
64568. If the column affinity is REAL but the number is an integer, then it ** might be stored in the table as an integer (using a compact ** representation) then converted to REAL by an OP_RealAffinity opcode. ** But we are getting ready to store this value back into an index, where ** it should be converted by to INTEGER again. So omit the OP_RealAffinity ** opcode if it is present
64569. Pointer to first entry with this hash
64570. Integer return code
64571. ** If the input is of the form Z (not Y.Z or X.Y.Z) then the name Z ** might refer to an result-set alias. This happens, for example, when ** we are resolving names in the WHERE clause of the following command: ** ** SELECT a+b AS x FROM table WHERE x<10; ** ** In cases like this, replace pExpr with a copy of the expression that ** forms the result set entry ("a+b" in the example) and return immediately. ** Note that the expression in the result set should have already been ** resolved by the time the WHERE clause is resolved. ** ** The ability to use an output result-set column in the WHERE, GROUP BY, ** or HAVING clauses, or as part of a larger expression in the ORDER BY ** clause is not standard SQL. This is a (goofy) SQLite extension, that ** is supported for backwards compatibility only. Hence, we issue a warning ** on sqlite3_log() whenever the capability is used.
64572. ** Attach PmaReader pReadr to file pFile (if it is not already attached to ** that file) and seek it to offset iOff within the file. Return SQLITE_OK ** if successful, or an SQLite error code if an error occurs.
64573. ** Advance the iterator to the next position. ** ** If no error occurs, SQLITE_OK is returned and the iterator is left ** pointing to the next entry. Otherwise, an error code and message is ** left in the RBU handle passed as the first argument. A copy of the ** error code is returned.
64574. Check that each phrase in the nearset matches the current row. ** Populate the pPhrase->poslist buffers at the same time. If any ** phrase is not a match, break out of the loop early.
64575. The original cursor to be duplicated
64576. Next unixShm with the same unixShmNode
64577. ***** End of build.c *****
64578. The sign in front of the number
64579. joinop ::= JOIN_KW JOIN
64580. Generate code to take the intersection of the two temporary ** tables.
64581. ** Compute the set of tables that might satisfy cases 1 or 3.
64582. The ExprSetVVAProperty() macro is used for Verification, Validation, ** and Accreditation only. It works like ExprSetProperty() during VVA ** processes but is a no-op for delivery.
64583. Append the node to this object
64584. wqlist ::= wqlist COMMA nm eidlist_opt AS LP select RP
64585. Page number obtained
64586. Verify that the WAL header checksum is correct
64587. ** This user-defined SQL function is invoked with a single argument - the ** name of a table expected to appear in the target database. It returns ** the number of auxilliary indexes on the table.
64588. If page 1 was just written, update Pager.dbFileVers to match ** the value now stored in the database file. If writing this ** page caused the database file to grow, update dbFileSize.
64589. where_opt ::= WHERE expr
64590. #include "btreeInt.h"
64591. Write a single frame for this page to the log.
64592. Need to call Mem.xDel() on Mem.z
64593. ** Return the auxiliary data pointer, if any, for the iArg'th argument to ** the user-function defined by pCtx. ** ** The left-most argument is 0. ** ** Undocumented behavior: If iArg is negative then access a cache of ** auxiliary data pointers that is available to all functions within a ** single prepared statement. The iArg values must match.
64594. All files and directories have already been synced, so the following ** calls to sqlite3BtreeCommitPhaseTwo() are only closing files and ** deleting or truncating journals. If something goes wrong while ** this is happening we don't really care. The integrity of the ** transaction is already guaranteed, but some stray 'cold' journals ** may be lying around. Returning an error code won't help matters.
64595. Points to conflicting row, if any
64596. Now search pTab for the exact column.
64597. True if currently recording
64598. Directory sync needed
64599. This is the RFC 7396 MergePatch algorithm.
64600. One of TK_UPDATE, TK_INSERT, TK_DELETE
64601. ** This routine handles sqlite3_file_control() calls that are specific ** to proxy locking.
64602. EVIDENCE-OF: R-24078-09375 Value is a NULL.
64603. ** 2004 April 6 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file implements an external (disk-based) database using BTrees. ** See the header comment on "btreeInt.h" for additional information. ** Including a description of file format and an overview of operation.

64604. nMax*9/10

64605. 330

64606. First arg to comparison function

64607. Dirty read

64608. ** Extra argument values from a PrintfArguments object

64609. **comment:** Comparison information for duplicate removal

label: code-design

64610. ** Return true if the cursor has a hint specified. This routine is ** only used from within assert() statements

64611. Regardless of whether this is a RELEASE or ROLLBACK, destroy all ** savepoints nested inside of the savepoint being operated on.

64612. True if any page has PGHDR_NEED_SYNC

64613. 1's digit

64614. Step 5b.

64615. Pointer to buffer containing term

64616. File descriptor object

64617. ** PRAGMA [schema.]cache_size ** PRAGMA [schema.]cache_size=N ** ** The first form reports the current local setting for the ** page cache size. The second form sets the local ** page cache size value. If N is positive then that is the ** number of pages in the cache. If N is negative, then the ** number of pages is adjusted so that the cache uses -N kibibytes ** of memory.

64618. synopsis: rowset(P1)=r[P2]

64619. ** The public interface to sqlite3Realloc. Make sure that the memory ** subsystem is initialized prior to invoking sqliteRealloc.

64620. First argument to xPreUpdateCallback

64621. EVIDENCE-OF: R-37002-32774 The two-byte integer at offset 3 gives the ** number of cells on the page.

64622. ** Return true if this pager uses a write-ahead log to read page pgno. ** Return false if the pager reads pgno directly from the database.

64623. The database index number

64624. Next registered tokenizer module

64625. Storage handle

64626. Need to free the WhereTerm.u.pAndInfo obj

64627. Initialize memory locations used by GROUP BY aggregate processing

64628. Term suffix

64629. **comment:** ** The iterator passed as the first argument has the following fields set ** as follows. This function sets up the rest of the iterator so that it ** points to the first rowid in the doclist-index. ** ** pData: ** pointer to doclist-index record, ** ** When this function is called pIter->iLeafPgno is the page number the ** doclist is associated with (the one featuring the term).

label: code-design

64630. 1090

64631. (291) signed ::= plus_num (OPTIMIZED OUT)

64632. ** This routine is the only routine in this file with external ** linkage. It returns a pointer to a static sqlite3_mem_methods ** struct populated with the memsys5 methods.

64633. Name of attached database (e.g. "main")

64634. ** Set this global variable to 1 to enable tracing using the TRACE ** macro.

64635. ** Handler for proxy-locking file-control verbs. Defined below in the ** proxying locking division.

64636. The file contains at most 1 b-tree

64637. Invert the result for DESC sort order.

64638. Output each row of result

64639. The table that might be changing

64640. 'documents' values for current csr row

64641. Jump here if true.

64642. Opcode

64643. TRUE after OP_ColumnName has been issued to pVdbe

64644. ** Set the ExprList.a[] .zName element of the most recently added item ** on the expression list. ** ** pList might be NULL following an OOM error. But pName should never be ** NULL. If a memory allocation fails, the pParse->db->mallocFailed flag ** is set.

64645. The index used to access the table

64646. dbsize field from frame header

64647. What version of GCC is being used. 0 means GCC is not being used . ** Note that the GCC_VERSION macro will also be set correctly when using ** clang, since clang works hard to be gcc compatible. So the gcc ** optimizations will also work when compiling with clang.

64648. ** A segment of size nByte bytes has just been written to absolute level ** iAbsLevel. Promote any segments that should be promoted as a result.

64649. Call either BtreeData() or BtreePutData(). If SQLITE_ABORT is ** returned, clean-up the statement handle.

64650. Jump here if database corruption is detected after m has been ** allocated. Free the m object and return SQLITE_CORRUPT.

64651. Foreign key constraint

64652. Index of last frame read

64653. ** Make sure the cursor p is ready to read or write the row to which it ** was last positioned. Return an error code if an OOM fault or I/O error ** prevents us from positioning the cursor to its correct position. ** ** If a MoveTo operation is pending on the given cursor, then do that ** MoveTo now. If no move is pending, check to see if the row has been ** deleted out from under the cursor and if it has, mark the row as ** a NULL row. ** ** If the cursor is already pointing to the correct row and that row has ** not been deleted out from under the cursor, then this routine is a no-op.

64654. Info about the callback functions

64655. expr ::= RAISE LP raisetype COMMA nm RP

64656. If the response to a rowid conflict is REPLACE but the response ** to some other UNIQUE constraint is FAIL or IGNORE, then we need ** to defer the running of the rowid conflict checking until after ** the UNIQUE constraints have run.

64657. Write and decode big-endian 32-bit integer values

64658. The header was successfully read. Return zero.

64659. neverCorrupt

64660. If not at EOF but the current rowid occurs earlier than iFirst in ** the iteration order, move to document iFirst or later.

64661. nMaxpage + 10 - nMinPage

64662. xUpdate

64663. First callback argument

64664. Cursor for the main table

64665. Number of locks to acquire or release

64666. allocate a buffer and convert to UTF8

64667. ** Check the integrity of the freelist or of an overflow page list. ** Verify that the number of pages on the list is N.

64668. REPLACE => ID

64669. Overflows in sorted order

64670. Object containing old values

64671. Index for p->aIndex

64672. ** Extract the snippet text for fragment pFragment from cursor pCsr and ** append it to string buffer pOut.

64673. ** Return the currently defined page size

64674. 299

64675. ** Deregister and destroy an RBU vfs created by an earlier call to ** sqlite3rbu_create_vfs(). ** ** VFS objects are not reference counted. If a VFS object is destroyed ** before all database handles that use it have been closed, the results ** are undefined.

64676. TUNING: Slight bias in favor of no-sort plans
64677. Newly allocated SegReader object
64678. Rank callback (or NULL)
64679. In case this cursor is being reused, close and zero it.
64680. The pH to be searched
64681. Allowed return values from sqlite3WhereIsDistinct()
64682. Open a token cursor on the document.
64683. ** Clear all secondary memory allocations from the parser
64684. SQLITE_UTF16NATIVE
64685. ** current_timestamp() *** This function returns the same value as datetime('now').
64686. ** 2001 September 22 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This is the header file for the generic hash-table implementation ** used in SQLite.
64687. Default ON CONFLICT policy for triggers
64688. ** Add a new element to the pAggInfo->aFunc[] array. Return the index of ** the new element. Return a negative number if malloc fails.
64689. For a co-routine, change all OP_Column references to the table of ** the co-routine into OP_Copy of result contained in a register. ** OP_Rowid becomes OP_Null.
64690. Free the session object itself.
64691. We used to require that sqlite3_reset() be called before retrying ** sqlite3_step() after any error or after SQLITE_DONE. But beginning ** with version 3.7.0, we changed this so that sqlite3_reset() would ** be called automatically instead of throwing the SQLITE_MISUSE error. ** This "automatic-reset" change is not technically an incompatibility, ** since any application that receives an SQLITE_MISUSE is broken by ** definition. *** Nevertheless, some published applications that were originally written ** for version 3.6.23 or earlier do in fact depend on SQLITE_MISUSE ** returns, and those were broken by the automatic-reset change. As a ** a work-around, the SQLITE OMIT_AUTORESET compile-time restores the ** legacy behavior of returning SQLITE_MISUSE for cases where the ** previous sqlite3_step() returned something other than a SQLITE_LOCKED ** or SQLITE_BUSY error.
64692. SQLITE OMIT_VACUUM && SQLITE OMIT_ATTACH
64693. **comment:** ** All individual term iterators in pPhrase are guaranteed to be valid and ** pointing to the same rowid when this function is called. This function ** checks if the current rowid really is a match, and if so populates ** the pPhrase->poslist buffer accordingly. Output parameter *pbMatch ** is set to true if this is really a match, or false otherwise. *** SQLITE_OK is returned if an error occurs, or an SQLite error code ** otherwise. It is not considered an error code if the current rowid is ** not a match.
label: code-design
64694. Should be prime
64695. pColumns can only be NULL due to an OOM but an OOM will cause an ** exit prior to this routine being invoked
64696. WAS: ino_t ino;
64697. Column that is the INTEGER PRIMARY KEY
64698. Set isHighlight to true if this term should be highlighted.
64699. Destructor for the extra data
64700. ** The following routine implements the rough equivalent of localtime_r() ** using whatever operating-system specific localtime facility that ** is available. This routine returns 0 on success and ** non-zero on any kind of error. *** If the sqlite3GlobalConfig.bLocaltimeFault variable is true then this ** routine will always fail. *** EVIDENCE-OF: R-62172-00036 In this implementation, the standard C ** library function localtime_r() is used to assist in the calculation of ** local time.
64701. Number of cells on the page
64702. Results of stat() call
64703. ** Erase all schema information from all attached databases (including ** "main" and "temp") for a single database connection.
64704. Put saved Parse.zAuthContext here
64705. A-overwrites-W
64706. ** Generate code to do an analysis of all indices associated with ** a single table.
64707. SQLITE OMIT_WSD
64708. Extra zero data appended after pData,nData
64709. Either the parent node or NULL
64710. If not NULL, only analyze this one index
64711. ** Convert a UTF-8 filename into whatever form the underlying ** operating system wants filenames in. Space to hold the result ** is obtained from malloc and must be freed by the calling ** function.
64712. The next routine used only within assert() statements
64713. OUT: Set to mallocced error message
64714. SELECT may not be a compound query
64715. Generate code to check constraints and generate index keys and ** do the insertion.
64716. If this expression is a vector to the left or right of a ** inequality constraint (>, <, >= or <=), perform the processing ** on the first element of the vector.
64717. Address register for the output-B subroutine
64718. Value of nPos field
64719. ** PRAGMA soft_heap_limit ** PRAGMA soft_heap_limit = N *** IMPLEMENTATION-OF: R-26343-45930 This pragma invokes the ** sqlite3_soft_heap_limit64() interface with the argument N, if N is ** specified and is a non-negative integer. ** IMPLEMENTATION-OF: R-64451-07163 The soft_heap_limit pragma always ** returns the same integer that would be returned by the ** sqlite3_soft_heap_limit64(-1) C-language function.
64720. The memsys static mutex
64721. ** The sqlite3_mutex_held() and sqlite3_mutex_notheld() routine are ** intended for use only inside assert() statements.
64722. ** CAPI3REF: Run-Time Library Compilation Options Diagnostics *** ^The sqlite3_compileoption_used() function returns 0 or 1 ** indicating whether the specified option was defined at ** compile time. ^The SQLITE_ prefix may be omitted from the ** option name passed to sqlite3_compileoption_used(). *** ^The sqlite3_compileoption_get() function allows iterating ** over the list of options that were defined at compile time by ** returning the N-th compile time option string. ^If N is out of range, ** sqlite3_compileoption_get() returns a NULL pointer. ^The SQLITE_ ** prefix is omitted from any strings returned by ** sqlite3_compileoption_get(). *** ^Support for the diagnostic functions sqlite3_compileoption_used() ** and sqlite3_compileoption_get() may be omitted by specifying the ** [SQLITE OMIT_COMPILEOPTION_DIAGS] option at compile time. *** See also: SQL functions [sqlite_compileoption_used()] and ** [sqlite_compileoption_get()] and the [compile_options pragma].
64723. Merge the forest into a single sorted list on first call
64724. Parse of the input JSON
64725. Arguments for printf format string
64726. Used reduced-size Expr nodes
64727. ***** Begin file wal.h *****
64728. ** Add all changes in the changeset traversed by the iterator passed as ** the first argument to the changegroup hash tables.
64729. ONEPASS_OFF or _SINGLE or _MULTI
64730. Only one flag value allowed
64731. Opcode: Count P1 P2 *** Synopsis: r[P2]=count() *** Store the number of entries (an integer value) in the table or index ** opened by cursor P1 in register P2
64732. 1110
64733. having_opt ::= HAVING expr
64734. Opcode: Not P1 P2 *** Synopsis: r[P2]=!r[P1] *** Interpret the value in register P1 as a boolean value. Store the ** boolean complement in register P2. If the value in register P1 is ** NULL, then a NULL is stored in P2.
64735. ** This function is a wrapper around sqlite3WalFrames(). As well as logging ** the contents of the list of pages headed by pList (connected by pDirty), ** this function notifies any active backup processes that the pages have ** changed. *** The list of pages passed into this routine is always sorted by page number. **

Hence, if page 1 appears anywhere on the list, it will be the first page.

64736. ** Initialize a WhereMaskSet object

64737. The ORDER BY (or GROUP BY clause)

64738. ** Find the name of the index. Make sure there is not already another ** index or table with the same name. *** Exception: If we are reading the names of permanent indices from the ** sqlite_master table (because some other process changed the schema) and ** one of the index names collides with the name of a temporary table or ** index, then we will continue to process this index. *** If pName==0 it means that we are ** dealing with a primary key or UNIQUE constraint. We have to invent our ** own name.

64739. FROM clause term being indexed

64740. 1210

64741. Error message stored here

64742. Last token in coalesced phrase instance

64743. VM this frame belongs to

64744. Return code

64745. ** Free the expression object passed as the only argument.

64746. sqlite3_io_methods object name

64747. Handle for accessing memory mapping

64748. ** The following code executes when the parse fails

64749. Loop through the table columns, appending offset information to ** string-buffer res for each column.

64750. Unless an error occurs, the following loop runs one iteration for each ** page in the B-Tree structure (not including overflow pages).

64751. Current error code

64752. ** Each VDBE holds the result of the most recent sqlite3_step() call ** in p->rc. This routine sets that result back to SQLITE_OK.

64753. **comment:** ** This function is called when an ICU function called from within ** the implementation of an SQL scalar function returns an error. *** The scalar function context passed as the first argument is ** loaded with an error message based on the following two args.

label: code-design

64754. The record is corrupt if any of the following are true: ** (1) the bytes of the header extend past the declared header size ** (2) the entire header was used but not all data was used ** (3) the end of the data extends beyond the end of the record.

64755. Maximum value for szMmap

64756. .db =

64757. Term to select leaves for

64758. Number of previously allocated VDBE cursors

64759. Opcode: Noop * * * * * Do nothing. This instruction is often useful as a jump ** destination.

64760. The journal file into which to write

64761. Various register numbers

64762. True for a read-only database

64763. If the shared-memory file has not yet been opened, open it now.

64764. ** END CODE GENERATED BY tool/mkctime.tcl

64765. (293) carglist ::= carglist ccons

64766. An internal query

64767. Floating point. %f

64768. 8-byte alignment of pageSize

64769. If the iterator is in the error-state, return immediately.

64770. **comment:** ** CAPI3REF: Obtain new.* Values From A Changeset Iterator *** The pIter argument passed to this function may either be an iterator ** passed to a conflict-handler by [sqlite3changeset_apply()], or an iterator ** created by [sqlite3changeset_start()]. In the latter case, the most recent ** call to [sqlite3changeset_next()] must have returned SQLITE_ROW. ** Furthermore, it may only be called if the type of change that the iterator ** currently points to is either [SQLITE_UPDATE] or [SQLITE_INSERT]. Otherwise, ** this function returns [SQLITE_MISUSE] and sets *ppValue to NULL. *** Argument iVal must be greater than or equal to 0, and less than the number ** of columns in the table affected by the current change. Otherwise, ** [SQLITE_RANGE] is returned and *ppValue is set to NULL. ** If successful, this function sets *ppValue to point to a protected ** sqlite3_value object containing the iVal'th value from the vector of ** new row values stored as part of the UPDATE or INSERT change and ** returns SQLITE_OK. If the change is an UPDATE and does not include ** a new value for the requested column, *ppValue is set to NULL and ** SQLITE_OK returned. The name of the function comes from the fact that ** this is similar to the "new.*" columns available to update or delete ** triggers. *** If some other error occurs (e.g. an OOM condition), an SQLite error code ** is returned and *ppValue is set to NULL.

label: code-design

64771. ** The buffer that the argument points to contains a serialized SQL value. ** Return the number of bytes of space occupied by the value (including ** the type byte).

64772. Zero the output variable in case an error occurs.

64773. If the LHS of the MATCH expression was a user column, apply the ** implicit column-filter.

64774. Offset from the beginning of the file

64775. ** Version of sqlite3_free() that is always a function, never a macro.

64776. True if I/O errors persist

64777. ** An array of the following structures is assembled as part of the process ** of selecting tokens to defer before the query starts executing (as part ** of the xFilter() method). There is one element in the array for each ** token in the FTS expression. *** Tokens are divided into AND/NEAR clusters. All tokens in a cluster belong ** to phrases that are connected only by AND and NEAR operators (not OR or ** NOT). When determining tokens to defer, each AND/NEAR cluster is considered ** separately. The root of a tokens AND/NEAR cluster is stored in ** Fts3TokenAndCost.pRoot.

64778. ** Return the mutex associated with a database connection.

64779. 268

64780. Hash value for the element

64781. xFilter - configure scan constraints

64782. ** Register a new auxiliary function with global context pGlobal.

64783. One entry for each identifier on the list

64784. The parse

64785. Name of the term in the USING clause

64786. ** If it is not NULL, pTerm is a term that provides an upper or lower ** bound on a range scan. Without considering pTerm, it is estimated ** that the scan will visit nNew rows. This function returns the number ** estimated to be visited after taking pTerm into account. *** If the user explicitly specified a likelihood() value for this term, ** then the return value is the likelihood multiplied by the number of ** input rows. Otherwise, this function assumes that an "IS NOT NULL" term ** has a likelihood of 0.50, and any other term a likelihood of 0.25.

64787. &'

64788. If the Fts3SegFilter defines a specific term (or term prefix) to search ** for, then advance each segment iterator until it points to a term of ** equal or greater value than the specified term. This prevents many ** unnecessary merge/sort operations for the case where single segment ** b-tree leaf nodes contain more than one term.

64789. This should never happen. We should always be able to ** reacquire the read lock

64790. cmd ::= DROP INDEX ifexists fullname

64791. Level to read input from

64792. Determine the maximum amount of space required.

64793. ** The following three macros, FUNCTION(), LIKEFUNC() and AGGREGATE() are ** used to create the initializers for the FuncDef structures. ***

FUNCTION(zName, nArg, iArg, bNC, xFunc) ** Used to create a scalar function definition of a function zName ** implemented by C function xFunc that accepts nArg arguments. The ** value passed as iArg is cast to a (void*) and made available ** as the user-data (sqlite3_user_data()) for the function. If ** argument bNC is true, then the SQLITE_FUNC_NEEDCOLL flag is set. *** VFUNCTION(zName, nArg, iArg, bNC, xFunc) ** Like FUNCTION except it omits the SQLITE_FUNC_CONSTANT flag. *** DFUNCTION(zName, nArg, iArg, bNC, xFunc) ** Like FUNCTION except it omits the

SQLITE_FUNC_CONSTANT flag and ** adds the SQLITE_FUNC_SLOCHNG flag. Used for date & time functions ** and functions like sqlite_version() that can change, but not during ** a single query. The iArg is ignored. The user-data is always set ** to a NULL pointer. The bNC parameter is not used. *** **
PURE_DATE(zName, nArg, iArg, bNC, xFunc) ** Used for "pure" date/time functions, this macro is like DFUNCTION ** except that it does set the SQLITE_FUNC_CONSTANT flags. iArg is ** ignored and the user-data for these functions is set to an ** arbitrary non-NUL pointer. The bNC parameter is not used. *** ** AGGREGATE(zName, nArg, iArg, bNC, xStep, xFinal) ** Used to create an aggregate function definition implemented by ** the C functions xStep and xFinal. The first four parameters ** are interpreted in the same way as the first 4 parameters to ** FUNCTION(). *** ** LIKEFUNC(zName, nArg, pArg, flags) ** Used to create a scalar function definition of a function zName ** that accepts nArg arguments and is implemented by a call to C ** function likeFunc. Argument pArg is cast to a (void *) and made ** available as the function user-data (sqlite3_user_data()). The ** FuncDef.flags variable is set to the value passed as the flags ** parameter.

64794. Check if this is a tokenizer specification

64795. ***** End of pcache1.c *****

64796. **** This function is only used as part of an assert() statement. *** ** ** Check to see if pBtree holds the required locks to read or write to the ** table with root page iRoot. Return 1 if it does and 0 if not. *** ** For example, when writing to a table with root-page iRoot via ** Btree connection pBtree: *** ** assert(hasSharedCacheTableLock(pBtree, iRoot, 0, WRITE_LOCK)); ** ** When writing to an index that resides in a sharable database, the ** caller should have first obtained a lock specifying the root page of ** the corresponding table. This makes things a bit more complicated, ** as this module treats each table as a separate structure. To determine ** the table corresponding to the index being written, this ** function has to search through the database schema. *** ** Instead of a lock on the table/index rooted at page iRoot, the caller may ** hold a write-lock on the schema table (root page 1). This is also ** acceptable.

64797. Number of nested statement-transactions

64798. fts5YYERRORSYMBOL is not defined

64799. Transitive constraints

64800. ** Commit data to disk.

64801. Register for new.* record

64802. Size of the output buffer

64803. Hint this end-of-scan boundary term if not NULL

64804. **comment:** ** CAPI3REF: Setting The Result Of An SQL Function ** METHOD: sqlite3_context ** ** These routines are used by the xFunc or xFinal callbacks that ** implement SQL functions and aggregates. See ** [sqlite3_create_function()] and [sqlite3_create_function16()] ** for additional information. *** ** These functions work very much like the [parameter binding] family of ** functions used to bind values to host parameters in prepared statements. ** Refer to the [SQL parameter] documentation for additional information. *** ** ^The sqlite3_result_blob() interface sets the result from ** an application-defined function to be the BLOB whose content is pointed ** to by the second parameter and which is N bytes long where N is the ** third parameter. *** ** ^The sqlite3_result_zeroblob(C,N) and sqlite3_result_zeroblob64(C,N) ** interfaces set the result of the application-defined function to be ** a BLOB containing all zero bytes and N bytes in size. *** ** ^The sqlite3_result_double() interface sets the result from ** an application-defined function to be a floating point value specified ** by its 2nd argument. *** ** ^The sqlite3_result_error() and sqlite3_result_error16() functions ** cause the implemented SQL function to throw an exception. *** ^SQLite uses the string pointed to by the ** 2nd parameter of sqlite3_result_error() or sqlite3_result_error16() ** as the text of an error message. ^SQLite interprets the error ** message string from sqlite3_result_error() as UTF-8. ^SQLite ** interprets the string from sqlite3_result_error16() as UTF-16 in native ** byte order. ^If the third parameter to sqlite3_result_error() ** or sqlite3_result_error16() is negative then SQLite takes as the error ** message all text up through the first zero character. ** ^If the third parameter to sqlite3_result_error() or ** sqlite3_result_error16() is non-negative then SQLite takes that many ** bytes (not characters) from the 2nd parameter as the error message. ** ^The sqlite3_result_error() and sqlite3_result_error16() ** routines make a private copy of the error message text before ** they return. Hence, the calling function can deallocate or ** modify the text after they return without harm. ** ^The sqlite3_result_error_code() function changes the error code ** returned by SQLite as a result of an error in a function. ^By default, ** the error code is SQLITE_ERROR. ^A subsequent call to sqlite3_result_error() ** or sqlite3_result_error16() resets the error code to SQLITE_ERROR. *** ** ^The sqlite3_result_error_toobig() interface causes SQLite to throw an ** error indicating that a string or BLOB is too long to represent. *** ** ^The sqlite3_result_error_nomem() interface causes SQLite to throw an ** error indicating that a memory allocation failed. *** ** ^The sqlite3_result_int() interface sets the return value ** of the application-defined function to be the 32-bit signed integer ** value given in the 2nd argument. ** ^The sqlite3_result_int64() interface sets the return value ** of the application-defined function to be the 64-bit signed integer ** value given in the 2nd argument. *** ** ^The sqlite3_result_null() interface sets the return value ** of the application-defined function to be NULL. *** ** ^The sqlite3_result_text(), sqlite3_result_text16(), ** sqlite3_result_text16e(), and sqlite3_result_text16be() interfaces ** set the return value of the application-defined function to be ** a text string which is represented as UTF-8, UTF-16 native byte order, ** UTF-16 little endian, or UTF-16 big endian, respectively. ** ^The sqlite3_result_text64() interface sets the return value of an ** application-defined function to be a text string in an encoding ** specified by the fifth (and last) parameter, which must be one ** of [SQLITE_UTF8], [SQLITE_UTF16], [SQLITE_UTF16BE], or [SQLITE_UTF16LE]. ** ^SQLite takes the text result from the application from ** the 2nd parameter of the sqlite3_result_text* interfaces. ** ^If the 3rd parameter to the sqlite3_result_text* interfaces ** is negative, then SQLite takes result text from the 2nd parameter ** through the first zero character. ** ^If the 3rd parameter to the sqlite3_result_text* interfaces ** is non-negative, then as many bytes (not characters) of the text ** pointed to by the 2nd parameter are taken as the application-defined ** function result. If the 3rd parameter is non-negative, then it ** must be the byte offset into the string where the NUL terminator would ** appear if the string were NUL terminated. If any NUL characters occur ** in the string at a byte offset that is less than the value of the 3rd ** parameter, then the resulting string will contain embedded NULs and the ** result of expressions operating on strings with embedded NULs is undefined. ** ^If the 4th parameter to the sqlite3_result_text* interfaces ** or sqlite3_result_blob is a non-NULL pointer, then SQLite calls that ** function as the destructor on the text or BLOB result when it has ** finished using that result. ** ^If the 4th parameter to the sqlite3_result_text* interfaces or to ** sqlite3_result_blob is the special constant SQLITE_STATIC, then SQLite ** assumes that the text or BLOB result is in constant space and does not ** copy the content of the parameter nor call a destructor on the content ** when it has finished using that result. ** ^If the 4th parameter to the sqlite3_result_text* interfaces ** or sqlite3_result_blob is the special constant SQLITE_TRANSIENT ** then SQLite makes a copy of the result into space obtained ** from [sqlite3_malloc()] before it returns. *** ** ^The sqlite3_result_value() interface sets the result of ** the application-defined function to be a copy of the ** [unprotected sqlite3_value] object specified by the 2nd parameter. ^The ** sqlite3_result_value() interface makes a copy of the [sqlite3_value] ** so that the [sqlite3_value] specified in the parameter may change or ** be deallocated after sqlite3_result_value() returns without harm. ** ^A [protected sqlite3_value] object may always be used where an ** [unprotected sqlite3_value] object is required, so either ** kind of [sqlite3_value] object can be used with this interface. *** ** ^The sqlite3_result_pointer(C,P,T,D) interface sets the result to an ** SQL NULL value, just like [sqlite3_result_null(C)], except that it ** also associates the host-language pointer P or type T with that ** NULL value such that the pointer can be retrieved within an ** [application-defined SQL function] using [sqlite3_value_pointer()]. ** ^If the D parameter is not NULL, then it is a pointer to a destructor ** for the P parameter. ^SQLite invokes D with P as its only argument ** when SQLite is finished with P. The T parameter should be a static ** string and preferably a string literal. The sqlite3_result_pointer() ** routine is part of the [pointer passing interface] added for SQLite 3.20.0. *** ** If these routines are called from within the different thread ** than the one containing the application-defined function that received ** the [sqlite3_context] pointer, the results are undefined.

label: code-design

64805. Fill in the payload

64806. Process NATURAL keywords, and ON and USING clauses of joins.

64807. An operand of a comparison operator

64808. **comment:** The entire output segment fits on a single node. Normally, this means ** the node would be stored as a blob in the "root" column of the %_segdir ** table. However, this is not permitted in this case. The problem is that ** space has already been reserved in the %_segments table, and so the ** start_block and end_block fields of the %_segdir table must be populated. ** And, by design or by accident, released versions of FTS cannot handle ** segments that fit entirely on the root node with start_block!=0. *** ** Instead, create a synthetic root node that contains nothing but a ** pointer to the single content node. So that the segment consists of a ** single leaf and a single interior (root) node. *** ** Todo: Better might be to defer allocating space in the %_segments ** table until we are sure it is needed.

label: code-design

64809. Loop through all columns of the index and deal with the ones ** that are not constrained by == or IN.

64810. A bunch of assert() statements to check the transaction state variables ** of handle p (type Btree*) are internally consistent.

64811. Regardless of the current state, a temp-file connection always behaves ** as if it has an exclusive lock on the database file. It never updates ** the change-counter field, so the changeCountDone flag is always set.

64812. The database encoding

64813. Store a pointer to the memory block in global structure mem3.

64814. ** Default permissions when creating a new file

64815. ** Set a table filter on a Session Object.

64816. CSE2 from below
64817. The iCol-th column of the result set
64818. See if we are dealing with a quoted phrase. If this is the case, then ** search for the closing quote and pass the whole string to getNextString() ** for processing.
This is easy to do, as fts3 has no syntax for escaping ** a quote character embedded in a string.
64819. comparison function
64820. ** The following constant value is used by the SQLITE_BIGENDIAN and ** SQLITE_LITTLEENDIAN macros.
64821. OK to update SQLITE_MASTER
64822. Bytes of header sector written
64823. same as TK_BITOR, in1, in2, out3
64824. Read the serial types for the next element in each key.
64825. Resize the hash table so that it contains "new_size" buckets. *** The hash table might fail to resize if sqlite3_malloc() fails or ** if the new size is the same as the prior size. ** Return TRUE if the resize occurs and false if not.
64826. If EP_FromJoin, the right table of the join
64827. serial type of first record field
64828. Pointer to a static string
64829. If everything has worked, invoke fts3DisconnectMethod() to free the ** memory associated with the Fts3Table structure and return SQLITE_OK. ** Otherwise, return an SQLite error code.
64830. Column token appears in (-ve -> delete)
64831. IMP: R-37514-35566
64832. True if pStart obtained from sqlite3_malloc()
64833. OUT: Extracted value
64834. An instance of this object describes bulk memory available for use ** by subcomponents of a prepared statement. Space is allocated out ** of a ReusableSpace object by the allocSpace() routine below.
64835. Number of VDBEs currently running
64836. Depth of a subtree
64837. If this is a "NEAR" keyword, check for an explicit nearness.
64838. OUT: Frame associated with *paPgn[0]
64839. sqlite3_mem_methods*
64840. ** Flags passed to the sqlite3ExprDup() function. See the header comment ** above sqlite3ExprDup() for details.
64841. xPagecount
64842. ** The following structure contains information used by the sqliteFix... ** routines as they walk the parse tree to make database references ** explicit.
64843. Remove the search point with the lowest current score.
64844. Copy of flags byte
64845. **comment:** The SQLITE_TEST_REALLOC_STRESS compile-time option is designed to force ** more frequent reallocs and hence provide more opportunities for ** simulated OOM faults. SQLITE_TEST_REALLOC_STRESS is generally used ** during testing only. With SQLITE_TEST_REALLOC_STRESS grow the op array ** by the minimum* amount required until the size reaches 512. Normal ** operation (without SQLITE_TEST_REALLOC_STRESS) is to double the current ** size of the op array or add 1KB of space, whichever is smaller.
label: code-design
64846. Database name
64847. The database on which the SQL executes
64848. Candidate unixInodeInfo object
64849. The major token to shift in
64850. Update the pointer-map and meta-data with the new root-page number.
64851. SQLITE_PAGER_PRAGMAS
64852. Select a master journal file name
64853. Attempt to locate an existing hash entry
64854. Left hand serial type
64855. Silence a compiler warning
64856. Sum of nMax for purgeable caches
64857. UPDATE statement
64858. ** This function is called by the parser after the table-name in ** an "ALTER TABLE <table-name> ADD" statement is parsed. Argument ** pSrc is the full-name of the table being altered. ** ** This routine makes a (partial) copy of the Table structure ** for the table being altered and sets Parse.pNewTable to point ** to it. Routines called by the parser as the column definition ** is parsed (i.e. sqlite3AddColumn()) add the new Column data to ** the copy. The copy of the Table structure is deleted by tokenize.c ** after parsing is finished. ** ** Routine sqlite3AlterFinishAddColumn() will be called to complete ** coding the "ALTER TABLE ... ADD" statement.
64859. Memory register used by AUTOINC
64860. 305
64861. ** Analyze the ORDER BY clause in a compound SELECT statement. Modify ** each term of the ORDER BY clause is a constant integer between 1 ** and N where N is the number of columns in the compound SELECT. ** ** ORDER BY terms that are already an integer between 1 and N are ** unmodified. ORDER BY terms that are integers outside the range of ** 1 through N generate an error. ORDER BY terms that are expressions ** are matched against result set expressions of compound SELECT ** beginning with the left-most SELECT and working toward the right. ** At the first match, the ORDER BY expression is transformed into ** the integer column number. ** ** Return the number of errors seen.
64862. OUT: Size of doclist in bytes
64863. True for FTS4, false for FTS3
64864. Cannot be used with sqlite3RowSetText()
64865. ** Implementation of snippet() function.
64866. Deallocates any cached error strings.
64867. ** Save the current state of the PRNG.
64868. Second change operation
64869. ** If the SQLITE_ENABLE_IOTRACE exists then the global variable ** sqlite3IoTrace is a pointer to a printf-like routine used to ** print I/O tracing messages.
64870. **comment:** ** pX is the RHS of an IN operator. If pX is a SELECT statement ** that can be simplified to a direct table access, then return ** a pointer to the SELECT statement. If pX is not a SELECT statement, ** or if the SELECT statement needs to be manifested into a transient ** table, then return NULL.
label: code-design
64871. 140
64872. Array of target column types
64873. Open the sorter cursor if we are to use one.
64874. ** If pFile is currently larger than iSize bytes, then truncate it to ** exactly iSize bytes. If pFile is not larger than iSize bytes, then ** this function is a no-op. ** ** Return SQLITE_OK if everything is successful, or an SQLite error ** code if an error occurs.
64875. Previous term written to new node
64876. OUT: Docid and doclist for new entry
64877. **comment:** ** The malloc.h header file is needed for malloc_usable_size() function ** on some systems (e.g. Linux).
label: code-design
64878. Create the automatic index
64879. 276
64880. Acquire the mutex
64881. ** Release a reference to a node. If the node is dirty and the reference ** count drops to zero, the node data is written to the database.
64882. Language id to return cksum for
64883. refact ::= RESTRICT

64884. Pager associated with pBt
64885. ***** End of os.c *****
64886. One of the CURSOR_XXX constants (see below)
64887. Offset to first byte to be locked/unlocked
64888. True if a periodic sample
64889. Address of jump instruction
64890. Shared library endings to try if zFile cannot be loaded as written
64891. isInit
64892. Number of leaf pages written
64893. The pager open on the database file
64894. ** Copy data from a buffer to a page, or from a page to a buffer. ** * pPayload is a pointer to data stored on database page pDbPage. ** If argument eOp is false, then nByte bytes of data are copied ** from pPayload to the buffer pointed at by pBuf. If eOp is true, ** then sqlite3PagerWrite() is called on pDbPage and nByte bytes ** of data are copied from the buffer pBuf to pPayload. ** * SQLITE_OK is returned on success, otherwise an error code.
64895. WHERE clause of the UPDATE statement
64896. Mask of old.* columns referenced
64897. ** Resolve names in expressions that can only reference a single table: ** * CHECK constraints ** * WHERE clauses on partial indices ** * The Expr.iTable value for Expr.op==TK_COLUMN nodes of the expression ** is set to -1 and the Expr.iColumn value is set to the column number. ** * Any errors cause an error message to be set in pParse.
64898. ** Function to query the hash-table of tokenizers (see README.tokenizers).
64899. 34
64900. This file is used for Windows only
64901. REPLACE_CONTENT
64902. **comment:** ** Return a nul-terminated string containing the comma separated list of ** assignments that should be included following the "SET" keyword of ** an UPDATE statement used to update the table object that the iterator ** passed as the second argument currently points to if the rbu_control ** column of the data_xxx table entry is set to zMask. ** * The memory for the returned string is obtained from sqlite3_malloc(). ** It is the responsibility of the caller to eventually free it using ** sqlite3_free(). ** * If an OOM error is encountered when allocating space for the new ** string, an error code is left in the rbu handle passed as the first ** argument and NULL is returned. Or, if an error has already occurred ** when this function is called, NULL is returned immediately, without ** attempting the allocation or modifying the stored error code.
label: code-design
64903. IN/OUT: Checksum value
64904. P4 is a pointer to an sqlite3_context object
64905. sqlite3Parser_ENGINEALWAYSONSTACK
64906. System call return code
64907. Node most recently inserted into the tree
64908. The database has been modified
64909. If argument zPath is a NULL pointer, this function is required to open ** a temporary file. Use this buffer to store the file name in.
64910. Set the database size back to the value it was before the savepoint ** being reverted was opened.
64911. Used to iterate through phrase terms
64912. Input text being tokenized
64913. End of interior node buffer
64914. OUT: Sum of PCache1.nMin for purgeable caches
64915. We have an equality constraint on the rowid. Use strategy 1.
64916. ** Test whether or not expression pExpr, which was part of a WHERE clause, ** should be included in the cursor-hint for a table that is on the rhs ** of a LEFT JOIN. Set Walker.eCode to non-zero before returning if the ** expression is not suitable. ** * An expression is unsuitable if it might evaluate to non NULL even if ** a TK_COLUMN node that does affect the value of the expression is set ** to NULL. For example: ** * col IS NULL ** col IS NOT NULL ** coalesce(col, 1) ** CASE WHEN col THEN 0 ELSE 1 END
64917. ONEPASS valid for a single row update
64918. Contains one or more aggregate functions
64919. Second parameter (often the jump destination)
64920. Next pattern and input string chars
64921. If ENABLE_URI_00_ERROR is defined, "%00" in a URI is an error.
64922. Invoke the coroutine to extract information from the SELECT ** and add it to a transient table srcTab. The code generated ** here is from the 4th template: ***
B: open temp table ** L: yield X, goto M at EOF ** insert row from R..R+n into temp table ** goto L ** M: ...
64923. ** Move the seg-iter so that it points to the first rowid on page iLeafPgno. ** It is an error if leaf iLeafPgno does not exist or contains no rowids.
64924. Append nodes to complete path if not NULL
64925. Total number of pages to copy
64926. ** 2001 September 15 ** * The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** * May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Header file for the Virtual DataBase Engine (VDBE) *** This header defines the interface to the virtual database engine ** or VDBE. The VDBE implements an abstract machine that runs a ** simple program to access and modify the underlying database.
64927. Calling thread context
64928. Hash table for functions
64929. Number of reader locks obtained
64930. 20..27 !"#\$%&'
64931. IMP: R-31275-44060
64932. FTS3/FTS4 require virtual tables
64933. ** Expression pExpr is a vector that has been used in a context where ** it is not permitted. If pExpr is a sub-select vector, this routine ** loads the Parse object with a message of the form: ** * "sub-select returns N columns - expected 1" ** * Or, if it is a regular scalar vector: ** * "row value misused"
64934. Iterator variable for various purposes
64935. Query used to initialize the table
64936. Write the revised row estimate here
64937. ***** Begin file build.c *****
64938. Number of bytes in term prefix
64939. Object used to write to the file
64940. 0 for main database, 1 for TEMP, 2.. for ATTACHed
64941. Pointer to one of the indices
64942. (277) explain ::=
64943. ** Variables related to the accumulation of tokens and doclists within the ** in-memory hash tables before they are flushed to disk.
64944. Pointer to Fts5IntegrityCtx object
64945. Use covering indices for full-scans
64946. ** End of interface to code in fts5_vocab.c. *****
64947. Original SQL values of parameters
64948. ** Return a string that describes the kind of error specified in the ** argument. For now, this simply calls the internal sqlite3ErrStr() ** function.
64949. got it, set the type and return ok
64950. First register in array assigned to each index
64951. Used to iterate through memory cells
64952. Count the token characters

64953. ** No-op.
64954. SELECT statement
64955. ** Return a block of memory of at least nBytes in size. ** Return NULL if unable. Return NULL if nBytes==0. ** ** The caller guarantees that nByte is positive.
 *** The caller has obtained a mutex prior to invoking this ** routine so there is never any chance that two or more ** threads can be in this routine at the same time.
64956. Offset on new first leaf page
64957. ** Always assert. This xSelectCallback2 implementation proves that the ** xSelectCallback2 is never invoked.
64958. If the db->init.busy is 1 it means we are reading the SQL off the ** "sqlite_master" or "sqlite_temp_master" table on the disk. ** So do not write to the disk again.
 Extract the root page number ** for the table from the db->init.newTnum field. (The page number ** should have been put there by the sqliteOpenCb routine.) **
 ** If the root page number is 1, that means this is the sqlite_master ** table itself. So mark it read-only.
64959. First namecontext in the list
64960. Input contains fewer than nChar chars
64961. SQL_SEGMENT_IS_APPENDABLE ** Return a single row if the segment with end_block=? is appendable. Or ** no rows otherwise.
64962. Context for fts3ExprIterate()
64963. **comment:** ** Attempt the transfer optimization on INSERTs of the form ** ** INSERT INTO tab1 SELECT * FROM tab2; ** ** The xfer optimization transfers raw records from tab2 over to tab1. ** Columns are not decoded and reassembled, which greatly improves ** performance. Raw index records are transferred in the same way. ** ** The xfer optimization is only attempted if tab1 and tab2 are compatible. ** There are lots of rules for determining compatibility - see comments ** embedded in the code for details. ** ** This routine returns TRUE if the optimization is guaranteed to be used. ** Sometimes the xfer optimization will only work if the destination table ** is empty - a factor that can only be determined at run-time. In that ** case, this routine generates code for the xfer optimization but also ** does a test to see if the destination table is empty and jumps over the ** xfer optimization code if the test fails. In that case, this routine ** returns FALSE so that the caller will know to go ahead and generate ** an unoptimized transfer. This routine also returns FALSE if there ** is no chance that the xfer optimization can be applied. ** ** This optimization is particularly useful at making VACUUM run faster.
label: code-design
64964. ** Insert new entries into the FTS index and %_docszie table.
64965. Name of master journal file if any
64966. Index of column in child table
64967. Do nothing. The work was all in the test
64968. Abort the SQL statement with an error
64969. Registers to hold a row from pTab
64970. ** CAPI3REF: Pseudo-Random Number Generator ** ** SQLite contains a high-quality pseudo-random number generator (PRNG) used to ** select random [ROWID | ROWIDs] when inserting new records into a table that ** already uses the largest possible [ROWID]. The PRNG is also used for ** the build-in random() and randomblob() SQL functions. This interface allows ** applications to access the same PRNG for other purposes. ** ** ^A call to this routine stores N bytes of randomness into buffer P. ** ^The P parameter can be a NULL pointer. ** ** ^If this routine has not been previously called or if the previous ** call had N less than one or a NULL pointer for P, then the PRNG is ** seeded using randomness obtained from the xRandomness method of ** the default [sqlite3_vfs] object. ** ^If the previous call to this routine had an N of 1 or more and a ** non-NNULL P then the pseudo-randomness is generated ** internally and without recourse to the [sqlite3_vfs] xRandomness ** method.
64971. Properties such as "out2" or "jump" that are specified in ** comments following the "case" for each opcode in the vdbe.c ** are encoded into bitvectors as follows:
64972. Value returned by sqlite3_changes()
64973. The page getter for when memory-mapped I/O is enabled
64974. pnBlob must be non-NULL. paBlob may be NULL or non-NULL.
64975. True for a prefix iterator
64976. RP => nothing
64977. **comment:** An aggregate function in the WHERE clause of a query means this must ** be a correlated sub-query, and expression pExpr is an aggregate from ** the parent context. Do not walk the function arguments in this case. ** ** todo: It should be possible to replace this node with a TK_REGISTER ** expression, as the result of the expression must be stored in a ** register at this point. The same holds for TK_AGG_COLUMN nodes.
label: code-design
64978. addrBrk for the outermost loop
64979. Takes the conch by taking a shared lock and read the contents conch, if ** lockPath is non-NULL, the host ID and lock file path must match. A NULL ** lockPath means that the lockPath in the conch file will be used if the ** host IDs match, or a new lock path will be generated automatically ** and written to the conch file.
64980. Move the cursor so that it points to an entry near the key ** specified by pIdxKey or intKey. Return a success code. ** ** For INTKEY tables, the intKey parameter is used. pIdxKey ** must be NULL. For index tables, pldxKey is used and intKey ** is ignored. ** ** If an exact match is not found, then the cursor is always ** left pointing at a leaf page which would hold the entry if it ** were present. The cursor might point to an entry that comes ** before or after the key. ** ** An integer is written into *pRes which is the result of ** comparing the key with the entry to which the cursor is ** pointing. The meaning of the integer written into ** *pRes is as follows: ** ** *pRes<0 The cursor is left pointing at an entry that ** is smaller than intKey/pIdxKey or if the table is empty ** and the cursor is therefore left point to nothing. ** ** *pRes==0 The cursor is left pointing at an entry that ** exactly matches intKey/pIdxKey. ** ** *pRes>0 The cursor is left pointing at an entry that ** is larger than intKey/pIdxKey. ** ** For index tables, the pIdxKey->eqSeen field is set to 1 if there ** exists an entry in the table that exactly matches pIdxKey.
64981. The table containing the value
64982. SELECT rowid ...
64983. OUT: Bytes of string content
64984. ** Insert a new record into the BTree. The content of the new record ** is described by the pX object. The pCur cursor is used only to ** define what table the record should be inserted into, and is left ** pointing at a random location. ** ** For a table btree (used for rowid tables), only the pX.nKey value of ** the key is used. The pX.pKey value must be NULL. The pX.nKey is the ** rowid or INTEGER PRIMARY KEY of the row. The pX.nData,pData,nZero fields ** hold the content of the row. ** ** For an index btree (used for indexes and WITHOUT ROWID tables), the ** key is an arbitrary byte sequence stored in pX.pKey,nKey. The ** pX.pData,nData,nZero fields must be zero. ** ** If the seekResult parameter is non-zero, then a successful call to ** MovetoUnpacked() to seek cursor pCur to (pKey,nKey) has already ** been performed. In other words, if seekResult!=0 then the cursor ** is currently pointing to a cell that will be adjacent to the cell ** to be inserted. If seekResult<0 then pCur points to a cell that is ** smaller than (pKey,nKey). If seekResult>0 then pCur points to a cell ** that is larger than (pKey,nKey). ** ** If seekResult==0, that means pCur is pointing at some unknown location. ** In that case, this routine must seek the cursor to the correct insertion ** point for (pKey,nKey) before doing the insertion. For index btrees, ** if pX->nMem is non-zero, then pX->aMem contains pointers to the unpacked ** key values and pX->aMem can be used instead of pX->pKey to avoid having ** to decode the key.
64985. True if this is likely to be an append
64986. ** Check that: ** ** 1) All leaves of pSeg between iFirst and iLast (inclusive) exist and ** contain zero terms. ** 2) All leaves of pSeg between iNoRowid and iLast (inclusive) exist and ** contain zero rowids.
64987. ** Implementation of SQLite REGEXP operator. This scalar function takes ** two arguments. The first is a regular expression pattern to compile ** the second is a string to match against that pattern. If either ** argument is an SQL NULL, then NULL is returned. Otherwise, the result ** is 1 if the string matches the pattern, or 0 otherwise. ** ** SQLite maps the regexp() function to the regexp() operator such ** that the following two are equivalent: ** ** zString REGEXP zPattern
 ** regexp(zPattern, zString) ** ** Uses the following ICU regexp APIs: ** ** uregex_open() ** uregex_matches() ** uregex_close()
64988. This block calculates the checksum according to the FTS index.
64989. Space for values for UPDATE inversion
64990. Able to support an IN operator
64991. Entries in aFrame[] array
64992. If non-zero, replace first 4 bytes with this value
64993. index entries counter
64994. **comment:** ** The iterator object passed as the second argument currently contains ** no valid values except for the Fts5SegIter.pLeaf member variable. This ** function searches the leaf page for a term matching (pTerm/nTerm). ** ** If the specified term is found on the page, then the iterator is left ** pointing to it. If argument bGe is zero and the term is not found, ** the iterator is left pointing at EOF. ** ** If bGe is non-zero and the specified term is not found, then the ** iterator is left pointing to the smallest term in the segment that ** is larger than the specified term, even if this term is not on the ** current page.
label: code-design

64995. Max tokens per snippet
64996. The docid to be deleted
64997. CREATE
64998. Opcode: Function P1 P2 P3 P4 P5 ** Synopsis: r[P3]=func(r[P2@P5]) *** Invoke a user function (P4 is a pointer to an sqlite3_context object that ** contains a pointer to the function to be run) with P5 arguments taken ** from register P2 and successors. The result of the function is stored ** in register P3. Register P3 must not be one of the function inputs. *** P1 is a 32-bit bitmask indicating whether or not each argument to the ** function was determined to be constant at compile time. If the first ** argument was constant then bit 0 of P1 is set. This is used to determine ** whether meta data associated with a user function argument using the ** sqlite3_set_auxdata() API may be safely retained until the next ** invocation of this opcode. *** SQL functions are initially coded as OP_Function0 with P4 pointing ** to a FuncDef object. But on first evaluation, the P4 operand is ** automatically converted into an sqlite3_context object and the operation ** changed to this OP_Function opcode. In this way, the initialization of ** the sqlite3_context object occurs only once, rather than once for each ** evaluation of the function. *** See also: Function0, AggStep, AggFinal
64999. Library used incorrectly
65000. b: p3<<28 | p5<<14 | p7 (unmasked)
65001. The format field code letter
65002. IMP: R-51513-12026 The last_insert_rowid() SQL function is a ** wrapper around the sqlite3_last_insert_rowid() C/C++ interface ** function.
65003. ** Return the value of the integer key or "rowid" for a table btree. ** This routine is only valid for a cursor that is pointing into a ** ordinary table btree. If the cursor points to an index btree or ** is invalid, the result of this routine is undefined.
65004. languageid=?
65005. Content to be checksummed
65006. Number of columns in zTab
65007. ** Return a pointer to payload information from the entry that the ** pCur cursor is pointing to. The pointer is to the beginning of ** the key if index btrees (pPage->intKey==0) and is the data for ** table btrees (pPage->intKey==1). The number of bytes of available ** key/data is written into *pAmt. If *pAmt==0, then the value ** returned will not be a valid pointer. *** This routine is an optimization. It is common for the entire key ** and data to fit on the local page and for there to be no overflow ** pages. When that is so, this routine can be used to access the ** key and data without making a copy. If the key and/or data spills ** onto overflow pages, then accessPayload() must be used to reassemble ** the key/data and copy it into a preallocated buffer. *** The pointer returned by this routine looks directly into the cached ** page of the database. The data might change or move the next time ** any btree routine is called.
65008. Number of databases with an active write-transaction ** that are candidates for a two-phase commit using a ** master-journal
65009. The bottom of the main insertion loop, if the data source ** is a SELECT statement.
65010. ** Add a whole list of operations to the operation stack. Return a ** pointer to the first operation inserted. *** Non-zero P2 arguments to jump instructions are automatically adjusted ** so that the jump target is relative to the first operation inserted.
65011. Number of prior equality constraints on same index
65012. If iPtr is another freeblock (that is, if iPtr is not the freelist ** pointer in the page header) then check to see if iStart should be ** coalesced onto the end of iPtr.
65013. A Btree handle ** ** A database connection contains a pointer to an instance of ** this object for every database file that it has open. This structure ** is opaque to the database connection. The database connection cannot ** see the internals of this structure and only deals with pointers to ** this structure. *** For some database files, the same underlying database cache might be ** shared between multiple connections. In that case, each connection ** has its own instance of this object. But each instance of this object ** points to the same BtShared object. The database cache and the ** schema associated with the database file are all contained within ** the BtShared object. *** All fields in this structure are accessed under sqlite3.mutex. ** The pBt pointer itself may not be changed while there exists cursors ** in the referenced BtShared that point back to this Btree since those ** cursors have to go through this Btree to find their BtShared and ** they often do so without holding sqlite3.mutex.
65014. Pointer to current file mapping
65015. unlock failed with an error
65016. ** This function is a wrapper around the OS specific implementation of ** sqlite3_os_init(). The purpose of the wrapper is to provide the ** ability to simulate a malloc failure, so that the handling of an ** error in sqlite3_os_init() by the upper layers can be tested.
65017. If there are now no dirty pages in the cache, set eCreate to 2. ** This is an optimization that allows sqlite3PcacheFetch() to skip ** searching for a dirty page to eject from the cache when it might ** otherwise have to.
65018. Check to see if we should honor DESC requests on index columns
65019. Pointer to sqlite3_int64 variable
65020. Average size of database rows, in pages
65021. The SELECT statement with the ORDER BY clause
65022. ** Set the EP_FromJoin property on all terms of the given expression. ** And set the Expr.iRightJoinTable to iTable for every term in the ** expression. *** ** The EP_FromJoin property is used on terms of an expression to tell ** the LEFT OUTER JOIN processing logic that this term is part of the ** join restriction specified in the ON or USING clause and not a part ** of the more general WHERE clause. These terms are moved over to the ** WHERE clause during join processing but we need to remember that they ** originated in the ON or USING clause. *** ** The Expr.iRightJoinTable tells the WHERE clause processing that the ** expression depends on table iRightJoinTable even if that table is not ** explicitly mentioned in the expression. That information is needed ** for cases like this: *** ** SELECT * FROM t1 LEFT JOIN t2 ON t1.a=t2.b AND t1.x=5 ** ** The where clause needs to defer the handling of the t1.x=5 ** term until after the t2 loop of the join. In that way, a ** NULL t2 row will be inserted whenever t1.x!=5. If we do not ** defer the handling of t1.x=5, it will be processed immediately ** after the t1 loop and rows with t1.x!=5 will never appear in ** the output, which is incorrect.
65023. 324
65024. .eSubtype =
65025. ** Create a new cursor for the BTREE whose root is on the page ** iTable. If a read-only cursor is requested, it is assumed that ** the caller already has at least a read-only transaction open ** on the database already. If a write-cursor is requested, then ** the caller is assumed to have an open write transaction. *** ** If the BTREE_WRCR bit of wrFlag is clear, then the cursor can only ** be used for reading. If the BTREE_WRCR bit is set, then the cursor ** can be used for reading or for writing if other conditions for writing ** are also met. These are the conditions that must be met in order ** for writing to be allowed: *** ** 1: The cursor must have been opened with wrFlag containing BTREE_WRCR *** ** 2: Other database connections that share the same pager cache ** but which are not in the READ_UNCOMMITTED state may not have ** cursors open with wrFlag==0 on the same table. Otherwise ** the changes made by this write cursor would be visible to ** the read cursors in the other database connection. *** ** 3: The database must be writable (not on read-only media) *** ** 4: There must be an active transaction. *** ** The BTREE_FORDELETE bit of wrFlag may optionally be set if BTREE_WRCR ** is set. If FORDELETE is set, that is a hint to the implementation that ** this cursor will only be used to seek to and delete entries of an index ** as part of a larger DELETE statement. The FORDELETE hint is not used by ** this implementation. But in a hypothetical alternative storage engine ** in which index entries are automatically deleted when corresponding table ** rows are deleted, the FORDELETE flag is a hint that all SEEK and DELETE ** operations on this cursor can be no-ops and all READ operations can ** return a null row (2-bytes: 0x01 0x00). *** ** No checking is done to make sure that page iTable really is the ** root page of a b-tree. If it is not, then the cursor acquired ** will not work correctly. *** ** It is assumed that the sqlite3BtreeCursorZero() has been called ** on pCur to initialize the memory space prior to invoking this routine.
65026. Address of the OP_SorterOpen or OP_OpenEphemeral
65027. ** Append a message to the error message string.
65028. First argument to xLog()
65029. Truncate WAL to this size upon reset
65030. ** Print the value of a register for tracing purposes:
65031. **comment:** The next four values are not used by PRAGMAs or by sqlite3_dbconfig() and ** could be factored out into a separate bit vector of the sqlite3 object.
label: code-design
65032. 0x0f
65033. The pointer map page
65034. 14
65035. Generate a coroutine to evaluate the SELECT statement on ** the right - the "B" select
65036. 33
65037. At this point it must be known if the %_stat table exists or not. ** So bHasStat may not be 2.
65038. Store the page number here

65039. ** If the argument is not NULL, it points to a Wal object that holds a ** read-lock. This function returns the database page-size if it is known, ** or zero if it is not (or if pWal is NULL).

65040. WITHOUT => ID

65041. Restore the pRec->nField value before returning.

65042. Used in COMDB2 but not native SQLite

65043. The table already exists. If zWhere is not NULL, delete all entries ** associated with the table zWhere. If zWhere is NULL, delete the ** entire contents of the table.

65044. Leaf on which the required cell resides

65045. EVIDENCE-OF: R-29356-02391 If the database uses a 65536-byte page size ** and the reserved space is zero (the usual value for reserved space) ** then the cell content offset of an empty page wants to be 65536. ** However, that integer is too large to be stored in a 2-byte unsigned ** integer, so a value of 0 is used in its place.

65046. First byte of the locking range

65047. First arg to pass to pApi functions

65048. uppercase equivalent to lowercase

65049. ASC

65050. Change the journal mode.

65051. ** A WhereTerm with eOperator==WO_AND has its u.pAndInfo pointer set to ** a dynamically allocated instance of the following structure.

65052. Either CHANGESET_DATA or CONFLICT

65053. Pager structure

65054. **comment:** A complete hash table is an instance of the following structure. ** The internals of this structure are intended to be opaque -- client ** code should not attempt to access or modify the fields of this structure ** directly. Change this structure only by using the routines below. ** However, some of the "procedures" and "functions" for modifying and ** accessing this structure are really macros, so we can't really make ** this structure opaque. ** ** All elements of the hash table are on a single doubly-linked list. ** Hash.first points to the head of this list. ** ** There are Hash.htsize buckets. Each bucket points to a spot in ** the global doubly-linked list. The contents of the bucket are the ** element pointed to plus the next .ht.count-1 elements in the list. ** ** Hash.htsize and Hash.ht may be zero. In that case lookup is done ** by a linear search of the global list. For small tables, the ** Hash.ht table is never allocated because if there are few elements ** in the table, it is faster to do a linear search than to manage ** the hash table.

label: code-design

65055. **comment:** If the SELECT statement has an ORDER BY clause, zero the ** iOrderByCol variables. These are set to non-zero when an ** ORDER BY term exactly matches one of the terms of the ** result-set. Since the result-set of the SELECT statement may ** have been modified or reordered, these variables are no longer ** set correctly. Since setting them is just an optimization, ** it's easiest just to zero them here.

label: code-design

65056. **comment:** ** CAPI3REF: Destroy a snapshot ** EXPERIMENTAL ** ** ^The [sqlite3_snapshot_free(P)] interface destroys [sqlite3_snapshot] P. ** The application must eventually free every [sqlite3_snapshot] object ** using this routine to avoid a memory leak. ** ** The [sqlite3_snapshot_free()] interface is only available when the ** SQLITE_ENABLE_SNAPSHOT compile-time option is used.

label: code-design

65057. SQLITE_MEMDEBUG

65058. Call below is a no-op for NDEBUG builds

65059. Sorter file to read from

65060. ** Merge all doclists in the TermSelect.aaOutput[] array into a single ** doclist stored in TermSelect.aaOutput[0]. If successful, delete all ** other doclists (except the aaOutput[0] one) and return SQLITE_OK. ** ** If an OOM error occurs, return SQLITE_NOMEM. In this case it is ** the responsibility of the caller to free any doclists left in the ** TermSelect.aaOutput[] array.

65061. Context passed to xToken()

65062. If there is a schema mismatch on the current table, proceed to the ** next change. A log message has already been issued.

65063. The expression list to be coded

65064. If all query phrases seen by fts3BestSnippet() are present in at least ** one of the nSnippet snippet fragments, break out of the loop.

65065. nearphrases ::= phrase

65066. significand

65067. ** End of interface to code in fts5_expr.c. *****

65068. Table for TK_COLUMN expressions.

65069. Column cache

65070. column i is unchanged if aiChng[i]<

65071. RHS of LIKE/GLOB operator

65072. ** This routine is called to increment the value of the database file ** change-counter, stored as a 4-byte big-endian integer starting at ** byte offset 24 of the pager file. The secondary change counter at ** 92 is also updated, as is the SQLite version number at offset 96. ** ** But this only happens if the pPager->changeCountDone flag is false. ** To avoid excess churning of page 1, the update only happens once. ** See also the pager_write_changecounter() routine that does an ** unconditional update of the change counters. ** ** If the isDirectMode flag is zero, then this is done by calling ** sqlite3PagerWrite() on page 1, then modifying the contents of the ** page data. In this case the file will be updated when the current ** transaction is committed. ** ** The isDirectMode flag may only be non-zero if the library was compiled ** with the SQLITE_ENABLE_ATOMIC_WRITE macro defined. In this case, ** if isDirect is non-zero, then the database file is updated directly ** by writing an updated version of page 1 using a call to the ** sqlite3OsWrite() function.

65073. Number of entries in aTerm[]

65074. Run a separate WHERE clause for each term of the OR clause. After ** eliminating duplicates from other WHERE clauses, the action for each ** sub-WHERE clause is to invoke the main loop body as a subroutine.

65075. Opcode: Destroy P1 P2 P3 * * * * Delete an entire database table or index whose root page in the database ** file is given by P1. ** ** The table being destroyed is in the main database file if P3==0. If ** P3==1 then the table to be clear is in the auxiliary database file ** that is used to store tables create using CREATE TEMPORARY TABLE. ** ** If AUTOVACUUM is enabled then it is possible that another root page ** might be moved into the newly deleted root page in order to keep all ** root pages contiguous at the beginning of the database. The former ** value of the root page that moved - its value before the move occurred - ** is stored in register P2. If no page movement was required (because the ** table being dropped was already the last one in the database) then a ** zero is stored in register P2. If AUTOVACUUM is disabled then a zero ** is stored in register P2. ** ** This opcode throws an error if there are any active reader VMs when ** it is invoked. This is done to avoid the difficulty associated with ** updating existing cursors when a root page is moved in an AUTOVACUUM ** database. This error is thrown even if the database is not an AUTOVACUUM ** db in order to avoid introducing an incompatibility between autovacuum ** and non-autovacuum modes. ** ** See also: Clear

65076. ** Estimate the number of rows that will be returned based on ** an IN constraint where the right-hand side of the IN operator ** is a list of values. Example: ** ** WHERE x IN (1,2,3,4) ** ** Write the estimated row count into *pnRow and return SQLITE_OK. ** If unable to make an estimate, leave *pnRow unchanged and return ** non-zero. ** ** This routine can fail if it is unable to load a collating sequence ** required for string comparison, or if unable to allocate memory ** for a UTF conversion required for comparison. The error is stored ** in the pParse structure.

65077. defined(SQLITE_WIN32_HAS_WIDE)

65078. An ordinary table or view name in the FROM clause

65079. Extra information if (eOperator& WO_AND)!=0

65080. Cache that currently owns this page

65081. ** This function is invoked by the parser to call the xConnect() method ** of the virtual table pTab. If an error occurs, an error code is returned ** and an error left in pParse. ** ** This call is a no-op if table pTab is not a virtual table.

65082. Result pgno must be greater than iMin

65083. Number of arguments to the module

65084. Opcode: Copy P1 P2 P3 * * * * Synopsis: r[P2@P3+1]=r[P1@P3+1] ** ** Make a copy of registers P1..P1+P3 into registers P2..P2+P3. ** ** This instruction makes a deep copy of the value. A duplicate ** is made of any string or blob constant. See also OP_SCopy.

65085. For assigning database names to pTable

65086. The Vdbe we are building

65087. ** This function implements the FTS3_MATCHINFO_LCS matchinfo() flag. *** If the call is successful, the longest-common-substring lengths for each ** column are written into the first nCol elements of the pInfo->aMatchinfo[] ** array before returning. SQLITE_OK is returned in this case. *** Otherwise, if an error occurs, an SQLite error code is returned and the ** data written to the first nCol elements of pInfo->aMatchinfo[] is ** undefined.

65088. Right side of comparison

65089. Error msg written here

65090. Path to state db (or NULL if zRbu)

65091. having_opt

65092. setlist ::= setlist COMMA nm EQ expr

65093. **comment:** Overwrite freed memory with the 0x55 bit pattern to verify that it is ** not used after being freed
label: code-design

65094. ** Return the total number of entries read from the %_data table by ** this connection since it was created.

65095. 10

65096. we could have it if we want it

65097. iOnceResetThreshold

65098. ** Adjust the highwater mark if necessary. ** The caller must hold the appropriate mutex.

65099. Source database name

65100. SELECT

65101. Launch a background thread for this operation

65102. Value is a string

65103. in case we need to retry the :auto: lock file - ** we should never get here except via the 'continue' call.

65104. Current WhereLoop being processed.

65105. zero or more SQLITE_TRACE flags

65106. ** The xDestroy() virtual table method.

65107. Set \$p to point to the next leaf in the tree of eType nodes

65108. SQLITE_ENABLE_SESSION && SQLITE_ENABLE_PREUPDATE_HOOK

65109. ** Return the register of pOp->p2 after first preparing it to be ** overwritten with an integer value.

65110. IMPLEMENTATION-OF: R-42059-47211 If the argument N is positive then the ** suggested cache size is set to N.

65111. ** The datatype ynVar is a signed integer, either 16-bit or 32-bit. ** Usually it is 16-bits. But if SQLITE_MAX_VARIABLE_NUMBER is greater ** than 32767 we have to make it 32-bit. 16-bit is preferred because ** it uses less memory in the Expr object, which is a big memory user ** in systems with lots of prepared statements. And few applications ** need more than about 10 or 20 variables. But some extreme users want ** to have prepared statements with over 32767 variables, and for them ** the option is available (at compile-time).

65112. ** Argument eType must be one of RBU_INSERT, RBU_DELETE, RBU_IDX_INSERT or ** RBU_IDX_DELETE. This function performs the work of a single ** sqlite3rbu_step() call for the type of operation specified by eType.

65113. True if decimal point should be shown

65114. Buffer containing leaf data

65115. Zero out anything following *ppList

65116. Opcode: SeekRowid P1 P2 P3 * * * Synopsis: intkey=r[P3] ** P1 is the index of a cursor open on an SQL table btree (with integer ** keys). If register P3 does not contain an integer or if P1 does not ** contain a record with rowid P3 then jump immediately to P2. ** Or, if P2 is 0, raise an SQLITE_CORRUPT error. If P1 does contain ** a record with rowid P3 then ** leave the cursor pointing at that record and fall through to the next ** instruction. *** The OP_NotExists opcode performs the same operation, but with OP_NotExists ** the P3 register must be guaranteed to contain an integer value. With this ** opcode, register P3 might not contain an integer. *** The OP_NotFound opcode performs the same operation on index btrees ** (with arbitrary multi-value keys). *** This opcode leaves the cursor in a state where it cannot be advanced ** in either direction. In other words, the Next and Prev opcodes will ** not work following this opcode. *** See also: Found, NotFound, NoConflict, SeekRowid

65117. SQLITE_ALTER_TABLE

65118. ** Many system calls are accessed through pointer-to-functions so that ** they may be overridden at runtime to facilitate fault injection during ** testing and sandboxing. The following array holds the names and pointers ** to all overrideable system calls.

65119. Number of extra registers to allocate

65120. **comment:** ** Generate the beginning of the loop used for WHERE clause processing. ** The return value is a pointer to an opaque structure that contains ** information needed to terminate the loop. Later, the calling routine ** should invoke sqlite3WhereEnd() with the return value of this function ** in order to complete the WHERE clause processing. ** If an error occurs, this routine returns NULL. ** The basic idea is to do a nested loop, one loop for each table in ** the FROM clause of a select. (INSERT and UPDATE statements are the ** same as a SELECT with only a single table in the FROM clause.) For ** example, if the SQL is this: ** ** SELECT * FROM t1, t2, t3 WHERE ...; ** ** Then the code generated is conceptually like the following: ** ** foreach row1 in t1 do \
Code generated ** foreach row2 in t2 do -- by sqlite3WhereBegin() ** foreach row3 in t3 do / ** ... ** end \ Code generated ** end |-- by sqlite3WhereEnd() ** end / ** ** Note that the loops might not be nested in the order in which they ** appear in the FROM clause if a different order is better able to make ** use of indices. Note also that when the IN operator appears in ** the WHERE clause, it might result in additional nested loops for ** scanning through all values on the right-hand side of the IN. ** ** There are Btree cursors associated with each table. t1 uses cursor ** number pTabList->a[0].iCursor. t2 uses the cursor pTabList->a[1].iCursor. ** And so forth. This routine generates code to open those VDBE cursors ** and sqlite3WhereEnd() generates the code to close them. ** ** The code that sqlite3WhereBegin() generates leaves the cursors named ** in pTabList pointing at their appropriate entries. The [...] code ** can use OP_Column and OP_Rowid opcodes on these cursors to extract ** data from the various tables of the loop. ** ** If the WHERE clause is empty, the foreach loops must each scan their ** entire tables. Thus a three-way join is an O(N^3) operation. But if ** the tables have indices and there are terms in the WHERE clause that ** refer to those indices, a complete table scan can be avoided and the ** code will run much faster. Most of the work of this routine is checking ** to see if there are indices that can be used to speed up the loop. ** ** Terms of the WHERE clause are also used to limit which rows actually ** make it to the "..." in the middle of the loop. After each "foreach", ** terms of the WHERE clause that use only terms in that loop and outer ** loops are evaluated and if false a jump is made around all subsequent ** inner loops (or around the "..." if the test occurs within the inner- ** most loop) *** OUTER JOINS ** ** An outer join of tables t1 and t2 is conceptually coded as follows: ** ** foreach row1 in t1 do ** flag = 0 ** foreach row2 in t2 do ** start: ** ... ** flag = 1 ** end ** if flag==0 then ** move the row2 cursor to a null row ** goto start ** fi ** end ** ** ORDER BY CLAUSE PROCESSING ** ** pOrderBy is a pointer to the ORDER BY clause (or the GROUP BY clause ** if the WHERE_GROUPBY flag is set in wctrlFlags) of a SELECT statement ** if there is one. If there is no ORDER BY clause or if this routine ** is called from an UPDATE or DELETE statement, then pOrderBy is NULL. ** ** The iIdxCur parameter is the cursor number of an index. If ** WHERE_OR_SUBCLAUSE is set, iIdxCur is the cursor number of an index ** to use for OR clause processing. The WHERE clause should use this ** specific cursor. If WHERE_ONEPASS_DESIRED is set, then iIdxCur is ** the first cursor in an array of cursors for all indices. iIdxCur should ** be used to compute the appropriate cursor depending on which index is ** used.
label: code-design

65121. **comment:** First unused byte of aSpace1[]
label: code-design

65122. Step 1.

65123. ** pExpr is a CHECK constraint on a row that is being UPDATE-ed. The ** only columns that are modified by the UPDATE are those for which ** aiChng[i]>=0, and also the ROWID is modified if chngRowid is true. ** ** Return true if CHECK constraint pExpr does not use any of the ** changing columns (or the rowid if it is changing). In other words, ** return true if this CHECK constraint can be skipped when validating ** the new row in the UPDATE statement.

65124. sclp

65125. Name of tokenizer

65126. **comment:** FTS5INDEX_QUERY_XXX flags
label: code-design

65127. Columns in the other table

65128. LHS fields are not reordered

65129. ** Flags appropriate for the wctrlFlags parameter of sqlite3WhereBegin() ** and the WhereInfo.wctrlFlags member. *** Value constraints (enforced via assert()): ** WHERE_USE_LIMIT == SF_FixedLimit

65130. ***** End of fts3_snippet.c *****

65131. Ephemeral table holding all primary key values

65132. ** An rtree virtual-table object.
65133. Search strategy (see below)
65134. pPage is a leaf node. This loop navigates the cursor so that it ** points to the first interior cell that it points to the parent of ** the next page in the tree that has not yet been visited. The ** pCur->aiIdx[pCur->iPage] value is set to the index of the parent cell ** of the page, or to the number of cells in the page if the next page ** to visit is the right-child of its parent. *** If all pages in the tree have been visited, return SQLITE_OK to the ** caller.
65135. 278
65136. increment the cell count
65137. TEXT
65138. Non-zero if an error has occurred
65139. Number of PMAs to read
65140. Bytes of prefix compression for term
65141. Newly allocated session object
65142. xRename
65143. Table containing the REFERENCES clause (aka: Child)
65144. Name of the table into which we insert
65145. If either of the sqlite3_blob_open() or sqlite3_blob_reopen() calls ** above returned SQLITE_ERROR, return SQLITE_CORRUPT_VTAB instead. ** All the reasons those functions might return SQLITE_ERROR - missing ** table, missing row, non-blob/text in block column - indicate ** backing store corruption.
65146. ** This section provides definitions to allow the ** FTS3 extension to be compiled outside of the ** amalgamation.
65147. New object
65148. Counter variable
65149. 220
65150. ** Macros determining the rowids used by segment leaves and dlidx leaves ** and nodes. All nodes and leaves are stored in the %_data table with large ** positive rowids. *** Each segment has a unique non-zero 16-bit id. *** The rowid for each segment leaf is found by passing the segment id and ** the leaf page number to the FTS5_SEGMENT_ROWID macro. Leaves are numbered ** sequentially starting from 1.
65151. 27
65152. synopsis: r[P2]=parameter(P1,P4)
65153. **comment:** ** The pointer map is a lookup table that identifies the parent page for ** each child page in the database file. The parent page is the page that ** contains a pointer to the child. Every page in the database contains ** 0 or 1 parent pages. (In this context 'database page' refers ** to any page that is not part of the pointer map itself.) Each pointer map ** entry consists of a single byte 'type' and a 4 byte parent page number. ** The PTRMAP_XXX identifiers below are the valid types. *** The purpose of the pointer map is to facilitate moving pages from one ** position in the file to another as part of autovacuum. When a page ** is moved, the pointer in its parent must be updated to point to the ** new location. The pointer map is used to locate the parent page quickly. ***
PTRMAP_ROOTPAGE: The database page is a root-page. The page-number is not ** used in this case. *** PTRMAP_FREEPAGE: The database page is an unused (free) page. The page-number ** is not used in this case. *** PTRMAP_OVERFLOW1: The database page is the first page in a list of ** overflow pages. The page number identifies the page that ** contains the cell with a pointer to this overflow page. *** PTRMAP_OVERFLOW2: The database page is the second or later page in a list of ** overflow pages. The page-number identifies the previous ** page in the overflow page list. *** PTRMAP_BTREE: The database page is a non-root btree page. The page number ** identifies the parent page in the btree.
label: code-design
65154. The following line of code (and the "p++" below the while() loop) is ** normally all that is required to move pointer p to the desired ** position. The exception is if this node is being loaded from disk ** incrementally and pointer "p" now points to the first byte past ** the populated part of pReader->aNode[].
65155. synopsis: iDb=P1 root=P2 write=P3
65156. ** INSERT STACK UNION HERE **
65157. Maximum Sorter PMA size
65158. SQLITE_UTF8, SQLITE_UTF16BE, SQLITE_UTF16LE
65159. If the file exists, it fails.
65160. VdbeCursor row cache generation counter
65161. First register in a range holding values to insert
65162. Size in bytes of each node in the node table
65163. Finish committing the transaction to the destination database.
65164. (310) nmnum ::= nm (OPTIMIZED OUT)
65165. ** strftime(FORMAT, Timestring, MOD, MOD, ...) ** ** Return a string described by FORMAT. Conversions as follows: ** ** %d day of month ** %f ** fractional seconds SS.SSS ** %H hour 00-24 ** %j day of year 000-366 ** %J ** Julian day number ** %m month 01-12 ** %M minute 00-59 ** %s seconds since 1970-01-01 ** %S seconds 00-59 ** %w day of week 0-6 sunday==0 ** %W week of year 00-53 ** %Y year 0000-9999 ** %% %
65166. If this page has already been played back before during the current ** rollback, then don't bother to play it back again.
65167. Position-list
65168. 246
65169. Space for copies of dividers cells
65170. Find the next row in the Queue and output that row
65171. Insert the marker in the %_segments table to make sure nobody tries ** to steal the space just allocated. This is also used to identify ** appendable segments.
65172. start_block
65173. ccons ::= UNIQUE onconf
65174. Null
65175. If this is an UPDATE or DELETE, read the old.* record.
65176. IMP: R-31676-45509 If X is the integer -9223372036854775808 ** then abs(X) throws an integer overflow error since there is no ** equivalent positive 64-bit two complement value.
65177. ***** Interface to code in fts5_tokenizer.c.
65178. 125
65179. ** This function allocates a new level iLevel index in the segdir table. ** Usually, indexes are allocated within a level sequentially starting ** with 0, so the allocated index is one greater than the value returned ** by: *** SELECT max(idx) FROM %_segdir WHERE level = :iLevel ** ** However, if there are already FTS3_MERGE_COUNT indexes at the requested ** level, they are merged into a single level (iLevel+1) segment and the ** allocated index is 0. ** ** If successful, *piIdx is set to the allocated index slot and SQLITE_OK ** returned. Otherwise, an SQLite error code is returned.
65180. UTF16 with high-order bytes non-zero
65181. rowid = ? expression (or NULL)
65182. **comment:** Final cleanup
label: code-design
65183. Register for rowset of rows to delete
65184. Number of pages left to copy
65185. Used when p4type is P4_INTARRAY
65186. The database connection whose status is desired
65187. **comment:** ** Change the "soft" limit on the number of pages in the cache. ** Unused and unmodified pages will be recycled when the number of ** pages in the cache exceeds this soft limit. But the size of the ** cache is allowed to grow larger than this limit if it contains ** dirty pages or pages still in active use.
label: code-design
65188. xEof
65189. Init nFree to non-freeblock free space
65190. 128
65191. True if read-only
65192. This routine is just a convenient place to set a breakpoint that will ** fire after each opcode is inserted and displayed using ** "PRAGMA vdbe_addoptrace=on".
65193. If control flows to this point, then it was not possible to add the ** page being freed as a leaf page of the first trunk in the free-list. ** Possibly because the free-list is empty, or possibly because the ** first trunk in the free-list is full. Either way, the page being freed ** will become the new first trunk page in the free-list.

65194. Array of nodes containing the parse
65195. The opcode to be commented
65196. ** When this function is called, *ppPoslist is assumed to point to the ** start of a position-list. After it returns, *ppPoslist points to the ** first byte after the position-list. *** A position list is list of positions (delta encoded) and columns for ** a single document record of a doclist. So, in other words, this ** routine advances *ppPoslist so that it points to the next docid in ** the doclist, or to the first byte past the end of the doclist. *** If pp is not NULL, then the contents of the position list are copied ** to *pp. *pp is set to point to the first byte past the last byte copied ** before this function returns.
65197. Overwrite the old cell with the new if they are the same size. ** We could also try to do this if the old cell is smaller, then add ** the leftover space to the free list. But experiments show that ** doing that is no faster than skipping this optimization and just ** calling dropCell() and insertCell(). *** This optimization cannot be used on an autovacuum database if the ** new entry uses overflow pages, as the insertCell() call below is ** necessary to add the PTRMAP_OVERFLOW1 pointer-map entry.
65198. Figure out the size of the output
65199. Restore the original value of db->flags
65200. A pending-terms seg-reader for an FTS4 table that uses order=desc. ** Pending-terms doclists are always built up in ascending order, so ** we have to iterate through them backwards here.
65201. ** Combinations of two or more EP_* flags
65202. ** CAPI3REF: Fundamental Datatypes ** KEYWORDS: SQLITE_TEXT *** ^ (Every value in SQLite has one of five fundamental datatypes: ** ** ** 64-bit signed integer ** 64-bit IEEE floating point number ** string ** BLOB ** NULL **)*** These constants are codes for each of those types. *** Note that the SQLITE_TEXT constant was also used in SQLite version 2 ** for a completely different meaning. Software that links against both ** SQLite version 2 and SQLite version 3 should use SQLITE3_TEXT, not ** SQLITE_TEXT.
65203. Do any ON CASCADE, SET NULL or SET DEFAULT operations required to ** handle rows (possibly in other tables) that refer via a foreign key ** to the row just deleted.
65204. Set if thread is finished but not joined
65205. The complete WHERE clause
65206. Array to sort entries of
65207. '@', '#', '!'. Alphabetic SQL variables
65208. Now check that the iter.nEmpty leaves following the current leaf ** (a) exist and (b) contain no terms.
65209. ** End of the special code for wince *****
65210. The aArg[] array needs to grow.
65211. Hash of the name
65212. An instance of the IdxExprTrans object carries information about a ** mapping from an expression on table columns into a column in an index ** down through the Walker.
65213. If we reach this point, that means it is OK to perform the transformation
65214. Cell pCell is destined for new sibling page pNew. Originally, it ** was either part of sibling page iOld (possibly an overflow cell), ** or else the divider cell to the left of sibling page iOld. So, ** if sibling page iOld had the same page number as pNew, and if ** pCell really was a part of sibling page iOld (not a divider or ** overflow cell), we can skip updating the pointer map entries.
65215. Deal with as much of this write request as possible by transferring ** data from the memory mapping using memcpy().
65216. **comment:** TODO: Check that zTarget and zRbu are non-NULL
 label: requirement
65217. If not NULL, info on how to process DISTINCT
65218. Table we are trying to access
65219. Set up appropriate ctrlFlags
65220. **comment:** ** This function is called when initializing an FTS4 table that uses the ** content=xxx option. It determines the number of and names of the columns ** of the new FTS4 table. *** The third argument passed to this function is the value passed to the ** config=xxx option (i.e. "xxx"). This function queries the database for ** a table of that name. If found, the output variables are populated ** as follows: *** *pnCol: Set to the number of columns table xxx has, *** *pnStr: Set to the total amount of space required to store a copy ** of each columns name, including the nul-terminator. *** *pazCol: Set to point to an array of *pnCol strings. Each string is ** the name of the corresponding column in table xxx. The array ** and its contents are allocated using a single allocation. It ** is the responsibility of the caller to free this allocation ** by eventually passing the *pazCol value to sqlite3_free(). *** If the table cannot be found, an error code is returned and the output ** variables are undefined. Or, if an OOM is encountered, SQLITE_NOMEM is ** returned (and the output variables are undefined).
 label: code-design
65221. True if rowid is UPDATED
65222. ** Return the VDBE address or label to jump to in order to continue ** immediately with the next row of a WHERE clause.
65223. Numeric type of right operand
65224. Not an index on expressions
65225. ** Expand the space allocated for the given SrcList object by ** creating nExtra new slots beginning at iStart. iStart is zero based. ** New slots are zeroed. *** For example, suppose a SrcList initially contains two entries: A,B. ** To append 3 new entries onto the end, do this: *** sqlite3SrcListEnlarge(db, pSrcList, 3, 2); *** After the call above it would contain: A, B, nil, nil, nil. ** If the iStart argument had been 1 instead of 2, then the result ** would have been: A, nil, nil, nil, B. To prepend the new slots, ** the iStart value would be 0. The result then would ** be: nil, nil, nil, A, B. *** If a memory allocation fails the SrcList is unchanged. The ** db->mallocFailed flag will be set to true.
65226. WHERE analyse context
65227. ***** End of fts3_tokenizer.h *****
65228. selcollist ::= sclp nm DOT STAR
65229. SQLITE_OMIT_CAST
65230. Number of virtual machine steps
65231. File handle
65232. Offset of first byte to lock
65233. For nVector==1, combine steps 6 and 7 by immediately returning ** FALSE if the first comparison is not NULL
65234. 145
65235. Size of azCol[] and abPK[] arrays
65236. Create/open the named mutex
65237. Type of value (SQLITE_NULL, TEXT etc.)
65238. ** Delete a "porter" tokenizer.
65239. Only the admin user is allowed to know that the sqlite_user table ** exists
65240. This is the top of the main insertion loop
65241. Index of %_segdir entry to delete
65242. Find the start of the next token.
65243. EVIDENCE-OF: R-62920-47450 The busy-handler callback is never invoked ** in the SQLITE_CHECKPOINT_PASSIVE mode.
65244. Generate a single row of result
65245. Position in FROM clause of table for this loop
65246. fall through to real
65247. 266
65248. ** This routine is called to create a new foreign key on the table ** currently under construction. pFromCol determines which columns ** in the current table point to the foreign key. If pFromCol==0 then ** connect the key to the last column inserted. pTo is the name of ** the table referred to (a.k.a the "parent" table). pToCol is a list ** of tables in the parent pTo table. flags contains all ** information about the conflict resolution algorithms specified ** in the ON DELETE, ON UPDATE and ON INSERT clauses. *** An FKey structure is created and added to the table currently ** under construction in the pParse->pNewTable field. ** ** The foreign key is set for IMMEDIATE processing. A subsequent call ** to sqlite3DeferForeignKey() might change this to DEFERRED.
65249. Delete the cell in question from the leaf node.
65250. ** Rtree virtual table module xEof method. *** Return non-zero if the cursor does not currently point to a valid ** record (i.e if the scan has finished), or zero otherwise.

65251. LP => nothing
65252. ** Retrieve a pointer to a static mutex or allocate a new dynamic one.
65253. sortlist ::= expr sortorder
65254. Do not delete from this cursor
65255. raisetype ::= ABORT
65256. Log2 of iFullSz/POW2_MIN
65257. Delete everything from the shadow tables. Except, leave %_content as ** is if bContent is false.
65258. Begin the database scan. *** Do not consider a single-pass strategy for a multi-row update if ** there are any triggers or foreign keys to process, or rows may ** be deleted as a result of REPLACE conflict handling. Any of these ** things might disturb a cursor being used to scan through the table ** or index, causing a single-pass approach to malfunction.
65259. refact ::= CASCADE
65260. The parsing context. Error messages written here
65261. Opcode: Last P1 P2 P3 * * * * * The next use of the Rowid or Column or Prev instruction for P1 ** will refer to the last entry in the database table or index. ** If the table or index is empty and P2>0, then jump immediately to P2. ** If P2 is 0 or if the table or index is not empty, fall through ** to the following instruction.
*** This opcode leaves the cursor configured to move in reverse order, ** from the end toward the beginning. In other words, the cursor is ** configured to use Prev, not Next. *** If P3 is -1, then the cursor is positioned at the end of the btree ** for the purpose of appending a new entry onto the btree. In that ** case P2 must be 0. It is assumed that the cursor is used only for ** appending and so if the cursor is valid, then the cursor must already ** be pointing at the end of the btree and so no changes are made to ** the cursor.
65262. Allocated size of aDlidx[] array
65263. ** If the source-list item passed as an argument was augmented with an ** INDEXED BY clause, then try to locate the specified index. If there ** was such a clause and the named index cannot be found, return ** SQLITE_ERROR and leave an error in pParse. Otherwise, populate ** pFrom->pIndex and return SQLITE_OK.
65264. multiselect_op ::= UNION ALL
65265. ** Given an expression list, generate a KeyInfo structure that records ** the collating sequence for each expression in that expression list. *** If the ExprList is an ORDER BY or GROUP BY clause then the resulting ** KeyInfo structure is appropriate for initializing a virtual index to ** implement that clause. If the ExprList is the result set of a SELECT ** then the KeyInfo structure is appropriate for initializing a virtual ** index to implement a DISTINCT test. *** Space to hold the KeyInfo structure is obtained from malloc. The calling ** function is responsible for seeing that this structure is eventually ** freed.
65266. Token to defer
65267. ** Buffer zInput, length nInput, contains the contents of a quoted string ** that appeared as part of an fts3 query expression. Neither quote character ** is included in the buffer. This function attempts to tokenize the entire ** input buffer and create an Fts3Expr structure of type FTSQUERY_PHRASE ** containing the results. *** If successful, SQLITE_OK is returned and *ppExpr set to point at the ** allocated Fts3Expr structure. Otherwise, either SQLITE_NOMEM (out of memory ** error) or SQLITE_ERROR (tokenization error) is returned and *ppExpr set ** to 0.
65268. OUT: Error code. SQLITE_OK or ERROR
65269. Length of the keyword
65270. ** Return the affinity character for a single column of a table.
65271. First PmaReader to compare
65272. Register to hold temp table ROWID
65273. increment entry count
65274. Set up for cell analysis
65275. ** If *pRc is not SQLITE_OK, or if pExpr is not the root node of a NEAR ** cluster, then this function returns 1 immediately. *** Otherwise, it checks if the current row really does match the NEAR ** expression, using the data currently stored in the position lists ** (Fts3Expr->pPhrase.doclist.pList/nList) for each phrase in the expression. *** If the current row is a match, the position list associated with each ** phrase in the NEAR expression is edited in place to contain only those ** phrase instances sufficiently close to their peers to satisfy all NEAR ** constraints. In this case it returns 1. If the NEAR expression does not ** match the current row, 0 is returned. The position lists may or may not ** be edited if 0 is returned.
65276. 35
65277. Base cursor number
65278. Mask of shared locks held
65279. Create the custom VFS.
65280. ** This routine converts the sqlite3_pcachepage object returned by ** sqlite3PcacheFetch() into an initialized PgHdr object. This routine ** must be called after sqlite3PcacheFetch() in order to get a usable ** result.
65281. Values
65282. Amount of usable space on each page
65283. One of the COLNAME_* constants
65284. ** Create a "porter" tokenizer.
65285. 140
65286. 1010
65287. Number of bytes to zero in aPgno[]
65288. IMP: R-04922-24076 The sqlite_compileoption_get() SQL function ** is a wrapper around the sqlite3_compileoption_get() C/C++ function.
65289. ** Invoke open(). Do so multiple times, until it either succeeds or ** fails for some reason other than EINTR. *** If the file creation mode "m" is 0 then set it to the default for ** SQLite. The default is SQLITE_DEFAULT_FILE_PERMISSIONS (normally ** 0644) as modified by the system umask. If m is not 0, then ** make the file creation mode be exactly m ignoring the umask. *** The m parameter will be non-zero only when creating -wal, -journal, ** and -shm files. We want those files to have *exactly* the same ** permissions as their original database, unadulterated by the umask. ** In that way, if a database file is -rw-rw-rw or -rw-rw-r-, and a ** transaction crashes and leaves behind hot journals, then any ** process that is able to write to the database will also be able to ** recover the hot journals.
65290. Heap that might have been lookaside
65291. Shared memory instance
65292. Name of file to be deleted
65293. Wal to close
65294. True (1) if tz is valid
65295. select ::= with selectnowith
65296. Restriction 20
65297. Number of columns or fields in the key
65298. Make sure it is not a system table being altered, or a reserved name ** that the table is being renamed to.
65299. fts3_aux.c
65300. EVIDENCE-OF: R-07291-35328 A value of 5 (0x05) means the page is an ** interior table b-tree page.
65301. The only possible error from InitPage
65302. 192
65303. The Expr.x union is never used at the same time as Expr.pRight
65304. FROM clause must have exactly one term
65305. SQLITE_RANGE
65306. Name of VFS module to use
65307. ** End of code taken from fossil. *****
65308. Size of phrase in terms
65309. Call sqlite3_declare_vtab()
65310. **comment:** ** This function is used to parse both URIs and non-URI filenames passed by the ** user to API functions sqlite3_open() or sqlite3_open_v2(), and for database ** URIs specified as part of ATTACH statements. *** The first argument to this function is the name of the VFS to use (or ** a NULL to signify the default VFS) if the URI does not contain a "vfs=xxx" ** query parameter. The second argument contains the URI (or non-URI filename) ** itself. When this function is called the *pFlags variable should contain ** the default flags to open the database handle with. The value stored in *** *pFlags may be updated before

returning if the URI filename contains ** "cache=xxx" or "mode=xxx" query parameters. *** If successful, SQLITE_OK is returned. In this case *ppVfs is set to point to ** the VFS that should be used to open the database file. *pzFile is set to ** point to a buffer containing the name of the file to open. It is the ** responsibility of the caller to eventually call sqlite3_free() to release ** this buffer. *** If an error occurs, then an SQLite error code is returned and *pzErrMsg ** may be set to point to a buffer containing an English language error ** message. It is the responsibility of the caller to eventually release ** this buffer by calling sqlite3_free().

label: code-design

65311. For FTS5_STRING - cluster of phrases

65312. 3rd input operand

65313. Update a row in a main table b-tree

65314. Number of aMem[] value. Might be zero

65315. case_exprlist

65316. Column snippet is extracted from

65317. ** If pBt points to an empty file then convert that empty file ** into a new empty database by initializing the first page of ** the database.

65318. ** Translate UTF-8 to UTF-8. *** This has the effect of making sure that the string is well-formed ** UTF-8. Miscoded characters are removed. *** The translation is done in-place and aborted if the output ** overruns the input.

65319. SQLITE_AMALGAMATION

65320. ** Maximum supported symbolic links

65321. !defined(SQLITE_CORE) || defined(SQLITE_ENABLE_FTS5)

65322. 1470

65323. ** This function is called to handle the SQLITE_FCNTL_SIZE_HINT ** file-control operation. Enlarge the database to nBytes in size ** (rounded up to the next chunk-size). If the database is already ** nBytes or larger, this routine is a no-op.

65324. Pointer map page index

65325. If this is a prefix-search, and if the term that apSegment[0] points ** to does not share a suffix with pFilter->zTerm/nTerm, then all ** required callbacks have been made. In this case exit early. *** Similarly, if this is a search for an exact match, and the first term ** of segment apSegment[0] is not a match, exit early.

65326. porter rule condition: (m = 1)

65327. True for statements that do not write

65328. Part of the record being decoded

65329. Columns etc. for shadow table

65330. Size of array argv[]

65331. Create a cursor to hold the database open

65332. Number of "," to insert

65333. Adjust ancestry of this node.

65334. The next index associated with the same table

65335. trigger_cmd ::= UPDATE orconf trnm tridxby SET setlist where_opt

65336. Success code

65337. xCommit

65338. If the destination is DistFifo, then cursor (iParm+1) is open ** on an ephemeral index. If the current row is already present ** in the index, do not write it to the output. If not, add the ** current row to the index and proceed with writing it to the ** output table as well.

65339. ** Free an outstanding memory allocation.

65340. True to store descending indexes

65341. Sort data from this cursor

65342. Variables populated by statDecodePage():

65343. ** sqlite3WalkExpr() callback used by sqlite3ExprIsConstantOrGroupBy().

65344. The next line of code computes as follows, only faster: ** if(oc==OP_SeekGT || oc==OP_SeekLE){ ** r.default_rc = -1; ** }else{ ** r.default_rc = +1; ** }

65345. Number of hidden columns

65346. ** Return the type of the argument.

65347. Value is undefined

65348. Database in which module is registered

65349. Used when pWLoop->wsFlags&WHERE_IN_ABLE

65350. 1310

65351. Maximum number of samples to accumulate

65352. ** Search the first N tables in pSrc, from left to right, looking for a ** table that has a column named zCol. *** When found, set *piTab and *piCol to the table index and column index ** of the matching column and return TRUE. *** If not found, return FALSE.

65353. Split the master block. Return the tail.

65354. ** Return a pointer to the Table structure for the table that a trigger ** is set on.

65355. SQLITE_PROTOCOL

65356. pNew->aSeg[]

65357. 58..5f XYZ[\]^_

65358. Add the term to each of the prefix indexes that it is not too ** short for.

65359. When allocating a new Vdbe object, all of the fields below should be ** initialized to zero or NULL

65360. Name of the new SQL function

65361. ** Attempt to memory map file pFile. If successful, set *pp to point to the ** new mapping and return SQLITE_OK. If the mapping is not attempted ** (because the file is too large or the VFS layer is configured not to use ** mmap), return SQLITE_OK and set *pp to NULL. *** Or, if an error occurs, return an SQLite error code. The final value of ** *pp is undefined in this case.

65362. if not __APPLE__

65363. These conditions have already been verified in btreeInitPage() ** if PRAGMA cell_size_check=ON.

65364. Address of "OP_Rewind iIdxCur"

65365. **comment:** Any of the WHERE_COLUMN_xxx values

label: code-design

65366. If there are no aggregate functions in the result-set, and no GROUP BY ** expression, do not allow aggregates in any of the other expressions.

65367. ** SQLite calls this function immediately after a call to unixDlSym() or ** unixDlOpen() fails (returns a null pointer). If a more detailed error ** message is available, it is written to zBufOut. If no error message ** is available, zBufOut is left unmodified and SQLite uses a default ** error message.

65368. OUT: Values appended to the record

65369. The thread identifier

65370. ANALYZE

65371. Number of tokens in table

65372. PRAGMA index_xinfo = ?

65373. ***** End of vdbemem.c *****

65374. ** Implementation of the json_array(VALUE,...) function. Return a JSON ** array that contains all values given in arguments. Or if any argument ** is a BLOB, throw an error.

65375. Address of test for empty pDest

65376. **comment:** ** CAPI3REF: Attempt To Free Heap Memory *** ^The sqlite3_release_memory() interface attempts to free N bytes ** of heap memory by deallocating non-essential memory allocations ** held by the database library. Memory used to cache database ** pages to improve performance is an example of non-essential memory. ** ^sqlite3_release_memory() returns the number of bytes actually freed, ** which might be more or less than the amount requested. ** ^The sqlite3_release_memory() routine is a no-op returning zero ** if SQLite is not compiled with [SQLITE_ENABLE_MEMORY_MANAGEMENT]. *** See also: [sqlite3_db_release_memory()]

label: code-design

65377. True if "#" flag is present

65378. If true, create page if it does not exist already
65379. Minimum number of regions required to be mapped.
65380. Lookup key for the unixinodeInfo
65381. Check that the LHS of the comparison is a column reference to ** the right column of the right source table. And that the sort ** order of the index column is the same as the sort order of the ** leftmost index column.
65382. The conditions tested below might not be true ** in a corrupt database
65383. Buffer containing previous term on page
65384. The Queue table
65385. ** This function is called when deleting or updating a row to implement ** any required CASCADE, SET NULL or SET DEFAULT actions.
65386. Failed to find the required lock.
65387. Calculate nLeafEst.
65388. 5 TRIGGER:
65389. We know that the regSampleRowid row exists because it was read by ** the previous loop. Thus the not-found jump of seekOp will never ** be taken
65390. **comment:** ** CAPI3REF: Define New Collating Sequences ** METHOD: sqlite3 *** ^These functions add, remove, or modify a [collation] associated ** with the [database connection] specified as the first argument. ** ** ^The name of the collation is a UTF-8 string ** for sqlite3_create_collation() and sqlite3_create_collation_v2() ** and a UTF-16 string in native byte order for sqlite3_create_collation16(). ** ^Collation names that compare equal according to [sqlite3_strnicmp()] are ** considered to be the same name. ** ** ^The third argument (eTextRep) must be one of the constants: ** ** [SQLITE_UTF8], ** [SQLITE_UTF16LE], ** [SQLITE_UTF16BE], ** [SQLITE_UTF16], or ** [SQLITE_UTF16_ALIGNED]. ** ^ ** ^The eTextRep argument determines the encoding of strings passed ** to the collating function callback, xCallback. ** ^The [SQLITE_UTF16] and [SQLITE_UTF16_ALIGNED] values for eTextRep ** force strings to be UTF16 with native byte order. ** ^The [SQLITE_UTF16_ALIGNED] value for eTextRep forces strings to begin ** on an even byte address. ** ** ^The fourth argument, pArg, is an application data pointer that is passed ** through as the first argument to the collating function callback. ** ** ^The fifth argument, xCallback, is a pointer to the collating function. ** ^Multiple collating functions can be registered using the same name but ** with different eTextRep parameters and SQLite will use whichever ** function requires the least amount of data transformation. ** ^If the xCallback argument is NULL then the collating function is ** deleted. ^When all collating functions having the same name are deleted, ** that collation is no longer usable. ** ** ^The collating function callback is invoked with a copy of the pArg ** application data pointer and with two strings in the encoding specified ** by the eTextRep argument. The collating function must return an ** integer that is negative, zero, or positive ** if the first string is less than, equal to, or greater than the second, ** respectively. A collating function must always return the same answer ** given the same inputs. If two or more collating functions are registered ** to the same collation name (using different eTextRep values) then all ** must give an equivalent answer when invoked with equivalent strings. ** The collating function must obey the following properties for all ** strings A, B, and C: ** ** ** If A==B then B==A. ** If A==B and B==C then A==C. ** If A<>B THEN B>A. ** If A<>B and B<>C then A<>C. ** ** ** ^If a collating function fails any of the above constraints and that ** collating function is registered and used, then the behavior of SQLite ** is undefined. ** ** ^The sqlite3_create_collation_v2() works like sqlite3_create_collation() ** with the addition that the xDestroy callback is invoked on pArg when ** the collating function is deleted. ** ^Collating functions are deleted when they are overridden by later ** calls to the collation creation functions or when the ** [database connection] is closed using [sqlite3_close()]. ** ** ^The xDestroy callback is <u>not</u> called if the ** sqlite3_create_collation_v2() function fails. Applications that invoke ** sqlite3_create_collation_v2() with a non-NUL xDestroy argument should ** check the return code and dispose of the application data pointer ** themselves rather than expecting SQLite to deal with it for them. ** This is different from every other SQLite interface. The inconsistency ** is unfortunate but cannot be changed without breaking backwards ** compatibility. ** ** See also: [sqlite3_collation_needed()] and [sqlite3_collation_needed16()].
label: code-design
65391. ** CAPI3REF: Determine The Virtual Table Conflict Policy ** ** This function may only be called from within a call to the [xUpdate] method ** of a [virtual table] implementation for an INSERT or UPDATE operation. ^The ** value returned is one of [SQLITE_ROLLBACK], [SQLITE_IGNORE], [SQLITE_FAIL], ** [SQLITE_ABORT], or [SQLITE_REPLACE], according to the [ON CONFLICT] mode ** of the SQL statement that triggered the call to the [xUpdate] method of the ** [virtual table].
65392. Integrity checking context
65393. Generic pointer
65394. Table name
65395. Current write offset in the journal file
65396. Create the serialized output changeset based on the contents of the ** hash tables attached to the SessionTable objects in list p->pList.
65397. **comment:** ** NOTE: Windows CE is handled differently here due its lack of the Win32 ** API UnlockFile.
label: test
65398. True if doclist is loaded incrementally
65399. **comment:** ***** Begin Named Semaphore Locking ***** ** ** Named semaphore locking is only supported on VxWorks. ** ** Semaphore locking is like dot-lock and flock in that it really only ** supports EXCLUSIVE locking. Only a single process can read or write ** the database file at a time. This reduces potential concurrency, but ** makes the lock implementation much easier.
label: code-design
65400. An FTS3_EVAL_XX constant
65401. The use of a character not in [a-zA-Z] means that we fallback ** to the copy stemmer
65402. ** Unlink the chunk at index i from ** whatever list is currently a member of.
65403. Preserve the application id
65404. ** If X is a character that can be used in an identifier then ** IdChar(X) will be true. Otherwise it is false. ** ** For ASCII, any character with the high-order bit set is ** allowed in an identifier. For 7-bit characters, ** sqlite3IsIdChar[X] must be 1. ** ** For EBCDIC, the rules are more complex but have the same ** end result. ** ** Ticket #1066, the SQL standard does not allow '\$' in the ** middle of identifiers. But many SQL implementations do. ** SQLite will allow '\$' in identifiers for compatibility. ** But the feature is undocumented.
65405. ** Returns non-zero if the specified path name starts with a drive letter ** followed by a colon character.
65406. Cx
65407. Win32 last error
65408. Do not spill the cache when non-zero
65409. ** Enable or disable the shared pager and schema features. ** ** This routine has no effect on existing database connections. ** The shared cache setting effects only future calls to ** sqlite3_open(), sqlite3_open16(), or sqlite3_open_v2().
65410. Opcode: IfNullRow P1 P2 P3 ** ** Synopsis: if P1=nullRow then r[P3]=NULL, goto P2 ** ** Check the cursor P1 to see if it is currently pointing at a NULL row. ** If it is, then set register P3 to NULL and jump immediately to P2. ** If P1 is not on a NULL row, then fall through without making any ** changes.
65411. pTerm can be evaluated using just the index. So reduce ** the expected number of table lookups accordingly
65412. 48
65413. Statement changes (Vdbe.nChange)
65414. Index of constrained coordinate
65415. xCreate
65416. Origin of objects in apRankArg[]
65417. 90..97,
65418. ** The default value for the second argument to matchinfo().
65419. 239
65420. PAGER_GET_NOCONTENT or PAGER_GET_READONLY
65421. ** This routine implements the xFindFunction method for the FTS3 ** virtual table.
65422. Opcode: RowSetAdd P1 P2 ** ** ** Synopsis: rowset(P1)=r[P2] ** ** Insert the integer value held by register P2 into a RowSet object ** held in register P1. ** ** An assertion fails if P2 is not an integer.
65423. The P2 operand
65424. Initialize vdbe cursor object
65425. These verifications occurs for the main database only
65426. See if any siblings hold this same lock

65427. **comment:** ** Like realloc(). Resize an allocation previously obtained from ** sqlite3MemMalloc(). ** ** For this low-level interface, we know that pPrior!=0. Cases where ** pPrior==0 while have been intercepted by higher-level routine and ** redirected to xMalloc. Similarly, we know that nByte>0 because ** cases where nByte<=0 will have been intercepted by higher-level ** routines and redirected to xFree.
label: code-design

65428. Session object already attached to db
65429. If *pp1 is exactly nTokens before *pp2
65430. Above do not own any resources. Must free those below
65431. ** Set the P4 on the most recently added opcode to the KeyInfo for the ** index given.
65432. ***** End of random.c *****
65433. ** Close a connection to shared-memory. Delete the underlying ** storage if deleteFlag is true. ** ** If there is no shared memory associated with the connection then this ** routine is a harmless no-op.
65434. Called by SQLite to clean up pUser
65435. OP_Open** used to open bulk cursor
65436. ** The proxy locking style is intended for use with AFP filesystems. ** And since AFP is only supported on MacOSX, the proxy locking is also ** restricted to MacOSX. ** * ***** End of the proxy lock implementation *****

65437. OUT: Number of frames in WAL
65438. ** This is a public wrapper for the winUtf8ToMbc() function.
65439. Error information
65440. ** If SQLite is compiled to support shared-cache mode and to be threadsafe, ** this routine obtains the mutex associated with each BtShared structure ** that may be accessed by the VM passed as an argument. In doing so it also ** sets the BtShared.db member of each of the BtShared structures, ensuring ** that the correct busy-handler callback is invoked if required. ** ** If SQLite is not threadsafe but does support shared-cache mode, then ** sqlite3BtreeEnter() is invoked to set the BtShared.db variables ** of all of BtShared structures accessible via the database handle ** associated with the VM. ** ** If SQLite is not threadsafe and does not support shared-cache mode, this ** function is a no-op. ** ** The p->btreeMask field is a bitmask of all btrees that the prepared ** statement p will ever use. Let N be the number of bits in p->btreeMask ** corresponding to btrees that use shared cache. Then the runtime of ** this routine is N*N. But as N is rarely more than 1, this should not ** be a problem.
65441. SQLITE_FORMAT
65442. If the table being dropped is the table with the largest root-page ** number in the database, put the root page on the free list.
65443. Total size in bytes of journal file pJml
65444. Index of first cell to the left of right sibling
65445. Opcode: ResultRow P1 P2 * * * * Synopsis: output=[P1@P2] ** ** The registers P1 through P1+P2-1 contain a single row of ** results. This opcode causes the sqlite3_step() call to terminate ** with an SQLITE_ROW return code and it sets up the sqlite3_stmt ** structure to provide access to the r(P1)..r(P1+P2-1) values as ** the result row.
65446. KEY => ID
65447. For a two-way OR, attempt to implementation case 2.
65448. Special case: If db->aVTrans is NULL and db->nVTrans is greater ** than zero, then this function is being called from within a ** virtual module xSync() callback. It is illegal to write to ** virtual module tables in this case, so return SQLITE_LOCKED.
65449. Stat4Accum.anLt
65450. Input function
65451. ** The Win32 native heap cannot be modified because it may be in use.
65452. **comment:** If the second argument to this function is NULL, generate a ** temporary file name to use
label: code-design
65453. SQLITE_AMALGAMATION && SQLITE_BYTEORDER==0
65454. Virtual table object to return
65455. Dotfile locking uses the file path so it needs to be included in ** the dotlockLockingContext
65456. Callback implementation user data
65457. Hashing function for the aHash representation. ** Empirical testing showed that the *37 multiplier ** (an arbitrary prime)in the hash function provided ** no fewer collisions than the no-op *1.
65458. Array of cells
65459. OUT: List length pointer
65460. ** Must be first **
65461. A table that describes the SELECT results
65462. ABORT => ID
65463. This branch happens only when content is on overflow pages
65464. ** The journal file must be open when this function is called. ** ** This function is a no-op if the journal file has not been written to ** within the current transaction (i.e. if Pager.journalOff==0). ** ** If doTruncate is non-zero or the Pager.journalSizeLimit variable is ** set to 0, then truncate the journal file to zero bytes in size. Otherwise, ** zero the 28-byte header at the start of the journal file. In either case, ** if the pager is not in no-sync mode, sync the journal file immediately ** after writing or truncating it. ** ** If Pager.journalSizeLimit is set to a positive, non-zero value, and ** following the truncation or zeroing described above the size of the ** journal file in bytes is larger than this value, then truncate the ** journal file to Pager.journalSizeLimit bytes. The journal file does ** not need to be synced following this operation. ** ** If an IO error occurs, abandon processing and return the IO error code. ** Otherwise, return SQLITE_OK.
65465. ** Allocate nByte bytes of space using sqlite3Malloc(). If the ** allocation fails, call sqlite3_result_error_nomem() to notify ** the database handle that malloc() has failed and return NULL. ** If nByte is larger than the maximum string or blob length, then ** raise an SQLITE_TOOBIG exception and return NULL.
65466. The return value is stored here
65467. The Next opcode is only used after SeekGT, SeekGE, and Rewind. ** The Prev opcode is only used after SeekLT, SeekLE, and Last.
65468. oneselect ::= SELECT distinct selcollist from where_opt groupby_opt having_opt orderby_opt limit_opt
65469. eidlist
65470. ** Checkpoint database zDb.
65471. Offset in aData[] of current change
65472. in1, jump
65473. ** Generate code to extract the value of the iCol-th column of a table.
65474. Local size of all cells in apCell[]
65475. Iterate through the phrases in the expression to count them. The same ** callback makes sure the doclists are loaded for each phrase.
65476. Rowid for this leaf
65477. Back out changes but do no rollback transaction
65478. Most senior NEAR ancestor (or pExpr)
65479. ** Test the existence of or access permissions of file zPath. The ** test performed depends on the value of flags: ** ** SQLITE_ACCESS_EXISTS: Return 1 if the file exists ** SQLITE_ACCESS_READWRITE: Return 1 if the file is read and writable. ** SQLITE_ACCESS_READONLY: Return 1 if the file is readable. ** ** Otherwise return 0.
65480. Expr to test. May or may not be root.
65481. free the UTF8 buffer
65482. Skip the OP_IdxLt or OP_IdxGT that follows
65483. ** 2015 May 30 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Routines for varint serialization and deserialization.
65484. Deadlock detected.
65485. ** Specify the activation key for a CEROD database. Unless ** activated, none of the CEROD routines will work.
65486. Do not copy the destructor

65487. See above

65488. IN/OUT: Stucture of index

65489. *** whereLoopXfer() copies fields above *****

65490. Free the current contents of p->apValue[], if any.

65491. Prepared statement under construction

65492. b0..b7

65493. Format string

65494. pageFindSlot() has already verified that free blocks are sorted ** in order of offset within the page, and that no block extends ** past the end of the page. Provided the two free slots do not ** overlap, this guarantees that the memmove() calls below will not ** overwrite the usableSize byte buffer, even if the database page ** is corrupt.

65495. ** Populate the buffer zErrMsg (size nByte bytes) with a human readable ** utf-8 string describing the most recent error encountered associated ** with dynamic libraries.

65496. Want to do one-pass UPDATE/DELETE

65497. Flags that describe this loop

65498. Cursor used to iterate through aRight

65499. **comment:** The inability to allocates space for a larger hash table is ** a performance hit but it is not a fatal error. So mark the ** allocation as a benign. Use sqlite3Malloc()/memset(0) instead of ** sqlite3MallocZero() to make the allocation, as sqlite3MallocZero() ** only zeroes the requested number of bytes whereas this module will ** use the actual amount of space allocated for the hash table (which ** may be larger than the requested amount).

label: code-design

65500. Write error messages here

65501. ** This routine is called prior to sqlite3PagerCommit when a transaction ** is committed for an auto-vacuum database. ** ** If SQLITE_OK is returned, then *pnTrunc is set to the number of pages ** the database file should be truncated to during the commit process. ** i.e. the database has been reorganized so that only the first *pnTrunc ** pages are in use.

65502. ** Delete a "unicode61" tokenizer.

65503. Add the default collation sequence BINARY. BINARY works for both UTF-8 ** and UTF-16, so add a version for each to avoid any unnecessary ** conversions. The only error that can occur here is a malloc() failure. ** ** EVIDENCE-OF: R-52786-44878 SQLite defines three built-in collating ** functions:

65504. ** Extract the user data from a sqlite3_context structure and return a ** pointer to it.

65505. Name of the constraint currently being parsed

65506. Session object that owns SessionTable

65507. 0x30 .. 0x3F

65508. Keyword code

65509. The input value in P3 might be of any type: integer, real, string, ** blob, or NULL. But it needs to be an integer before we can do ** the seek, so convert it.

65510. ** CAPI3REF: Read Data From A BLOB Incrementally ** METHOD: sqlite3_blob *** ^This function is used to read data from an open [BLOB handle] into a ** caller-supplied buffer. N bytes of data are copied into buffer Z ** from the open BLOB, starting at offset iOffset.)^ ** ** ^If offset iOffset is less than N bytes from the end of the BLOB, ** [SQLITE_ERROR] is returned and no data is read. ^If N or iOffset is ** less than zero, [SQLITE_ERROR] is returned and no data is read. ** ^The size of the blob (and hence the maximum value of N+iOffset) ** can be determined using the [sqlite3_blob_bytes()] interface. ** ** ^An attempt to read from an expired [BLOB handle] fails with an ** error code of [SQLITE_ABORT]. ** ** ^On success, sqlite3_blob_read() returns SQLITE_OK. ** Otherwise, an [error code] or an [extended error code] is returned.)^ ** ** This routine only works on a [BLOB handle] which has been created ** by a prior successful call to [sqlite3_blob_open()] and which has not ** been closed by [sqlite3_blob_close()]. Passing any other pointer in ** to this routine results in undefined and probably undesirable behavior. ** ** See also: [sqlite3_blob_write()].

65511. Start of original out buffer

65512. Length of token z

65513. ifndef SQLITE_DISABLE_FTS3_UNICODE

65514. ** This routine is the same as the sqlite3_complete() routine described ** above, except that the parameter is required to be UTF-16 encoded, not ** UTF-8.

65515. Pointer to position list following iDocid

65516. Docid of row being written

65517. Name of database containing r-tree table

65518. Step 2: Check to see if the LHS contains any NULL columns. If the ** LHS does contain NULLs then the result must be either FALSE or NULL. ** We will then skip the binary search of the RHS.

65519. Write index of pSrc->a[*piTab].pTab->aCol[] here

65520. ** Return TRUE if pFile has been renamed or unlinked since it was first opened.

65521. ** Vowel or consonant

65522. Key for WITHOUT ROWID tables

65523. ** Delete any previous value and set the value stored in *pMem to val, ** manifest type REAL.

65524. idlist_opt ::=

65525. If the pager is configured to use locking_mode=exclusive, and an ** exclusive lock on the database is not already held, obtain it now.

65526. Length of zTerm in bytes

65527. Parsing context to record errors

65528. **comment:** Ticket b351d95f9cd5ef17e9d9dbae18f5ca8611190001: ** The value in regFree1 might get SCopy-ed into the file result. ** So make sure that the regFree1 register is not reused for other ** purposes and possibly overwritten.

label: code-design

65529. The SELECT statement to be fixed to one database

65530. If required, populate the output variables with a pointer to and the ** size of the previous offset-list.

65531. ** Tokenize using the porter tokenizer.

65532. ** This lookup table is used to help decode the first byte of ** a multi-byte UTF8 character. It is copied here from SQLite source ** code file utf8.c.

65533. Disallow the load_extension() SQL function unless the SQLITE_LoadExtFunc ** flag is set. See the sqlite3_enable_load_extension() API.

65534. Minimum number of segments to merge

65535. ** 2010 February 1 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This header file defines the interface to the write-ahead logging ** system. Refer to the comments below and the header comment attached to ** the implementation of each function in log.c for further details.

65536. A syntax error has occurred. ** The response to an error depends upon whether or not the ** grammar defines an error token "ERROR". ** ** This is what we do if the grammar does define ERROR: ** ** Call the %syntax_error function. ** ** Begin popping the stack until we enter a state where ** it is legal to shift the error symbol, then shift ** the error symbol. ** ** Set the error count to three. ** ** Begin accepting and shifting new tokens. No new error ** processing will occur until three tokens have been ** shifted successfully. **

65537. Vector of values to consider

65538. Create a divider cell to insert into pParent. The divider cell ** consists of a 4-byte page number (the page number of pPage) and ** a variable length key value (which must be the same value as the ** largest key on pPage). ** ** To find the largest key value on pPage, first find the right-most ** cell on pPage. The first two fields of this cell are the ** record-length (a variable length integer at most 32-bits in size) ** and the key value (a variable length integer, may have any value). ** The first of the while(...) loops below skips over the record-length ** field. The second while(...) loop copies the key value from the ** cell on pPage into the pSpace buffer.

65539. Value returned by xInstCount()

65540. Which metric to return

65541. ***** Begin file parse.c *****

65542. ** NOTE: The WinRT sub-platform is always assumed to be based on the NT ** kernel.

65543. LEFT

65544. Size of allocated buffer

65545. ** Bind a text or BLOB value.

65546. Do not save the results anywhere
65547. Restriction (2b)
65548. ** Generate an expression tree to implement the WHERE, ORDER BY, ** and LIMIT/OFFSET portion of DELETE and UPDATE statements. ** ** DELETE FROM table_wxyz WHERE a<5 ORDER BY a LIMIT 1; ** _____/ ** pLimitWhere (pInClause)
65549. end-of-error-codes
65550. ** 2007 August 27 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used to implement mutexes on Btree objects. ** This code really belongs in btree.c. But btree.c is getting too ** big and we want to break it down some. This packaged seemed like ** a good breakout.
65551. If the first byte was a '[', then the close-quote character is a ']'
65552. ** This function may only be called if the b-tree connection already ** has a read or write transaction open on the database. ** ** Read the meta-information out of a database file. Meta[0] ** is the number of free pages currently in the database. Meta[1] ** through meta[15] are available for use by higher layers. Meta[0] ** is read-only, the others are read/write. ** ** The schema layer numbers meta values differently. At the schema ** layer (and the SetCookie and ReadCookie opcodes) the number of ** free pages is not visible. So Cookie[0] is the same as Meta[1]. ** ** This routine treats Meta[BTREE_DATA_VERSION] as a special case. Instead ** of reading the value out of the header, it instead loads the "DataVersion" ** from the pager. The BTREE_DATA_VERSION value is not actually stored in the ** database file. It is a number computed by the pager. But its access ** pattern is the same as header meta values, and so it is convenient to ** read it from this routine.
65553. This particular expression does not need to be expanded.
65554. This case should work for GCC
65555. trigger_cmd_list
65556. Number of elements left to test
65557. ** CAPI3REF: Source Of Data In A Query Result ** METHOD: sqlite3_stmt ** ** ^These routines provide a means to determine the database, table, and ** table column that is the origin of a particular result column in ** [SELECT] statement. ** ^The name of the database or table or column can be returned as ** either a UTF-8 or UTF-16 string. ^The _database_ routines return ** the database name, the _table_ routines return the table name, and ** the origin_ routines return the column name. ** ^The returned string is valid until the [prepared statement] is destroyed ** using [sqlite3_finalize()] or until the statement is automatically ** reprepared by the first call to [sqlite3_step()] for a particular run ** or until the same information is requested ** again in a different encoding. **
** ^The names returned are the original un-aliased names of the ** database, table, and column. ** ** ^The first argument to these interfaces is a [prepared statement]. ** ^These functions return information about the Nth result column returned by ** the statement, where N is the second function argument. ** ^The left-most column is column 0 for these routines. ** ** ^If the Nth column returned by the statement is an expression or ** subquery and is not a column value, then all of these functions return ** NULL. ^These routine might also return NULL if a memory allocation error ** occurs. ^Otherwise, they return the name of the attached database, table, ** or column that query result column was extracted from. ** ** ^As with all other SQLite APIs, those whose names end with "16" return ** UTF-16 encoded strings and the other functions return UTF-8. ** ** ^These APIs are only available if the library was compiled with the ** [SQLITE_ENABLE_COLUMN_METADATA] C-preprocessor symbol. ** ** If two or more threads call one or more of these routines against the same ** prepared statement and column at the same time then the results are ** undefined. ** ** If two or more threads call one or more ** [sqlite3_column_database_name | column metadata interfaces] ** for the same [prepared statement] and result column ** at the same time then the results are undefined.
65558. Read-only HEADER_VALUE
65559. init_deferred_pred_opt ::=
65560. Automatically expire on reset
65561. If the operating system support deletion of open files, then ** close the journal file when dropping the database lock. Otherwise ** another connection with journal_mode=delete might delete the file ** out from under us.
65562. Code the SELECT statements to our left
65563. Flags for Fts5IndexQuery
65564. ** Each database file to be accessed by the system is an instance ** of the following structure. There are normally two of these structures ** in the sqlite.aDb[] array. aDb[0] is the main database file and ** aDb[1] is the database file used to hold temporary tables. Additional ** databases may be attached.
65565. ** Return the size of the header on each page of this PCACHE implementation.
65566. AFTER
65567. EVIDENCE-OF: R-03996-12088 The M parameter must be a valid checkpoint ** mode:
65568. Smallest power of two >= nReader
65569. Extra bytes append to each in-memory page
65570. Insert the new record into the r-tree
65571. IN: Size of poslist in bytes
65572. ** When building up a FROM clause in the parser, the join operator ** is initially attached to the left operand. But the code generator ** expects the join operator to be on the right operand. This routine ** Shifts all join operators from left to right for an entire FROM ** clause. ** ** Example: Suppose the join is like this: **
** A natural cross join B. ** ** The operator is "natural cross join". The A and B operands are stored ** in p->a[0] and p->a[1], respectively. The parser initially stores the ** operator with A. This routine shifts that operator over to B.
65573. If the bPreserve flag is set to true, then the cursor position must ** be preserved following this delete operation. If the current delete ** will cause a b-tree rebalance, then this is done by saving the cursor ** key and leaving the cursor in CURSOR_REQUIRESEEK state before ** returning. ** ** Or, if the current delete will not cause a rebalance, then the cursor ** will be left in CURSOR_SKIPNEXT state pointing to the entry immediately ** before or after the deleted entry. In this case set bSkipnext to true.
65574. xColumn - read data
65575. P4 is a pointer to a Mem* structure
65576. ** sqlite3_test_control(SQLITE_TESTCTRL_ASSERT, int X) ** ** This action provides a run-time test to see whether or not ** assert() was enabled at compile-time. If X is true and assert() ** is enabled, then the return value is true. If X is true and ** assert() is disabled, then the return value is zero. If X is ** false and assert() is enabled, then the assertion fires and the ** process aborts. If X is false and assert() is disabled, then the ** return value is zero.
65577. Length of th name
65578. Original name of the table
65579. growing the file does not occur until ** the write succeeds
65580. comment: Some terms not completely tested
label: test
65581. OUT: Total number of pages available for recycling
65582. Number of valid entries in aTo[] and aFrom[]
65583. ** Compute both YMD and HMS
65584. Database idx for pTab
65585. OUT: Returned database handle
65586. ** Add a lock on the table with root-page iTable to the shared-btree used ** by Btree handle p. Parameter eLock must be either READ_LOCK or ** WRITE_LOCK. ** ** This function assumes the following: ** ** (a) The specified Btree object p is connected to a sharable ** database (one with the BtShared.sharable flag set), and ** ** (b) No other Btree objects hold a lock that conflicts ** with the requested lock (i.e. querySharedCacheTableLock() has ** already been called and returned SQLITE_OK). ** ** SQLITE_OK is returned if the lock is added successfully. SQLITE_NOMEM ** is returned if a malloc attempt fails.
65587. Set a flag that indicates we're the first to create the memory so it ** must be zero-initialized
65588. ** Initialize a SelectDest structure.
65589. Transfer the next row in Queue over to Current
65590. It is not possible for eType to be SQLITE_NULL here. The session ** module does not record changes for rows with NULL values stored in ** primary key columns.
65591. ** Clear the i-th bit. ** ** pBuf must be a pointer to at least BITVEC_SZ bytes of temporary storage ** that BitvecClear can use to rebuilt its hash table.
65592. Number of bits in the bitmap array.
65593. Start transaction here if not NULL

65594. The code below is handling the return value of osAllocate() ** correctly. posix_fallocate() is defined to "returns zero on success, ** or an error number on failure". See the manpage for details.

65595. Array with one element per query phrase

65596. Save the current value of the database flags so that it can be ** restored before returning. Then set the writable-schema flag, and ** disable CHECK and foreign key constraints.

65597. FTS5 mask for idxFlags

65598. OUT: Frame number (or zero)

65599. SQLITE_OMIT_QUICKBALANCE

65600. Page is on the PCache.pDirty list

65601. The pPg->nFree field is now set incorrectly. The caller will fix it.

65602. ** The following structure holds pointers to all of the SQLite API ** routines. *** WARNING: In order to maintain backwards compatibility, add new ** interfaces to the end of this structure only. If you insert new ** interfaces in the middle of this structure, then older different ** versions of SQLite will not be able to load each other's shared ** libraries!

65603. #define TRANSLATE_TRACE 1

65604. Cursor variable

65605. **comment:** ** Return a pointer corresponding to database zDb (i.e. "main", "temp") ** in connection handle pDb. If such a database cannot be found, return ** a NULL pointer and write an error message to pErrorDb. *** If the "temp" database is requested, it may need to be opened by this ** function. If an error occurs while doing so, return 0 and write an ** error message to pErrorDb.
label: code-design

65606. **comment:** How far to indent SELECTTRACE() output
label: code-design

65607. Db handle used to check for interrupts

65608. ** Sync the database file for the pager pPager. zMaster points to the name ** of a master journal file that should be written into the individual ** journal file. zMaster may be NULL, which is interpreted as no master ** journal (a single database transaction). *** This routine ensures that: *** * The database file change-counter is updated, *** * the journal is synced (unless the atomic-write optimization is used), *** * all dirty pages are written to the database file, *** * the database file is truncated (if required), and *** * the database file synced. *** The only thing that remains to commit the transaction is to finalize ** (delete, truncate or zero the first part) of the journal file (or ** delete the master journal file if specified). *** Note that if zMaster==NULL, this does not overwrite a previous value ** passed to an sqlite3PagerCommitPhaseOne() call. *** If the final parameter - noSync - is true, then the database file itself ** is not synced. The caller must call sqlite3PagerSync() directly to ** sync the database file before calling CommitPhaseTwo() to delete the ** journal file in this case.

65609. Opcode: Ge P1 P2 P3 P4 P5 ** Synopsis: IF r[P3]>=r[P1] *** This works just like the Lt opcode except that the jump is taken if ** the content of register P3 is greater than or equal to the content of ** register P1. See the Lt opcode for additional information.

65610. expr ::= expr ISNULL|NOTNULL

65611. ** Sizes and pointers of various parts of the Parse object.

65612. IMP: R-22934-25134

65613. ***** End of fts3.h *****

65614. ** Move the page pPg to location pgno in the file. *** There must be no references to the page previously located at ** pgno (which we call pPgOld) though that page is allowed to be ** in cache. If the page previously located at pgno is not already ** in the rollback journal, it is not put there by this routine. *** ** References to the page pPg remain valid. Updating any ** meta-data associated with pPg (i.e. data stored in the nExtra bytes ** allocated along with the page) is the responsibility of the caller. *** A transaction must be active when this routine is called. It used to be ** required that a statement transaction was not active, but this restriction ** has been removed (CREATE INDEX needs to move a page when a statement ** transaction is active). *** If the fourth argument, isCommit, is non-zero, then this page is being ** moved as part of a database reorganization just before the transaction ** is being committed. In this case, it is guaranteed that the database page ** pPg refers to will not be written to again within this transaction. *** This function may return SQLITE_NOMEM or an IO error code if an error ** occurs. Otherwise, it returns SQLITE_OK.

65615. If the file was successfully opened for read/write access, ** choose a default page size in case we have to create the ** database file. The default page size is the maximum of: ** + SQLITE_DEFAULT_PAGE_SIZE, ** + The value returned by sqlite3OsSectorSize() ** + The largest page size that can be written atomically.

65616. iLevel value for root of the tree

65617. **comment:** ** CAPI3REF: Unlock Notification ** METHOD: sqlite3 *** ^When running in shared-cache mode, a database operation may fail with ** an [SQLITE_LOCKED] error if the required locks on the shared-cache or ** individual tables within the shared-cache cannot be obtained. See ** [SQLite Shared-Cache Mode] for a description of shared-cache locking. ** ^This API may be used to register a callback that SQLite will invoke ** when the connection currently holding the required lock relinquishes it. ** ^This API is only available if the library was compiled with the ** [SQLITE_ENABLE_UNLOCK_NOTIFY] C-preprocessor symbol defined. *** See Also: [Using the SQLite Unlock Notification Feature]. *** ^Shared-cache locks are released when a database connection concludes ** its current transaction, either by committing it or rolling it back. *** ^When a connection (known as the blocked connection) fails to obtain a ** shared-cache lock and SQLITE_LOCKED is returned to the caller, the ** identity of the database connection (the blocking connection) that ** has locked the required resource is stored internally. ^After an ** application receives an SQLITE_LOCKED error, it may call the ** sqlite3_unlock_notify() method with the blocked connection handle as ** the first argument to register for a callback that will be invoked ** when the blocking connections current transaction is concluded. ^The ** callback is invoked from within the [sqlite3_step] or [sqlite3_close] ** call that concludes the blocking connections transaction. *** ^If sqlite3_unlock_notify() is called in a multi-threaded application, ** there is a chance that the blocking connection will have already ** concluded its transaction by the time sqlite3_unlock_notify() is invoked. ** If this happens, then the specified callback is invoked immediately, ** from within the call to sqlite3_unlock_notify().)^** ** ^If the blocked connection is attempting to obtain a write-lock on a ** shared-cache table, and more than one other connection currently holds ** a read-lock on the same table, then SQLite arbitrarily selects one of ** the other connections to use as the blocking connection. *** ** ^There may be at most one unlock-notify callback registered by a ** blocked connection. If sqlite3_unlock_notify() is called when the ** blocked connection already has a registered unlock-notify callback, ** then the new callback replaces the old.^** ^If sqlite3_unlock_notify() is ** called with a NULL pointer as its second argument, then any existing ** unlock-notify callback is canceled. ^The blocked connections ** unlock-notify callback may also be canceled by closing the blocked ** connection using [sqlite3_close]. *** ^The unlock-notify callback is not reentrant. If an application invokes ** any sqlite3_xxx API functions from within an unlock-notify callback, a ** crash or deadlock may be the result. *** ** ^Unless deadlock is detected (see below), sqlite3_unlock_notify() always ** returns SQLITE_OK. *** Callback Invocation Details *** When an unlock-notify callback is registered, the application provides a ** single void* pointer that is passed to the callback when it is invoked. ** However, the signature of the callback function allows SQLite to pass ** it an array of void* context pointers. The first argument passed to ** an unlock-notify callback is a pointer to an array of void* pointers, ** and the second is the number of entries in the array. *** When a blocking connections transaction is concluded, there may be ** more than one blocked connection that has registered for an unlock-notify ** callback. ^If two or more such blocked connections have specified the ** same callback function, then instead of invoking the callback function ** multiple times, it is invoked once with the set of void* context pointers ** specified by the blocked connections bundled together into an array. ** This gives the application an opportunity to prioritize any actions ** related to the set of unblocked database connections. *** Deadlock Detection *** Assuming that after registering for an unlock-notify callback a ** database waits for the callback to be issued before taking any further ** action (a reasonable assumption), then using this API may cause the ** application to deadlock. For example, if connection X is waiting for ** connection Y's transaction to be concluded, and similarly connection ** Y is waiting on connection X's transaction, then neither connection ** will proceed and the system may remain deadlocked indefinitely. *** To avoid this scenario, the sqlite3_unlock_notify() performs deadlock ** detection. ^If a given call to sqlite3_unlock_notify() would put the ** system in a deadlocked state, then SQLITE_LOCKED is returned and no ** unlock-notify callback is registered. The system is said to be in ** a deadlocked state if connection A has registered for an unlock-notify ** callback on the conclusion of connection B's transaction, and connection ** B has itself registered for an unlock-notify callback when connection ** A's transaction is concluded. ^Indirect deadlock is also detected, so ** the system is also considered to be deadlocked if connection B has ** registered for an unlock-notify callback on the conclusion of connection ** C's transaction, where connection C is waiting on connection A. ^Any ** number of levels of indirection are allowed. *** The "DROP TABLE" Exception *** When a call to [sqlite3_step()] returns SQLITE_LOCKED, it is almost ** always appropriate to call sqlite3_unlock_notify(). There is however, ** one exception. When executing a "DROP TABLE" or "DROP INDEX" statement, ** SQLite checks if there are any currently executing SELECT statements ** that belong to the same connection. If there are, SQLITE_LOCKED is ** returned. In this case there is no "blocking connection", so invoking ** sqlite3_unlock_notify() results in the unlock-notify callback being ** invoked immediately. If the application then re-attempts the "DROP TABLE" ** or "DROP INDEX" query, an infinite loop might be the result. *** One way around this problem is to check the extended error code returned ** by an sqlite3_step() call. ^If there is a blocking connection, then the ** extended

error code is set to SQLITE_LOCKED_SHAREDCACHE. Otherwise, in ** the special "DROP TABLE/INDEX" case, the extended error code is just **
SQLITE_LOCKED.)^

label: code-design

65618. Language id to use

65619. same as TK_NOT, in1, out2

65620. OUT: New value (or NULL pointer)

65621. Advance to the next name context. The loop will exit when either ** we have a match (cnt>0) or when we run out of name contexts.

65622. ** Reclaim all memory of a Stat4Accum structure.

65623. Index in aSample[] of test sample

65624. ***** Begin file callback.c *****

65625. ** Return the number of bytes of payload for the entry that pCur is ** currently pointing to. For table btrees, this will be the amount ** of data. For index btrees, this will be the size of the key. *** The caller must guarantee that the cursor is pointing to a non-NULL ** valid entry. In other words, the calling procedure must guarantee ** that the cursor has Cursor.eState==CURSOR_VALID.

65626. EVIDENCE-OF: R-41613-20553 The sqlite3_wal_checkpoint(D,X) is equivalent to ** sqlite3_wal_checkpoint_v2(D,X,SQLITE_CHECKPOINT_PASSIVE,0,0).

65627. ** Implement a memory barrier or memory fence on shared memory. *** All loads and stores begun before the barrier must complete before ** any load or store begun after the barrier.

65628. ** Return true if the argument corresponds to a unicode codepoint ** classified as either a letter or a number. Otherwise false. *** The results are undefined if the value passed to this function ** is less than zero.

65629. ** A sub-routine used to implement a trigger program.

65630. **comment:** SQLITE_DEBUG || SQLITE_COVERAGE_TEST

label: test

65631. ** Attempt an incremental merge that writes nMerge leaf blocks. *** Incremental merges happen nMin segments at a time. The segments ** to be merged are the nMin oldest segments (the ones with the smallest ** values for the _segdir.idx field) in the highest level that contains ** at least nMin segments. Multiple merges might occur in an attempt to ** write the quota of nMerge leaf blocks.

65632. ** The journal header size for this pager. This is usually the same ** size as a single disk sector. See also setSectorSize().

65633. Make sure the schema version is at least 3. But do not upgrade ** from less than 3 to 4, as that will corrupt any preexisting DESC ** index.

65634. If foreign-keys are disabled, this function is a no-op.

65635. The requested data is available in the in-memory buffer. In this ** case there is no need to make a copy of the data, just return a ** pointer into the buffer to the caller.

65636. Write the number of available bytes here

65637. Sort order of primary key when pList==NULL

65638. ** Move the cursor to point to the root page of its b-tree structure. *** If the table has a virtual root page, then the cursor is moved to point ** to the virtual root page instead of the actual root page. A table has a ** virtual root page when the actual root page contains no cells and a ** single child page. This can only happen with the table rooted at page 1. *** If the b-tree structure is empty, the cursor state is set to ** CURSOR_INVALID. Otherwise, the cursor is set to point to the first ** cell located on the root (or virtual root) page and the cursor state ** is set to CURSOR_VALID. *** If this function returns successfully, it may be assumed that the ** page-header flags indicate that the [virtual] root-page is the expected ** kind of b-tree page (i.e. if when opening the cursor the caller did not ** specify a KeyInfo structure the flags byte is set to 0x05 or 0x0D, ** indicating a table b-tree, or if the caller did specify a KeyInfo ** structure the flags byte is set to 0x02 or 0x0A, indicating an index ** b-tree).

65639. If this function is inside of a trigger, the register array in aMem[] ** might change from one evaluation to the next. The next block of code ** checks to see if the register array has changed, and if so it ** reinitializes the relevant parts of the sqlite3_context object

65640. Increment the row counter

65641. ** fts3ExprIterate() callback used to collect the "local" part of the ** FTS3_MATCHINFO_HITS array. The local stats are those elements of the ** array that are different for each row returned by the query.

65642. Valid if eType==FTSQUERY_PHRASE

65643. Write values to this array

65644. Context for the sanity check

65645. ** Flush any data stored in the in-memory hash tables to the database. ** Also close any open blob handles.

65646. **comment:** ** Give a callback to the test harness that can be used to simulate faults ** in places where it is difficult or expensive to do so purely by means ** of inputs. *** The intent of the integer argument is to let the fault simulator know ** which of multiple sqlite3FaultSim() calls has been hit. *** Return whatever integer value the test callback returns, or return ** SQLITE_OK if no test callback is installed.

label: code-design

65647. Open page to move

65648. Next unread header byte

65649. Integer value used when MEM_Int is set in flags

65650. Establish a read-lock on the table at the shared-cache level. ** Open a read-only cursor on the table. Also allocate a cursor number ** to use for scanning indexes (iIdxCur). No index cursor is opened at ** this time though.

65651. Release the locks

65652. ** This function determines whether or not the atomic-write optimization ** can be used with this pager. The optimization can be used if: *** (a) the value returned by OsDeviceCharacteristics() indicates that ** a database page may be written atomically, and ** (b) the value returned by OsSectorSize() is less than or equal ** to the page size. *** The optimization is also always enabled for temporary files. It is ** an error to call this function if pPager is opened on an in-memory ** database. *** If the optimization cannot be used, 0 is returned. If it can be used, ** then the value returned is the size of the journal file when it ** contains rollback data for exactly one page.

65653. stl_prefix ::=

65654. ** Invoke this macro on memory cells just prior to changing the ** value of the cell. This macro verifies that shallow copies are ** not misused. A shallow copy of a string or blob just copies a ** pointer to the string or blob, not the content. If the original ** is changed while the copy is still in use, the string or blob might ** be changed out from under the copy. This macro verifies that nothing ** like that ever happens.

65655. Position-list terminator

65656. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** An tokenizer for SQL *** This file contains C code that implements the sqlite3_complete() API. ** This code used to be part of the tokenizer.c source file. But by ** separating it out, the code will be automatically omitted from ** static links that do not use it.

65657. Max cache entries for pcacheDump()

65658. ** Compare the values contained by the two memory cells, returning ** negative, zero or positive if pMem1 is less than, equal to, or greater ** than pMem2. Sorting order is NULL's first, followed by numbers (integers ** and reals) sorted numerically, followed by text ordered by the collating ** sequence pColl and finally blob's ordered by memcmp(). *** Two NULL values are considered equal by this function.

65659. defined(SQLITE_MUTEX_NOOP)

65660. Number of segments in incr-merge

65661. Name of new SQL function

65662. ** If a row with rowid iDel is present in the %_content table, add the ** delete-markers to the FTS index necessary to delete it. Do not actually ** remove the %_content row at this time though.

65663. type of the next token

65664. NULL set by OP_Null, not from data

65665. Pull data from this table

65666. A running thread

65667. Expression for VALUE in the x=VALUE constraint

65668. endif pExpr->iTable==pItem->iCursor

65669. ** CAPI3REF: Virtual Table Scan Flags

65670. Sizes in bytes of nPrefix prefix indexes
65671. Number of leaf pages per optimize step
65672. 16
65673. An SQL table corresponding to zName
65674. If the query was read-only and the error code is SQLITE_INTERRUPT, ** no rollback is necessary. Otherwise, at least a savepoint ** transaction must be rolled back to restore the database to a ** consistent state. *** Even if the statement is read-only, it is important to perform ** a statement or transaction rollback operation. If the error ** occurred while writing to the journal, sub-journal or database ** file as part of an effort to free up cache space (see function ** pagerStress() in pager.c), the rollback is required to restore ** the pager to a consistent state.
65675. ***** Begin file mem1.c *****
65676. Cursor number for current WhereLoop
65677. expr ::= EXISTS LP select RP
65678. No other row with the new.* primary key.
65679. This call may return SQLITE_ABORT if there has been a savepoint ** rollback since it was last used. In this case a new blob handle ** is required.
65680. ** Load all automatic extensions. *** If anything goes wrong, set an error in the database connection.
65681. ** Wrapper around OS specific sqlite3_os_init() function.
65682. Show short columns names
65683. True if this frame is valid
65684. btreeParseCell method
65685. FOR => ID
65686. Current UNION ALL term of the other query
65687. ** Free all resources currently held by the cursor passed as the only ** argument.
65688. Argument passed into xTask()
65689. ** xOpen - Open a cursor.
65690. ***** Begin file pcache.c *****
65691. ** Another built-in collating sequence: NOCASE. *** This collating sequence is intended to be used for "case independent ** comparison". SQLite's knowledge of upper and lower case equivalents ** extends only to the 26 characters used in the English language. *** At the moment there is only a UTF-8 implementation.
65692. ** A linked list of the following structures is stored at BtShared.pLock. ** Locks are added (or upgraded from READ_LOCK to WRITE_LOCK) when a cursor ** is opened on the table with root page BtShared.iTable. Locks are removed ** from this list when a transaction is committed or rolled back, or when ** a btree handle is closed.
65693. SQLITE_LOCKED
65694. New value for pMem->flags
65695. ** This function returns a pointer to the array of opcodes associated with ** the Vdbe passed as the first argument. It is the callers responsibility ** to arrange for the returned array to be eventually freed using the ** vdbeFreeOpArray() function. *** Before returning, *pnOp is set to the number of entries in the returned ** array. Also, *pnMaxArg is set to the larger of its current value and ** the number of entries in the Vdbe.apArg[] array required to execute the ** returned program.
65696. ** Most queries use only a single table (they are not joins) and have ** simple == constraints against indexed fields. This routine attempts ** to plan those simple cases using much less ceremony than the ** general-purpose query planner, and thereby yield faster sqlite3_prepare() ** times for the common case. *** Return non-zero on success, if this query can be handled by this ** no-frills query planner. Return zero if this query needs the ** general-purpose query planner.
65697. IN/OUT: Lastest rowid seen so far
65698. If 8+3 names are possible, then the journal file might not contain ** a '-' character. So check for that case and return early.
65699. Opcode: SorterSort P1 P2 * * * * * After all records have been inserted into the Sorter object ** identified by P1, invoke this opcode to actually do the sorting.
*** Jump to P2 if there are no records to be sorted. *** This opcode is an alias for OP_Sort and OP_Rewind that is used ** for Sorter objects.
65700. Value to return via *ppIdx
65701. Change a page number. Used by incr-vacuum.
65702. **comment:** PAGER_GET_XXX flags
 label: code-design
65703. 64-bit position read from poslist
65704. boundaries on sqlite3GlobalConfig.mnReq are enforced in sqlite3_config()
65705. TRUNCATE
65706. None of the existing best-so-far paths match the candidate.
65707. ** A specially optimized version of vdbeSorterCompare() that assumes that ** the first field of each key is an INTEGER value.
65708. ***** Begin file fts3_porter.c *****
65709. Loop through all indices on the table, checking each to see if it makes ** the DISTINCT qualifier redundant. It does so if: *** 1. The index is itself UNIQUE, and *** 2. All of the columns in the index are either part of the pDistinct ** list, or else the WHERE clause contains a term of the form "col=X", ** where X is a constant value. The collation sequences of the ** comparison and select-list expressions must match those of the index. *** 3. All of those index columns for which the WHERE clause does not ** contain a "col=X" term are subject to a NOT NULL constraint.
65710. !SQLITE OMIT_OR_OPTIMIZATION && !SQLITE OMIT_SUBQUERY
65711. Callback info
65712. If the createFlag parameter is true and the search did not reveal an ** exact match for the name, number of arguments and encoding, then add a ** new entry to the hash table and return it.
65713. Argument to xTask
65714. Copy of BtShared.minLocal or BtShared.minLeaf
65715. An error has occurred. Return an error code.
65716. 1=rel to end of fork, 0=rel to start
65717. ** Trace output macros
65718. ** Return the rowid of the entry that the iterator currently points ** to. If the iterator points to EOF when this function is called the ** results are undefined.
65719. Temp db schema
65720. ** This procedure generates VDBE code for a single invocation of either the ** sqlite_detach() or sqlite_attach() SQL user functions.
65721. ** Global data used by this cache.
65722. init_deferred_pred_opt ::= INITIALLY DEFERRED
65723. trigger_event ::= DELETE|INSERT
65724. **comment:** The WhereLoop object being coded
 label: code-design
65725. E
65726. P3
65727. Used to allocate unique cursor ids
65728. ** The list of all registered VFS implementations.
65729. ** PRAGMA [schema.]default_cache_size ** PRAGMA [schema.]default_cache_size=N *** The first form reports the current persistent setting for the ** page cache size. The value returned is the maximum number of ** pages in the page cache. The second form sets both the current ** page cache size value and the persistent page cache size value ** stored in the database file. *** Older versions of SQLite would set the default cache size to a ** negative number to indicate synchronous=OFF. These days, synchronous ** is always on by default regardless of the sign of the default cache ** size. But continue to take the absolute value of the default cache ** size of historical compatibility.
65730. Opcode: RowSetRead P1 P2 P3 * * * * * Synopsis: r[P3]=rowset(P1) *** Extract the smallest value from the RowSet object in P1 ** and put that value into register P3. ** Or, if RowSet object P1 is initially empty, leave P3 ** unchanged and jump to instruction P2.
65731. ** This function is called by the parser when it parses a command to create, ** release or rollback an SQL savepoint.
65732. This is an EOF condition
65733. Current number of users of this structure
65734. Length of a token prefix
65735. Virtual table cursor handle

65736. Maximum length of output string in bytes
65737. ***** End of sqlite3rtree.h *****

65738. ** CAPI3REF: Initialize The SQLite Library *** ^The sqlite3_initialize() routine initializes the ** SQLite library. ^The sqlite3_shutdown() routine ** deallocates any resources that were allocated by sqlite3_initialize(). ** These routines are designed to aid in process initialization and ** shutdown on embedded systems. Workstation applications using ** SQLite normally do not need to invoke either of these routines. *** A call to sqlite3_initialize() is an "effective" call if it is ** the first time sqlite3_initialize() is invoked during the lifetime of ** the process, or if it is the first time sqlite3_initialize() is invoked ** following a call to sqlite3_shutdown(). ^Only an effective call ** of sqlite3_initialize() does any initialization. All other calls ** are harmless no-ops.)^ *** A call to sqlite3_shutdown() is an "effective" call if it is the first ** call to sqlite3_shutdown() since the last sqlite3_initialize(). ^Only ** an effective call to sqlite3_shutdown() does any deinitialization. ** All other valid calls to sqlite3_shutdown() are harmless no-ops.)^ *** The sqlite3_initialize() interface is threadsafe, but sqlite3_shutdown() ** is not. The sqlite3_shutdown() interface must only be called from a ** single thread. All open [database connections] must be closed and all ** other SQLite resources must be deallocated prior to invoking ** sqlite3_shutdown(). *** Among other things, ^sqlite3_initialize() will invoke ** sqlite3_os_init(). Similarly, ^sqlite3_shutdown() ** will invoke sqlite3_os_end(). *** ^The sqlite3_initialize() routine returns [SQLITE_OK] on success. *** ^If for some reason, sqlite3_initialize() is unable to initialize ** the library (perhaps it is unable to allocate a needed resource such ** as a mutex) it returns an [error code] other than [SQLITE_OK]. *** ^The sqlite3_initialize() routine is called internally by many other ** SQLite interfaces so that an application usually does not need to ** invoke sqlite3_initialize() directly. For example, [sqlite3_open()] ** calls sqlite3_initialize() so the SQLite library will be automatically ** initialized when [sqlite3_open()] is called if it has not been initialized ** already. ^However, if SQLite is compiled with the [SQLITE_OMIT_AUTOINIT] ** compile-time option, then the automatic calls to sqlite3_initialize() ** are omitted and the application must call sqlite3_initialize() directly ** prior to using any other SQLite interface. For maximum portability, ** it is recommended that applications always invoke sqlite3_initialize() ** directly prior to using any other SQLite interface. Future releases ** of SQLite may require this. In other words, the behavior exhibited ** when SQLite is compiled with [SQLITE_OMIT_AUTOINIT] might become the ** default behavior in some future release of SQLite. *** ^The sqlite3_os_init() routine does operating-system specific ** initialization of the SQLite library. The sqlite3_os_end() ** routine undoes the effect of sqlite3_os_init(). Typical tasks ** performed by these routines include allocation or deallocation ** of static resources, initialization of global variables, ** setting up a default [sqlite3_vfs] module, or setting up ** a default configuration using [sqlite3_config()]. *** ^The application should never invoke either sqlite3_os_init() ** or sqlite3_os_end() directly. The application should only invoke ** sqlite3_initialize() and sqlite3_shutdown(). The sqlite3_os_init() ** interface is called automatically by sqlite3_initialize() and ** sqlite3_os_end() is called by sqlite3_shutdown(). Appropriate ** implementations for sqlite3_os_init() and sqlite3_os_end() ** are built into SQLite when it is compiled for Unix, Windows, or OS/2. ** When [custom builds | built for other platforms] ** (using the [SQLITE_OS_OTHER=1] compile-time ** option) the application must supply a suitable implementation for ** sqlite3_os_init() and sqlite3_os_end(). An application-supplied ** implementation of sqlite3_os_init() or sqlite3_os_end() ** must return [SQLITE_OK] on success and some other [error code] upon ** failure.

65739. Number of nested calls to sqlite3BtreeEnter()
65740. Default value
65741. Else this is a rollback operation, playback the specified savepoint. ** If this is a temp-file, it is possible that the journal file has ** not yet been opened. In this case there have been no changes to ** the database file, so the playback operation can be skipped.
65742. If we are creating a set for an "expr IN (SELECT ...)".
65743. Cost of running iterator
65744. Where to write the extracted value
65745. Second argument to pass to callback
65746. The walker node callback used to transform matching expressions into ** a reference to an index column for an index on an expression. *** If pExpr matches, then transform it into a reference to the index column ** that contains the value of pExpr.
65747. the complete WHERE clause
65748. UNIQUERYWITHOUTRELEASEATTACHAVINGGROUPDATEBEGINNERECURSIVE
65749. ** Obtain the BtShared mutex associated with B-Tree handle p. Also, ** set BtShared.db to the database handle associated with p and the ** p->locked boolean to true.
65750. 0x12
65751. Sort the list
65752. Resolve names in table-valued-function arguments
65753. 173
65754. ** When the core wants to read from the virtual table, it creates a ** virtual table cursor (an instance of the following structure) using ** the xOpen method. Cursors are destroyed using the xClose method.
65755. **comment:** Extra initialization needed
 label: code-design
65756. Used by readers to open (lock) and close (unlock) a snapshot. A ** snapshot is like a read-transaction. It is the state of the database ** at an instant in time. sqlite3WalOpenSnapshot gets a read lock and ** preserves the current state even if the other threads or processes ** write to or checkpoint the WAL. sqlite3WalCloseSnapshot() closes the ** transaction and releases the lock.
65757. Affinity for end of range constraint
65758. ** This function is only called for detail=columns tables.
65759. ** Push a new element onto the priority queue
65760. Offset to read at
65761. An empty option name. Ignore this option altogether.
65762. The old rowid
65763. int int
65764. **comment:** #define sqliteHashKey(E) ((E)->pKey) // NOT USED
 label: code-design
65765. Append the new position offset, if necessary
65766. Call fstat() to figure out the permissions on the database file. If ** a new *.shm file is created, an attempt will be made to create it ** with the same permissions.
65767. out of memory
65768. The index to be tested
65769. Table Name Module Name
65770. **comment:** ** Each instance of this object represents an algorithm for evaluating one ** term of a join. Every term of the FROM clause will have at least ** one corresponding WhereLoop object (unless INDEXED BY constraints ** prevent a query solution - which is an error) and many terms of the ** FROM clause will have multiple WhereLoop objects, each describing a ** potential way of implementing that FROM-clause term, together with ** dependencies and cost estimates for using the chosen algorithm. *** Query planning consists of building up a collection of these WhereLoop ** objects, then computing a particular sequence of WhereLoop objects, with ** one WhereLoop object per FROM clause term, that satisfy all dependencies ** and that minimize the overall cost.
 label: code-design
65771. ** Given cursor id iId, return a pointer to the corresponding Fts5Index ** object. Or NULL If the cursor id does not exist. *** If successful, set *ppConfig to point to the associated config object ** before returning.
65772. ** If the reference count has reached zero, rollback any active ** transaction and unlock the pager. *** Except, in locking_mode=EXCLUSIVE when there is nothing to in ** the rollback journal, the unlock is not performed and there is ** nothing to rollback, so this routine is a no-op.
65773. pBuf->p[pBuf->n++] = '0';
65774. **comment:** ** Macros to determine whether the machine is big or little endian, ** and whether or not that determination is run-time or compile-time. *** For best performance, an attempt is made to guess at the byte-order ** using C-preprocessor macros. If that is unsuccessful, or if ** -DSQLITE_RUNTIME_BYTERORDER=1 is set, then byte-order is determined ** at run-time.
 label: code-design
65775. Move the cursor to the last entry in the table. Return SQLITE_OK ** on success. Set *pRes to 0 if the cursor actually points to something ** or set *pRes to 1 if the table is empty.
65776. Correlated subq
65777. Number of pointers to this Table
65778. xShmMap
65779. Left subtree

65780. Number of times each op has been executed
65781. A one-pass strategy that might update more than one row may not ** be used if any column of the index used for the scan is being ** updated. Otherwise, if there is an index on "b", statements like ** the following could create an infinite loop: ** ** UPDATE t1 SET b=b+1 WHERE b>? ** ** Fall back to ONEPASS_OFF if where.c has selected a ONEPASS_MULTI ** strategy that uses an index for which one or more columns are being ** updated.
65782. ** Return the name of the table column from which a result column derives. ** NULL is returned if the result column is an expression or constant or ** anything else which is not an unambiguous reference to a database column.
65783. ** Append the nul-terminated string zStr to the buffer pBuf. This function ** ensures that the byte following the buffer data is set to 0x00, even ** though this byte is not included in the pBuf->n count.
65784. Result of prior seek or 0 if no USESEEKRESULT flag
65785. orderby_opt ::=
65786. ** Attempt to take an exclusive lock on the database file. If a PENDING lock ** is obtained instead, immediately release it.
65787. 89
65788. orderby_opt
65789. ***** Interface to code in fts5_config.c. fts5_config.c contains contains code ** to parse the arguments passed to the CREATE VIRTUAL TABLE statement.
65790. ** Write the header information in pWal->hdr into the wal-index. ** ** The checksum on pWal->hdr is updated before it is written.
65791. A positive nCol means the columns names for this view are ** already known.
65792. SQLITE_TOOBIG
65793. ** If the pBase expression originated in the ON or USING clause of ** a join, then transfer the appropriate markings over to derived.
65794. Index in aData of first cell pointer
65795. ** This following structure defines all the methods for the ** stmt virtual table.
65796. Wait for writers, then checkpoint
65797. Raw cell content
65798. Byte offset at which to write
65799. The prepared stmt under constructions
65800. xEof
65801. ** Check that there are at least nSavepoint savepoints open. If there are ** currently less than nSavepoints open, then open one or more savepoints ** to make up the difference. If the number of savepoints is already ** equal to nSavepoint, then this function is a no-op. ** ** If a memory allocation fails, SQLITE_NOMEM is returned. If an error ** occurs while opening the sub-journal file, then an IO error code is ** returned. Otherwise, SQLITE_OK.
65802. The aggregate function implementation
65803. * An instance of struct TriggerStep is used to store a single SQL statement * that is a part of a trigger-program. * * Instances of struct TriggerStep are stored in a singly linked list (linked * using the "pNext" member) referenced by the "step_list" member of the * associated struct Trigger instance. The first element of the linked list is * the first step of the trigger-program. * * The "op" member indicates whether this is a "DELETE", "INSERT", "UPDATE" or * "SELECT" statement. The meanings of the other members is determined by the * value of "op" as follows: * * (op == TK_INSERT) * orconf -> stores the ON CONFLICT algorithm * pSelect -> If this is an INSERT INTO ... SELECT ... statement, then * this stores a pointer to the SELECT statement. Otherwise NULL. * zTarget -> Dequoted name of the table to insert into. * pExprList -> If this is an INSERT INTO ... VALUES ... statement, then * this stores values to be inserted. Otherwise NULL. * pIdList -> If this is an INSERT INTO ... <column-names> VALUES ... * statement, then this stores the column-names to be * inserted into. * * (op == TK_DELETE) * zTarget -> Dequoted name of the table to delete from. * pWhere -> The WHERE clause of the DELETE statement if one is specified. * Otherwise NULL. * * (op == TK_UPDATE) * zTarget -> Dequoted name of the table to update. * pWhere -> The WHERE clause of the UPDATE statement if one is specified. * Otherwise NULL. * pExprList -> A list of the columns to update and the expressions to update * them to. See sqlite3Update() documentation of "pChanges" * argument.
65804. Result of xTask
65805. Enlarge the pWal->apWiData[] array if required
65806. Pointer to top element of the stack
65807. The table or view to which the trigger applies
65808. The hash for this key.
65809. APIs to grab new and old data with
65810. Shortened table list or OR-clause generation
65811. OUT: Page number
65812. ** Invoke the 'collation needed' callback to request a collation sequence ** in the encoding enc of name zName, length nName.
65813. prepared statement under construction
65814. ATTACH => ID
65815. File handle open on target db
65816. Name of the proxy lock file
65817. **comment:** Conditionally this LIKE operator term
 label: code-design
65818. Partial Index Expression
65819. 191
65820. ** Walk an expression tree. Return non-zero if the expression is constant ** that does no originate from the ON or USING clauses of a join. ** Return 0 if it involves variables or function calls or terms from ** an ON or USING clause.
65821. Pointer to allocated vtab
65822. Number of non-benign errors
65823. ** CAPI3REF: Query Progress Callbacks ** METHOD: sqlite3 *** ^The sqlite3_progress_handler(D,N,X,P) interface causes the callback ** function X to be invoked periodically during long running calls to ** [sqlite3_exec()], [sqlite3_step()] and [sqlite3_get_table()] for ** database connection D. An example use for this ** interface is to keep a GUI updated during a large query. ** ** ^The parameter P is passed through as the only parameter to the ** callback function X. ^The parameter N is the approximate number of ** [virtual machine instructions] that are evaluated between successive ** invocations of the callback X. ^If N is less than one then the progress ** handler is disabled. ** ** ^Only a single progress handler may be defined at one time per ** [database connection]; setting a new progress handler cancels the ** old one. ^Setting parameter X to NULL disables the progress handler. ** ^The progress handler is also disabled by setting N to a value less ** than 1. ** ** ^If the progress callback returns non-zero, the operation is ** interrupted. This feature can be used to implement a ** "Cancel" button on a GUI progress dialog box. ** ** The progress handler callback must not do anything that will modify ** the database connection that invoked the progress handler. ** Note that [sqlite3_prepare_v2()] and [sqlite3_step()] both modify their ** database connections for the meaning of "modify" in this paragraph. **
65824. Main VM
65825. ** Decode the flags byte (the first byte of the header) for a page ** and initialize fields of the MemPage structure accordingly. ** ** Only the following combinations are supported. Anything different ** indicates a corrupt database files: ** ** PTF_ZERODATA ** PTF_ZERODATA | PTF_LEAF ** PTF_LEAFDATA | PTF_INTKEY ** PTF_LEAFDATA | PTF_INTKEY | PTF_LEAF
65826. ** Close a connection to a log file.
65827. pPager->nPage = 0;
65828. Limit register for select-A
65829. ** GCC with -pedantic-errors says that C90 does not allow a void* to be ** cast into a pointer to a function. And yet the library dlsym() routine ** returns a void* which is really a pointer to a function. So how do we ** use dlsym() with -pedantic-errors? ** ** Variable x below is defined to be a pointer to a function taking ** parameters void* and const char* and returning a pointer to a function. ** We initialize x by assigning it a pointer to the dlsym() function. ** (That assignment requires a cast.) Then we call the function that ** x points to. ** ** This work-around is unlikely to work correctly on any system where ** you really cannot cast a function pointer into void*. But then, on the ** other hand, dlsym() will not work on such a system either, so we have ** not really lost anything.
65830. **comment:** ** CAPI3REF: Memory Allocator Statistics ** ** SQLite provides these two interfaces for reporting on the status ** of the [sqlite3_malloc()], [sqlite3_free()], and [sqlite3_realloc()] ** routines, which form the built-in memory allocation subsystem. ** ** ^The [sqlite3_memory_used()] routine returns the number of bytes ** of memory currently outstanding (malloced but not freed). ** ^The [sqlite3_memory_highwater()] routine returns the maximum ** value of [sqlite3_memory_used()] since the high-water mark ** was last reset. ^The values returned by [sqlite3_memory_used()] and ** [sqlite3_memory_highwater()] include any overhead ** added by SQLite in its implementation of [sqlite3_malloc()], ** but not overhead added by the any underlying system library ** routines

that [sqlite3_malloc()] may call. *** ^The memory high-water mark is reset to the current value of ** [sqlite3_memory_used()] if and only if the parameter to ** [sqlite3_memory_highwater()] is true. ^The value returned ** by [sqlite3_memory_highwater(1)] is the high-water mark ** prior to the reset.

label: code-design

65831. copy digits from after decimal to significand ** (decrease exponent by d to shift decimal right)

65832. JSON formatted without any backslash-escapes

65833. Mask of terms not to omit

65834. One of the journals pointed to by the master journal exists. ** Open it and check if it points at the master journal. If ** so, return without deleting the master journal file.

65835. Functions used to truncate the database file.

65836. ** Delete a linked list of TriggerStep structures.

65837. Level offering the most input segments

65838. Column names for INSERT

65839. ** Remember the current column cache context. Any new entries added ** added to the column cache after this call are removed when the ** corresponding pop occurs.

65840. OUT: Number of bytes consumed

65841. ** Set all the parameters in the compiled SQL statement to NULL.

65842. Expression to resolve. May be NULL.

65843. ERROR: unknown delta operator

65844. idlist ::= nm

65845. The P1 index cursor

65846. values

65847. Exponential notation. %e and %E

65848. ** Return true if it is necessary to write page *pPg into the sub-journal. ** A page needs to be written into the sub-journal if there exists one ** or more open savepoints for which: ** ** * The page-number is less than or equal to PagerSavepoint.nOrig, and ** * The bit corresponding to the page-number is not set in ** PagerSavepoint.pInSavepoint.

65849. ** Load the contents of the "averages" record from disk into the ** p->nTotalRow and p->aTotalSize[] variables. If successful, and if ** argument bCache is true, set the p->bTotalsValid flag to indicate ** that the contents of aTotalSize[] and nTotalRow are valid until ** further notice. ** * Return SQLITE_OK if successful, or an SQLite error code if an error ** occurs.

65850. The parsing (and code generating) context

65851. ** Given the name of a compile-time option, return true if that option ** was used and false if not. ** * The name can optionally begin with "SQLITE_" but the "SQLITE_" prefix ** is not required for a match.

65852. bUseCis

65853. ** PRAGMA [schema.]synchronous ** PRAGMA [schema.]synchronous=OFF|ON|NORMAL|FULL|EXTRA ** * Return or set the local value of the synchronous flag. Changing ** the local value does not make changes to the disk file and the ** default value will be restored the next time the database is ** opened.

65854. ** Arguments pLeft and pRight point to linked-lists of hash-entry objects, ** each sorted in key order. This function merges the two lists into a ** single list and returns a pointer to its first element.

65855. Position of token in zText

65856. If the bReplace flag is set, the change is an INSERT that has not ** been performed because the database already contains a row with the ** specified primary key and the conflict handler returned ** SQLITE_CHANGESET_REPLACE. In this case remove the conflicting row ** before reattempting the INSERT.

65857. ** The first argument points to a nul-terminated string containing a ** list of space separated integers. Read the first nOut of these into ** the array aOut[].

65858. List of free nodes. Linked by pParent.

65859. Flags originally passed to sqlite3WhereBegin()

65860. Create new statistic tables if they do not exist, or clear them ** if they do already exist.

65861. ***** End of mutex_unix.c *****

65862. ** Specify the activation key for a SEE database. Unless ** activated, none of the SEE routines will work.

65863. An instance of the BtreePayload object describes the content of a single ** entry in either an index or table btree. ** * Index btrees (used for indexes and also WITHOUT ROWID tables) contain ** an arbitrary key and no data. These btrees have pKey,nKey set to their ** key and pData,nData,nZero set to zero. ** * Table btrees (used for rowid tables) contain an integer rowid used as ** the key and passed in the nKey field. The pKey field is zero. ** pData,nData hold the content of the new entry. nZero extra zero bytes ** are appended to the end of the content when constructing the entry. ** * This object is used to pass information into sqlite3BtreeInsert(). The ** same information used to be passed as five separate parameters. But placing ** the information into this object helps to keep the interface more ** organized and understandable, and it also helps the resulting code to ** run a little faster by using fewer registers for parameter passing.

65864. Output vars

65865. ***** End of dbstat.c *****

65866. True if "!" flag is present

65867. OUT: Wal frame index of next page

65868. Value of 'pageno' column

65869. ** Insert a new row into the FTS content table.

65870. The thread handle

65871. ** If it has not already been allocated, allocate the UnpackedRecord ** structure at pTask->pUnpacked. Return SQLITE_OK if successful (or ** if no allocation was required), or SQLITE_NOMEM otherwise.

65872. ** Tokenizer callback used by implementation of highlight() function.

65873. a: p2<<28 | p4<<14 | p6 (unmasked)

65874. Register holding Current table

65875. Initial number of pages on free-list

65876. *****

65877. ** The set of routines that implement the simple tokenizer

65878. Index of next field to compare

65879. **comment:** ** Defragment the page given. This routine reorganizes cells within the ** page so that there are no free-blocks on the free-block list. ** * Parameter nMaxFrag is the maximum amount of fragmented space that may be ** present in the page after this routine returns. ** * EVIDENCE-OF: R-44582-60138 SQLite may from time to time reorganize a ** b-tree page so that there are no freeblocks or fragment bytes, all ** unused bytes are contained in the unallocated space region, and all ** cells are packed tightly at the end of the page.

label: code-design

65880. How to deal with SELECT result

65881. ** The expression object indicated by the second argument is guaranteed ** to be a scalar SQL function. If ** * all function arguments are SQL literals, ** * one of the SQLITE_FUNC_CONSTANT or _SLOCHNG function flags is set, and ** * the SQLITE_FUNC_NEEDCOLL function flag is not set, ** * then this routine attempts to invoke the SQL function. Assuming no ** error occurs, output parameter (*ppVal) is set to point to a value ** object containing the result before returning SQLITE_OK. ** * Affinity aff is applied to the result of the function before returning. ** If the result is a text value, the sqlite3_value object uses encoding ** enc. ** * If the conditions above are not met, this function returns SQLITE_OK ** and sets (*ppVal) to NULL. Or, if an error occurs, (*ppVal) is set to ** NULL and an SQLite error code returned.

65882. Bitmask of phrases already covered

65883. stl_prefix ::= seltblist joinop

65884. **comment:** ** variable-argument wrapper around sqlite3VXPrintf(). The bFlags argument ** can contain the bit SQLITE_PRINTF_INTERNAL enable internal formats.

label: code-design

65885. End of input

65886. True if holding a checkpoint lock

65887. Insert the new sample
65888. Skip VIEWS or VIRTUAL TABLES
65889. The parsed WHERE clause
65890. Special command to parse
65891. ** Advance the SegReader to point to the next docid in the doclist ** associated with the current term. *** If arguments ppOffsetList and pnOffsetList are not NULL, then ** *ppOffsetList is set to point to the first column-offset list ** in the doclist entry (i.e. immediately past the docid varint). *** *pnOffsetList is set to the length of the set of column-offset ** lists, not including the nul-terminator byte. For example:
65892. Namespace to resolve expressions in.
65893. Array of cell sizes
65894. Xfer function arguments to here
65895. 186
65896. LOOKUP
65897. IMP: R-48725-32206
65898. ** Register an unlock-notify callback. *** This is called after connection "db" has attempted some operation ** but has received an SQLITE_LOCKED error because another connection ** (call it pOther) in the same process was busy using the same shared ** cache. pOther is found by looking at db->pBlockingConnection. ** * If there is no blocking connection, the callback is invoked immediately, ** before this routine returns. *** If pOther is already blocked on db, then report SQLITE_LOCKED, to indicate ** a deadlock. *** Otherwise, make arrangements to invoke xNotify when pOther drops ** its locks. *** Each call to this routine overrides any prior callbacks registered ** on the same "db". If xNotify==0 then any prior callbacks are immediately ** cancelled.
65899. BTALLOC_EXACT, BTALLOC_LT, or BTALLOC_ANY
65900. Number of errors encountered
65901. Write the output into this preallocated buffer
65902. ** This C function implements an SQL user function that is used by SQL code ** generated by the ALTER TABLE ... RENAME command to modify the definition ** of any foreign key constraints that use the table being renamed as the ** parent table. It is passed three arguments: *** 1) The complete text of the CREATE TABLE statement being modified, ** 2) The old name of the table being renamed, and ** 3) The new name of the table being renamed. *** It returns the new CREATE TABLE statement. For example: *** sqlite_rename_parent('CREATE TABLE t1(a REFERENCES t2)', 't2', 't3') ** -> 'CREATE TABLE t1(a REFERENCES t3')'
65903. Number of elements in aSample[]
65904. SQLITE_STATUS_SCRATCH_SIZE
65905. Case 2.
65906. IN Loop terminator. OP_Next or OP_Prev
65907. When the schema cookie changes, record the new cookie internally
65908. ** Parameter zName points to a UTF-8 encoded string nName bytes long. ** Return the CollSeq* pointer for the collation sequence named zName ** for the encoding 'enc' from the database 'db'. ** * If the entry specified is not found and 'create' is true, then create a ** new entry. Otherwise return NULL. *** A separate function sqlite3LocateCollSeq() is a wrapper around ** this routine. sqlite3LocateCollSeq() invokes the collation factory ** if necessary and generates an error message if the collating sequence ** cannot be found. *** See also: sqlite3LocateCollSeq(), sqlite3GetCollSeq()
65909. Right subtree (larger entries) or list
65910. 212
65911. No. of ORDER BY terms
65912. Number of tables in pSrc->a[] to search
65913. Primary key flags array
65914. **comment:** No memory allocation is ever used on mem1. Prove this using ** the following assert(). If the assert() fails, it indicates a ** memory leak and a need to call sqlite3VdbeMemRelease(&mem1).
label: code-design
65915. Write to the unixFile structure here
65916. x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF
65917. File to write into
65918. ** The maximum number of times that a statement will try to reparse ** itself before giving up and returning SQLITE_SCHEMA.
65919. ** Open the file indicated and write a log of all unfreed memory ** allocations into that log.
65920. Figure out how much reusable memory is available at the end of the ** opcode array. This extra memory will be reallocated for other elements ** of the prepared statement.
65921. 1-byte (or larger) unsigned integer
65922. Token counter
65923. 730
65924. 0x11
65925. A character outside of the ascii range. Skip past it if it is ** a separator character. Or break out of the loop if it is not.
65926. Stub function when INCRBLOB is omitted
65927. **comment:** ** 2015-06-06 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** ***** This module contains C code that generates VDBE code used to process ** the WHERE clause of SQL statements. ** ** This file was split off from where.c on 2015-06-06 in order to reduce the ** size of where.c and make it easier to edit. This file contains the routines ** that actually generate the bulk of the WHERE loop code. The original where.c ** file retains the code that does query planning and analysis.
label: code-design
65928. End the loop over all WhereLoops from outer-most down to inner-most
65929. Write the master journal name into the journal file. If a master ** journal file name has already been written to the journal file, ** or if zMaster is NULL (no master journal), then this call is a no-op.
65930. isPCacheInit
65931. ** Class derived from sqlite3_tokenizer
65932. Return the value to pass to a sqlite3_wal_hook callback, the ** number of frames in the WAL at the point of the last commit since ** sqlite3WalCallback() was called. If no commits have occurred since ** the last call, then return 0.
65933. ** An instance of the following structure contains all information ** needed to generate code for a single SELECT statement. *** nLimit is set to -1 if there is no LIMIT clause. nOffset is set to 0. ** If there is a LIMIT clause, the parser sets nLimit to the value of the ** limit and nOffset to the value of the offset (or 0 if there is not ** offset). But later on, nLimit and nOffset become the memory locations ** in the VDBE that record the limit and offset counters. *** addrOpenEphm[] entries contain the address of OP_OpenEphemeral opcodes. ** These addresses must be stored so that we can go back and fill in ** the P4_KEYINFO and P2 parameters later. Neither the KeyInfo nor ** the number of columns in P2 can be computed at the same time ** as the OP_OpenEphm instruction is coded because not ** enough information about the compound query is known at that point. ** The KeyInfo for addrOpenTran[0] and [1] contains collating sequences ** for the result set. The KeyInfo for addrOpenEphm[2] contains collating ** sequences for the ORDER BY clause.
65934. **comment:** TODO: This definition is just included so other modules compile. It ** needs to be revisited.
label: code-design
65935. ** Iterate forwards through a doclist.
65936. The table pTrigger is attached to
65937. New colset object to return
65938. For each column used in source tables
65939. Edit this page
65940. ** datetime(Timestring, MOD, MOD, ...) ** ** Return YYYY-MM-DD HH:MM:SS
65941. ** This method is called to "rewind" the stmt_cursor object back ** to the first row of output. This method is always called at least ** once prior to any call to stmtColumn() or stmtRowid() or ** stmtEof().
65942. defined(SQLITE_ENABLE_FTS3) || defined(SQLITE_ENABLE_FTS4)

65943. ** Set or release locks on the WAL. Locks are either shared or exclusive. ** A lock cannot be moved directly between shared and exclusive - it must go ** through the unlocked state first. *** In locking_mode=EXCLUSIVE, all of these routines become no-ops.

65944. 114

65945. Version 3.20.0 and later

65946. Figure out how many new bytes are in this term

65947. ** Drop all shadow tables. Return SQLITE_OK if successful or an SQLite error ** code otherwise.

65948. If no error occurred, set the output variables.

65949. Dequoted copy of token p

65950. Changeset iterator

65951. Store 1 if the result is not empty

65952. The writer

65953. Foreign key to find index for

65954. RESTRICT

65955. Sort order for each column.

65956. TRANS_NONE, TRANS_READ or TRANS_WRITE

65957. Text to add to EQP output

65958. If a mutex is required for normal operation, allocate one

65959. ** The original API set ends here. All extensions can call any ** of the APIs above provided that the pointer is not NULL. But ** before calling APIs that follow, extension should check the ** sqlite3_libversion_number() to make sure they are dealing with ** a library that is new enough to support that API.

65960. Token number of previous deferred token

65961. **comment:** Go back under the static mutex and clean up the recursive ** mutex to prevent a resource leak.

label: code-design

65962. Query for next idx at level iLevel

65963. Noop

65964. The code within this loop is run only once if the 'searchList' variable ** is not true. Otherwise, it runs once for each trunk-page on the ** free-list until the page 'nearby' is located (eMode==BTALLOC_EXACT) ** or until a page less than 'nearby' is located (eMode==BTALLOC_LT)

65965. (320) database_kw_opt ::=

65966. Columns of index used so far

65967. Name of the conch file

65968. Don't bother checking for NOT NULL on columns that do not change

65969. ** EVIDENCE-OF: R-25715-37072 Memory allocation statistics are enabled by ** default unless SQLite is compiled with
SQLITE_DEFAULT_MEMSTATUS=0 in ** which case memory allocation statistics are disabled by default.

65970. ** Determine if we are dealing with Windows CE - which has a much reduced ** API.

65971. True if zTerm/nTerm is transient

65972. ** Streaming version of sqlite3changeset_concat().

65973. BEFORE

65974. ** Extract a 2-byte big-endian integer from an array of unsigned bytes. ** But if the value is zero, make it 65536. *** This routine is used to extract the "offset to cell content area" value ** from the header of a btree page. If the page size is 65536 and the page ** is empty, the offset should be 65536, but the 2-byte value stores zero. ** This routine makes the necessary adjustment to 65536.

65975. Input vars

65976. term column

65977. ** Free all PgHdr objects stored in the Pager.pMmapFreelist list.

65978. True to seek past initial nulls

65979. Next winShm with the same winShmNode

65980. ***** End of mutex_noop.c *****

65981. The original LIKE operator

65982. ORDER BY clause

65983. ** Figure out the number of pages needed to hold all b.nCell cells. ** Store this number in "k". Also compute szNew[] which is the total ** size of all cells on the i-th page and cntNew[] which is the index ** in b.apCell[] of the cell that divides page i from page i+1. ** cntNew[k] should equal b.nCell. *** Values computed by this block: *** k: The total number of sibling pages ** szNew[i]: Spaced used on the i-th sibling page. ** cntNew[i]: Index in b.apCell[] and b.szCell[] for the first cell to ** the right of the i-th sibling page. ** usableSpace: Number of bytes of space available on each sibling. **

65984. Opcode: IdxGT P1 P2 P3 P4 P5 ** Synopsis: key=r[P3@P4] *** The P4 register values beginning with P3 form an unpacked index ** key that omits the PRIMARY KEY. Compare this key value against the index ** that P1 is currently pointing to, ignoring the PRIMARY KEY or ROWID ** fields at the end. *** If the P1 index entry is greater than the key value ** then jump to P2. Otherwise fall through to the next instruction.

65985. If the CREATE VIRTUAL TABLE statement is being entered for the ** first time (in other words if the virtual table is actually being ** created now instead of just being read out of sqlite_master) then ** do additional initialization work and store the statement text ** in the sqlite_master table.

65986. Locate the statement handle used to insert data into the %_content ** table. The SQL for this statement is: ** ** INSERT INTO %_content VALUES(?, ?, ?, ...) ** ** The statement features N '?' variables, where N is the number of user ** defined columns in the FTS3 table, plus one for the docid field.

65987. Full-text index

65988. The main db file

65989. Bypass the creation of the PRIMARY KEY btree and the sqlite_master ** table entry. This is only required if currently generating VDBE ** code for a CREATE TABLE (not when parsing one as part of reading ** a database schema).

65990. First argument to xCodec... methods

65991. Delete after closing if true

65992. True once an index has been found

65993. ** Set up the lookaside buffers for a database connection. ** Return SQLITE_OK on success. ** If lookaside is already active, return SQLITE_BUSY. *** The sz parameter is the number of bytes in each lookaside slot. ** The cnt parameter is the number of slots. If pStart is NULL the ** space for the lookaside memory is obtained from sqlite3_malloc(). ** If pStart is not NULL then it is sz*cnt bytes of memory to use for ** the lookaside memory.

65994. ** If the expression passed as the second argument is a vector, generate ** code to write the first nReg elements of the vector into an array ** of registers starting with iReg. *** If the expression is not a vector, then nReg must be passed 1. In ** this case, generate code to evaluate the expression and leave the ** result in register iReg.

65995. If the rhs of the LIKE expression is a variable, and the current ** value of the variable means there is no need to invoke the LIKE ** function, then no OP_Variable will be added to the program. ** This causes problems for the sqlite3_bind_parameter_name() ** API. To work around them, add a dummy OP_Variable here.

65996. ** The text of the conversion is pointed to by "bufpt" and is ** "length" characters long. The field width is "width". Do ** the output.

65997. True if this is a commit

65998. ** CAPI3REF: Set Or Clear the Indirect Change Flag *** Each change recorded by a session object is marked as either direct or ** indirect. A change is marked as indirect if either: *** ** The session object "indirect" flag is set when the change is ** made, or ** The change is made by an SQL trigger or foreign key action ** instead of directly as a result of a users SQL statement. ** *** If a single row is affected by more than one operation within a session, ** then the change is considered indirect if all operations meet the criteria ** for an indirect change above, or direct otherwise. *** This function is used to set, clear or query the session object indirect ** flag. If the second argument passed to this function is zero, then the ** indirect flag is cleared. If it is greater than zero, the indirect flag ** is set. Passing a value less than zero does not modify the current value ** of the indirect flag, and may be used to query the current state of the ** indirect flag for the specified session object. *** The return value indicates the final state of the indirect flag: 0 if ** it is clear, or 1 if it is set.

65999. ** Delete a WhereLoop object

66000. **comment:** ** Macro to return the number of elements in an array. SQLite has a ** similar macro called ArraySize(). Use a different name to avoid ** a collision when building an amalgamation with built-in FTS3.

label: code-design

66001. True if this is a read-only file
66002. **comment:** ** FTS4 is really an extension for FTS3. It is enabled using the ** SQLITE_ENABLE_FTS3 macro. But to avoid confusion we also all ** the SQLITE_ENABLE_FTS4 macro to serve as an alias for SQLITE_ENABLE_FTS3.
label: code-design
66003. Column token must occur in
66004. "columnsize=" option value (dflt==1)
66005. Page size for host database
66006. If all of the OR-connected terms are optimized using the same ** index, and the index is opened using the same cursor number ** by each call to sqlite3WhereBegin() made by this loop, it may ** be possible to use that index as a covering index. ** ** If the call to sqlite3WhereBegin() above resulted in a scan that ** uses an index, and this is either the first OR-connected term ** processed or the index is the same as that used by all previous ** terms, set pCov to the candidate covering index. Otherwise, set ** pCov to NULL to indicate that no candidate covering index will ** be available.
66007. 1-byte unsigned integer
66008. **comment:** ** Deserialize the data blob pointed to by buf as serial type serial_type ** and store the result in pMem. Return the number of bytes read. ** ** This function is implemented as two separate routines for performance. ** The few cases that require local variables are broken out into a separate ** routine so that in most cases the overhead of moving the stack pointer ** is avoided.
label: code-design
66009. Bounding box for pNode
66010. Bytes of payload stored locally
66011. Index in p->aIndex[] to merge
66012. ** Return an approximation for the amount of memory currently used ** by all pagers associated with the given database connection. The ** highwater mark is meaningless and is returned as zero.
66013. Behavioral bits. UNIXFILE_* flags
66014. Get shared locks at the system level, if necessary
66015. ** Shorthand for starting a new tree item that consists of a single label
66016. The key
66017. Index of database housing pTab
66018. Analyze the schema named as the argument
66019. 104
66020. Root page of the open btree cursor
66021. 32
66022. (325) vtabarg ::= vtabarg vtabargtoken
66023. First argument to disposal method
66024. ** Configure the VM passed as the first argument with an ** sqlite3_stmt_scanstatus() entry corresponding to the scan used to ** implement level pLvl. Argument pSrcList is a pointer to the FROM ** clause that the scan reads data from. ** ** If argument addrExplain is not 0, it must be the address of an ** OP_Explain instruction that describes the same loop.
66025. Language id to merge
66026. **comment:** If no error has occurred and pPage has an overflow cell, call balance() ** to redistribute the cells within the tree. Since balance() may move ** the cursor, zero the BtCursor.info.nSize and BTCF_ValidNKey ** variables. ** ** Previous versions of SQLite called moveToRoot() to move the cursor ** back to the root page as balance() used to invalidate the contents ** of BtCursor.apPage[] and BtCursor.aiIdx[]. Instead of doing that, ** set the cursor state to "invalid". This makes common insert operations ** slightly faster. ** ** There is a subtle but important optimization here too. When inserting ** multiple records into an intkey b-tree using a single cursor (as can ** happen while processing an "INSERT INTO ... SELECT" statement), it ** is advantageous to leave the cursor pointing to the last entry in ** the b-tree if possible. If the cursor is left pointing to the last ** entry in the table, and the next row inserted has an integer key ** larger than the largest existing key, it is possible to insert the ** row without seeking the cursor. This can be a big performance boost.
label: code-design
66027. wrFlag = (wrFlag ? 1 : 0);
66028. The extra pAppData, if any.
66029. A sub-task in the sort process
66030. This branch is taken when committing a transaction in rollback-journal ** mode if the database file on disk is larger than the database image. ** At this point the journal has been finalized and the transaction ** successfully committed, but the EXCLUSIVE lock is still held on the ** file. So it is safe to truncate the database file to its minimum ** required size.
66031. ** CAPI3REF: Test if a changeset has recorded any changes. ** ** Return non-zero if no changes to attached tables have been recorded by ** the session object passed as the first argument. Otherwise, if one or ** more changes have been recorded, return zero. ** ** Even if this function returns zero, it is possible that calling ** [sqlite3session_changeset()] on the session handle may still return a ** changeset that contains no changes. This can happen when a row in ** an attached table is modified and then later on the original values ** are restored. However, if this function returns non-zero, then it is ** guaranteed that a call to sqlite3session_changeset() will return a ** changeset containing zero changes.
66032. ** The name of the schema table.
66033. Acquire an EXCLUSIVE lock
66034. The "OUTER" keyword is present
66035. Offset of first type in header
66036. Bitvec to ensure pages played back only once
66037. synopsis: key=r[P3@P4]
66038. Used to iterate from 0 to nCol
66039. Index of first cell currently on page
66040. Iterator variable
66041. ** Begin a read transaction on the database. ** ** This routine used to be called sqlite3OpenSnapshot() and with good reason: ** it takes a snapshot of the state of the WAL and wal-index for the current ** instant in time. The current thread will continue to use this snapshot. ** Other threads might append new content to the WAL and wal-index but ** that extra content is ignored by the current thread. ** ** If the database contents have changes since the previous read ** transaction, then *pChanged is set to 1 before returning. The ** Pager layer will use this to know that its cache is stale and ** needs to be flushed.
66042. Index loop counter
66043. True if the only wildcard is % in the last character
66044. Offset within buffer to read from
66045. TODO2: Doclist terminator written here.
66046. Filter out attempts to run UPDATE or DELETE on contentless tables. ** This is not supported.
66047. True if doclists are in reverse order
66048. P5ErrMsg type
66049. **comment:** pRHS cannot be NULL because a malloc error would have been detected ** before now and control would have never reached this point
label: code-design
66050. ** Change the size of an existing memory allocation. ** ** The outer layer memory allocator prevents this routine from ** being called with pPrior==0. ** ** nBytes is always a value obtained from a prior call to ** memsys5Round(). Hence nBytes is always a non-negative power ** of two. If nBytes==0 that means that an oversize allocation ** (an allocation larger than 0x40000000) was requested and this ** routine should return 0 without freeing pPrior.
66051. Affinity string for comparisons
66052. If the tree is a leaf-data tree, and the siblings are leaves, ** then there is no divider cell in b.apCell[]. Instead, the divider ** cell consists of the integer key for the right-most cell of ** the sibling-page assembled above only.
66053. lock semaphore now but bail out when already locked.
66054. ** Write an error message into pParse->zErrMsg that explains that the ** user-supplied authorization function returned an illegal value.
66055. ** Delete all the content of a Select structure. Deallocate the structure ** itself only if bFree is true.
66056. ***** Begin file sqlite3session.c *****
66057. OUT: Byte offset of end of token in input buffer

66058. Pointer to position list for phrase
66059. Skip over multiple "*" characters in the pattern. If there ** are also "?" characters, skip those as well, but consume a ** single character of the input string for each "?" skipped
66060. Deleting *pRowid leaves the table empty
66061. Bit N (1<N) set if column N of pTab is used
66062. 0x60..0x6F
66063. ** Indices of various locking bytes. WAL_NREADER is the number ** of available reader locks and should be at least 3. The default ** is SQLITE_SHM_NLOCK==8 and WAL_NREADER==5.
66064. Key associated with this element
66065. Key columns for SRT_Queue and SRT_DistQueue
66066. SQLITE_BUSY
66067. EVIDENCE-OF: R-13523-04394 The second integer on a freelist trunk page ** is the number of leaf page pointers to follow.
66068. A table being constructed by CREATE TABLE
66069. 4-byte aligned
66070. First element
66071. NULL=NULL
66072. Ensure the default expression is something that sqlite3ValueFromExpr() ** can handle (i.e. not CURRENT_TIME etc.)
66073. **comment:** ** Create the byte sequence used to represent a cell on page pPage ** and write that byte sequence into pCell[]. Overflow pages are ** allocated and filled in as necessary. The calling procedure ** is responsible for making sure sufficient space has been allocated ** for pCell[]. *** Note that pCell does not necessary need to point to the pPage->aData ** area. pCell might point to some temporary storage. The cell will ** be constructed in this temporary area then copied into pPage->aData ** later.
label: code-design
66074. Array of SegReader objects
66075. A SessionChange object never has a NULL value in a PK column
66076. Next element in list of dirty pages
66077. Operation terminated by sqlite3_interrupt()
66078. Top of the step-6 loop
66079. ** This function is an xTokenize() callback used by the auxiliary snippet() ** function. Its job is to identify tokens that are the first in a sentence. ** For each such token, an entry is added to the SFinder.aFirst[] array.
66080. Count the number of terms in the query
66081. ** Implementation of the special optimize() function for FTS3. This ** function merges all segments in the database to a single segment. ** Example usage is: **
** SELECT optimize(t) FROM t LIMIT 1; *** where 't' is the name of an FTS3 table.
66082. This function is always called by the main user thread. *** If this function is being called after SorterRewind() has been called, ** it is possible that thread pSorter->aTask[pSorter->nTask-1].pThread ** is currently attempt to join one of the other threads. To avoid a race ** condition where this thread also attempts to join the same object, join ** thread pSorter->aTask[pSorter->nTask-1].pThread first.
66083. CLEAN pages not on dirty list
66084. A modifier of the form (+-)HH:MM:SS.FFF adds (or subtracts) the ** specified number of hours, minutes, seconds, and fractional seconds ** to the time. The ".FFF" may be omitted. The ":"SS.FFF" may be ** omitted.
66085. Recalculate checksums within the wal file if required.
66086. When setting the auto_vacuum mode to either "full" or ** "incremental", write the value of meta[6] in the database ** file. Before writing to meta[6], check that meta[3] indicates ** that this really is an auto-vacuum capable database.
66087. ** CAPI3REF: Prepare Flags *** These constants define various flags that can be passed into ** "prepFlags" parameter of the [sqlite3_prepare_v3()] and ** [sqlite3_prepare16_v3()] interfaces. *** New flags may be added in future releases of SQLite. ** <dl> ** [[SQLITE_PREPARE_PERSISTENT]]
^<dt>SQLITE_PREPARE_PERSISTENT</dt> ** <dd>The SQLITE_PREPARE_PERSISTENT flag is a hint to the query planner ** that the prepared statement will be retained for a long time and ** probably reused many times.)^<dd>Without this flag, [sqlite3_prepare_v3()] ** and [sqlite3_prepare16_v3()] assume that the prepared statement will ** be used just once or at most a few times and then destroyed using ** [sqlite3_finalize()] relatively soon. The current implementation acts ** on this hint by avoiding the use of [lookaside memory] so as not to ** deplete the limited store of lookaside memory. Future versions of ** SQLite may act on this hint differently. ** </dl>
66088. ** End of interface to code in fts5_unicode2.c. *****
66089. 0 1 2 3 4 5 6 7 8 9 A B C D E F
66090. ** Release an array of N Mem elements
66091. Number of cols in current table
66092. ** When called, *ppPoslist must point to the byte immediately following the ** end of a position-list. i.e. ((*ppPoslist)[-1]==POS_END). This function ** moves *ppPoslist so that it instead points to the first byte of the ** same position list.
66093. Number of index columns including rowid
66094. Nul-terminated UTF-8 string <value>, or NULL
66095. ***** Begin file notify.c *****
66096. EXPLAIN => ID
66097. ** CAPI3REF: Authorizer Return Codes *** The [sqlite3_set_authorizer | authorizer callback function] must ** return either [SQLITE_OK] or one of these two constants in order ** to signal SQLite whether or not the action is permitted. See the ** [sqlite3_set_authorizer | authorizer documentation] for additional ** information. *** Note that SQLITE_IGNORE is also used as a [conflict resolution mode] ** returned from the [sqlite3_vtab_on_conflict()] interface.
66098. True if this is likely an append
66099. The pragma virtual table object
66100. SQLITE_READONLY
66101. Database number to create the table in
66102. ** End of interface to code in fts5_storage.c. *****
66103. Position of token within column
66104. SQLITE OMIT_FLAG_PRAGMAS
66105. ** A structure for holding a single date and time.
66106. ** The cursor "p" has a pending seek operation that has not yet been ** carried out. Seek the cursor now. If an error occurs, return ** the appropriate error code.
66107. Next field of pPKey2 to compare
66108. DATABASE
66109. ignore IS [NOT] NULL constraints on NOT NULL columns
66110. Figure out how many columns of data are supplied. If the data ** is coming from a SELECT statement, then generate a co-routine that ** produces a single row of the SELECT on each invocation. The ** co-routine is the common header to the 3rd and 4th templates.
66111. Fts3SegReader object to return
66112. c8..cf,
66113. The allocateSpace() routine guarantees the following properties ** if it returns successfully
66114. 8-byte unsigned integer
66115. ***** Begin file fts3.c *****
66116. **comment:** * If compiled with SQLITE_WIN32_MALLOC on Windows, we will use the * various Win32 API heap functions instead of our own.
label: code-design
66117. True if we acquired a PENDING lock this time
66118. Building an imposter table
66119. Next phrase instance index
66120. Ignore DESC
66121. First allowable cell or freeblock offset
66122. For looping over all siblings

66123. EVIDENCE-OF: R-02982-34736 In order to maintain full backwards ** compatibility for legacy applications, the URI filename capability is ** disabled by default. *** EVIDENCE-OF: R-38799-08373 URI filenames can be enabled or disabled ** using the SQLITE_USE_URI=1 or SQLITE_USE_URI=0 compile-time options. *** EVIDENCE-OF: R-43642-56306 By default, URI handling is globally ** disabled. The default value may be changed by compiling with the ** SQLITE_USE_URI symbol defined. *** URI filenames are enabled by default if SQLITE_HAS_CODEC is ** enabled.

66124. **comment:** Lookaside malloc configuration
label: code-design

66125. SQLITE_STATUS_MEMORY_USED

66126. Exactly one of regOld and regNew should be non-zero.

66127. ** Register the query expression parser test function fts3_exprtest() ** with database connection db.

66128. SELECT rowid, * FROM ... ORDER BY 1 DESC

66129. An offset into mem5.aCtrl[]

66130. **comment:** ** When this is called the journal file for pager pPager must be open. ** This function attempts to read a master journal file name from the ** end of the file and, if successful, copies it into memory supplied ** by the caller. See comments above writeMasterJournal() for the format ** used to store a master journal file name at the end of a journal file. *** zMaster must point to a buffer of at least nMaster bytes allocated by ** the caller. This should be sqlite3_vfs.mxPathname+1 (to ensure there is ** enough space to write the master journal name). If the master journal ** name in the journal is longer than nMaster bytes (including a ** nul-terminator), then this is handled as if no master journal name ** were present in the journal. *** If a master journal file name is present at the end of the journal ** file, then it is copied into the buffer pointed to by zMaster. A ** nul-terminator byte is appended to the buffer following the master ** journal file name. *** If it is determined that no master journal file name is present ** zMaster[0] is set to 0 and SQLITE_OK returned. *** If an error occurs while reading from the journal file, an SQLite ** error code is returned.
label: code-design

66131. ** An instance of this structure is used to create a segment b-tree in the ** database. The internal details of this type are only accessed by the ** following functions: *** fts3SegWriterAdd() ** fts3SegWriterFlush() ** fts3SegWriterFree()

66132. FROM

66133. ** Allocate a new simple tokenizer. Return a pointer to the new ** tokenizer in *ppModule

66134. SQLITE OMIT_BTREECOUNT

66135. ** Allowed values for sqlite3_rtree_query.eWithin and .eParentWithin.

66136. Characters. %c

66137. ** Add all WhereLoop objects for a single table of the join where the table ** is identified by pBuilder->pNew->iTab. That table is guaranteed to be ** a b-tree table, not a virtual table. *** The costs (WhereLoop.rRun) of the b-tree loops added by this function ** are calculated as follows: *** For a full scan, assuming the table (or index) contains nRow rows: *** cost = nRow * 3.0 // full-table scan ** cost = nRow * K // scan of covering index ** cost = nRow * (K+3.0) // scan of non-covering index *** where K is a value between 1.1 and 3.0 set based on the relative ** estimated average size of the index and table records. *** For an index scan, where nVisit is the number of index rows visited ** by the scan, and nSeek is the number of seek operations required on ** the index b-tree: *** cost = nSeek * (log(nRow) + K * nVisit) // covering index ** cost = nSeek * (log(nRow) + (K+3.0) * nVisit) // non-covering index *** Normally, nSeek is 1. nSeek values greater than 1 come about if the ** WHERE clause includes "x IN (...) terms used in place of "x=?". Or when ** implicit "x IN (SELECT x FROM tbl)" terms are added for skip-scans. *** The estimated values (nRow, nVisit, nSeek) often contain a large amount ** of uncertainty. For this reason, scoring is designed to pick plans that ** "do the least harm" if the estimates are inaccurate. For example, a ** log(nRow) factor is omitted from a non-covering index scan in order to ** bias the scoring in favor of using an index, since the worst-case ** performance of using an index is far better than the worst-case performance ** of a full table scan.

66138. **comment:** ** An instance of the following object records the location of ** each unused scratch buffer.
label: code-design

66139. LHS is entirely deferred. So we assume it matches every row. ** Advance the RHS iterator to find the next row visited.

66140. File to read from

66141. ** If the numeric type of argument pVal is "integer", then return it ** converted to a 64-bit signed integer. Otherwise, return a copy of ** the second parameter, iDefault.

66142. Default ON CONFLICT policy

66143. Conversion paradigm

66144. Size of aSeg[] array

66145. ***** Continuing where we left off in fts3Int.h *****

66146. ** Factor out the code of the given expression to initialization time. ** If regDest>=0 then the result is always stored in that register and the ** result is not reusable. If regDest<0 then this routine is free to ** store the value wherever it wants. The register where the expression ** is stored is returned. When regDest<0, two identical expressions will ** code to the same register.

66147. Mask of exclusive locks held

66148. True for recursive reference in WITH

66149. Underlying file handle

66150. A-overwrites-R

66151. **comment:** This case runs if the aggregate has no GROUP BY clause. The ** processing is much simpler since there is only a single row ** of output.
label: code-design

66152. b.apCell

66153. ** Interface to the testing logic.

66154. Variables used only by sqlite3Fts5PoslistIterXXX() functions.

66155. nearphrases

66156. ** Add an entry for token pToken to the pCsr->pDeferred list.

66157. Errno value from last system error

66158. If the contents of this write should be stored in memory

66159. If no zName is given, restore all system calls to their default ** settings and return NULL

66160. INSERT + UPDATE

66161. Use the LIMIT in cost estimates

66162. Iterate through the contents of the WAL, copying data to the db file

66163. ** Deallocate all memory associated with a WhereOrInfo object.

66164. Iterator used while populating OLD.*

66165. 60..67 `abcdefg

66166. Task that owns this merger

66167. ** CAPI3REF: Virtual Table Cursor Object ** KEYWORDS: sqlite3_vtab_cursor {virtual table cursor} ** Every [virtual table module] implementation uses a subclass of the ** following structure to describe cursors that point into the ** [virtual table] and are used ** to loop through the virtual table. Cursors are created using the ** [sqlite3_module.xOpen | xOpen] method of the module and are destroyed ** by the [sqlite3_module.xClose | xClose] method. Cursors are used ** by the [xFILTER], [xNext], [xEof], [xColumn], and [xRowid] methods ** of the module. Each module implementation will define ** the content of a cursor structure to suit its own needs. *** This superclass exists in order to define fields of the cursor that ** are common to all implementations.

66168. Drop all triggers associated with the table being dropped. Code ** is generated to remove entries from sqlite_master and/or ** sqlite_temp_master if required.

66169. ** The next global variable is incremented each time the OP_Sort opcode ** is executed. The test procedures use this information to make sure that ** sorting is occurring or not occurring at appropriate times. This variable ** has no function other than to help verify the correct operation of the ** library.

66170. 300

66171. Tokens count up to end of this phrase

66172. Mapping of columns in pFrom to columns in zTo

66173. String accumulator

66174. ** This is a utility routine, useful to VFS implementations, that checks ** to see if a database file was a URI that contained a specific query ** parameter, and if so obtains the value of the query parameter. *** The zFilename argument is the filename pointer passed into the xOpen() ** method of a VFS implementation. The zParam argument is the name of the ** query parameter we seek. This routine returns the value of the zParam ** parameter if it exists. If the parameter does not exist, this routine ** returns a NULL pointer.

66175. OUT: stats written here
66176. **comment:** ** Compute a string length that is limited to what can be stored in ** lower 30 bits of a 32-bit signed integer. *** The value returned will never be negative. Nor will it ever be greater ** than the actual length of the string. For very long strings (greater ** than 1GiB) the value returned might be less than the true string length.
label: code-design
66177. If the PRAGMA command was of the form "PRAGMA <db>.integrity_check", ** then iDb is set to the index of the database identified by <db>. ** In this case, the integrity of database iDb only is verified by ** the VDBE created below. *** Otherwise, if the command was simply "PRAGMA integrity_check" (or ** "PRAGMA quick_check"), then iDb is set to 0. In this case, set iDb ** to -1 here, to indicate that the VDBE should verify the integrity ** of all attached databases.
66178. ** The output variable *ppExpr is populated with an allocated Fts3Expr ** structure, or set to 0 if the end of the input buffer is reached. *** Returns an SQLite error code. SQLITE_OK if everything works, SQLITE_NOMEM ** if a malloc failure occurs, or SQLITE_ERROR if a parse error is encountered. ** If SQLITE_ERROR is returned, pContext is populated with an error message.
66179. Opcode: Once P1 P2 * * * Fall through to the next instruction the first time this opcode is ** encountered on each invocation of the byte-code program. Jump to P2 ** on the second and all subsequent encounters during the same invocation. *** Top-level programs determine first invocation by comparing the P1 ** operand against the P1 operand on the OP_Init opcode at the beginning ** of the program. If the P1 values differ, then fall through and make ** the P1 of this opcode equal to the P1 of OP_Init. If P1 values are ** the same then take the jump. *** For subprograms, there is a bitmask in the VdbeFrame that determines ** whether or not the jump should be taken. The bitmask is necessary ** because the self-altering code trick does not work for recursive ** triggers.
66180. mxPathname
66181. Strings with \" doubled. %q
66182. OUT: Best snippet found
66183. ** Walk an expression tree. Return non-zero if the expression is constant ** for any single row of the table with cursor iCur. In other words, the ** expression must not refer to any non-deterministic function nor any ** table other than iCur.
66184. !SQLITE OMIT_SHARED_CACHE
66185. ** Implementation of xCommit() method. This is a no-op. The contents of ** the pending-terms hash-table have already been flushed into the database ** by fts3SyncMethod().
66186. Absolute level of segment to modify
66187. Read in the complete content of the index entry
66188. Opaque type used by code in vdbeSort.c
66189. Do extra syncs of the journal for robustness
66190. Case 3: We have an inequality comparison against the ROWID field.
66191. If this is an INSERT or UPDATE operation, insert the new record.
66192. ifnotexists ::= IF NOT EXISTS
66193. ** Return the name of the Nth column of the result set returned by SQL ** statement pStmt.
66194. Pointer to the conversion buffer
66195. Total number of ordered columns in the index
66196. The number of tables in the FROM clause is limited by the number of ** bits in a Bitmask
66197. Pointer to the index object
66198. isSavept is 0 or 1
66199. Size of zTerm in bytes
66200. ** Implementation of the non-aggregate min() and max() functions
66201. ** 2008 June 18 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** ***** This module implements the sqlite3_status() interface and related ** functionality.
66202. OP_OpenEphem opcodes related to this select
66203. ** Free a parsed fts3 query expression allocated by sqlite3Fts3ExprParse(). ** ** This function would be simpler if it recursively called itself. But ** that would mean passing a sufficiently large expression to ExprParse() ** could cause a stack overflow.
66204. The "break" point is here, just past the end of the outer loop. ** Set it.
66205. ** Check if the program stored in the VM associated with pParse may ** throw an ABORT exception (causing the statement, but not entire transaction ** to be rolled back). This condition is true if the main program or any ** sub-programs contains any of the following: *** * OP_Halt with P1=SQLITE_CONSTRAINT and P2=OE_Abort. *** OP_HaltIfNull with P1=SQLITE_CONSTRAINT and P2=OE_Abort. *** OP_Destroy *** OP_VUpdate *** OP_VRename *** OP_FkCounter with P2=0 (immediate foreign key constraint) *** OP_CreateTable and OP_InitCoroutine (for CREATE TABLE AS SELECT ...) *** Then check that the value of Parse.mayAbort is true if an ** ABORT may be thrown, or false otherwise. Return true if it does ** match, or false otherwise. This function is intended to be used as ** part of an assert statement in the compiler. Similar to: *** assert(sqlite3VdbeAssertMayAbort(pParse->pVdbe, pParse->mayAbort));
66206. Fill in the Pager.zFilename and Pager.zJournal buffers, if required.
66207. Do not discard pages from an in-memory database since we might ** need to rollback later. Just move the page out of the way.
66208. Error messages will be written here
66209. True to ignore empty segments
66210. Name of database iDb
66211. .z =
66212. ** Release a lock acquired by winceMutexAcquire()
66213. Return value from subprocedures
66214. One of: TK_UNION TK_ALL TK_INTERSECT TK_EXCEPT
66215. 1360
66216. ** Return true if the floating point value is Not a Number (NaN). *** Use the math library isnan() function if compiled with SQLITE_HAVE_ISNAN. ** Otherwise, we have our own implementation that works on most systems.
66217. ** Return the size in bytes of a PCache object.
66218. Constant inputs give a constant output
66219. OUT: term (nul-terminated)
66220. True if current term already output
66221. New apRegion[] array
66222. SQLITE_DEBUG
66223. VACUUM => ID
66224. Size of pKey for indexes. PRIMARY KEY for tabs
66225. ** "LIMIT -1" always shows all rows. There is some ** controversy about what the correct behavior should be. ** The current implementation interprets "LIMIT 0" to mean ** no rows.
66226. Number of entries in aFunc[]
66227. sqlite3_step() has another row ready
66228. Best match found so far
66229. Various flags
66230. exponent
66231. xRelease
66232. Either: *** 1) an error has occurred, or ** 2) the iterator points to EOF, or ** 3) the iterator points to an entry with term (pTerm/nTerm), or ** 4) the FTS5INDEX_QUERY_SCAN flag was set and the iterator points ** to an entry with a term greater than or equal to (pTerm/nTerm).
66233. Pointer to disk image of the page data
66234. Accumulate the error message text here
66235. The table we are writing to
66236. Sanity checking on calling parameters

66237. 260
66238. 40
66239. Used by: function_list
66240. The trigger containing the target token
66241. OK to factor out constants
66242. One of the WHERE_DISTINCT_* operators
66243. TRUE if currently initializing
66244. **comment:** Number of temporary registers in aTempReg[]
 label: code-design
66245. Segment to iterate through
66246. ** Return the most recent error code generated by an SQLite routine. If NULL is ** passed to this function, we assume a malloc() failed during sqlite3_open().
66247. Offset in CREATE TABLE stmt to add a new column
66248. Name of the other table
66249. ** Functions for accessing sqlite3_file methods
66250. Register all built-in functions, but do not attempt to read the ** database schema yet. This is delayed until the first time the database ** is accessed.
66251. Database connection for this stmt vtab
66252. Attempt to leave results in this register
66253. conslist_opt ::=
66254. Begin reading this far into payload
66255. Base class containing output vars
66256. **comment:** Also mutex, for redundancy
 label: code-design
66257. Absolute level containing segments
66258. 0x1c
66259. Source database page to backup
66260. If this is the initial CREATE INDEX statement (or CREATE TABLE if the ** index is an implied index for a UNIQUE or PRIMARY KEY constraint) then ** emit code to allocate the index rootpage on disk and make an entry for ** the index in the sqlite_master table and populate the index with ** content. But, do not do this if we are simply reading the sqlite_master ** table to parse the schema, or if this index is the PRIMARY KEY index ** of a WITHOUT ROWID table. ** If pTblName==0 it means this index is generated as an implied PRIMARY KEY ** or UNIQUE index in a CREATE TABLE statement. Since the table ** has just been created, it contains no data and the index initialization ** step can be skipped.
66261. First of nPk registers holding PRIMARY KEY value
66262. ** Check the existence and status of a file.
66263. Size of aStat[] array
66264. True if a transaction is opened
66265. Index of first sample >= pRec
66266. The aiRowLogEst[] value for the sPk index
66267. The inode of fd
66268. True if any part of database file changed
66269. Object declarations
66270. CURTYPE_VTAB. Vtab cursor
66271. For looping over indices
66272. Expression node to test
66273. ** The second argument points to an FKey object representing a foreign key ** for which pTab is the child table. An UPDATE statement against pTab ** is currently being processed. For each column of the table that is ** actually updated, the corresponding element in the aChange[] array ** is zero or greater (if a column is unmodified the corresponding element ** is set to -1). If the rowid column is modified by the UPDATE statement ** the bChngRowid argument is non-zero. *** This function returns true if any of the columns that are part of the ** child key for FK constraint *p are modified.
66274. Start of code for Step 6
66275. Index of first record in sub-journal
66276. a pointer to a Token structure
66277. At one time, code such as "SELECT new.x" within a trigger would ** cause this condition to run. Since then, we have restructured how ** trigger code is generated and so this condition is no longer ** possible. However, it can still be true for statements like ** the following: *** CREATE TABLE t1(col INTEGER); ** SELECT (SELECT t1.col) FROM t1; ** ** when columnType() is called on the expression "t1.col" in the ** sub-select. In this case, set the column type to NULL, even ** though it should really be "INTEGER". *** This is not a problem, as the column type of "t1.col" is never ** used. When columnType() is called on the expression ** "(SELECT t1.col)", the correct type is returned (see the TK_SELECT *** branch below).
66278. INDEXED
66279. ***** Interface to code in fts5_expr.c.
66280. zDb
66281. Set content-flag (detail=none mode)
66282. "Managed" code needs this. Ticket #3382.
66283. **comment:** Silence some compiler warnings
 label: code-design
66284. Find modified rows
66285. Read the value of meta[3] from the database to determine where the ** root page of the new table should go. meta[3] is the largest root-page ** created so far, so the new root-page is (meta[3]+1).
66286. Parse a MATCH expression.
66287. !defined(SQLITE OMIT VIEW) && !defined(SQLITE OMIT_TRIGGER)
66288. Text of the new string
66289. 0 INVALID:
66290. Opcode: And P1 P2 P3 * * * Synopsis: r[P3]=(r[P1] && r[P2]) * * * Take the logical AND of the values in registers P1 and P2 and ** write the result into register P3. ** ** If either P1 or P2 is 0 (false) then the result is 0 even if ** the other input is NULL. A NULL and true or two NULLs give ** a NULL output.
66291. zNum is exactly 9223372036854775808. Fits if negative. The ** special case 2 overflow if positive
66292. ** Open a file.
66293. ** Sleep for nMicro microseconds. Return the number of microseconds ** actually slept.
66294. Ignore COLLATE or affinity on this tree
66295. Flags to sync db file with (or 0)
66296. If the approximation iKey is larger than the actual real search ** term, substitute >= for > and < for <=. e.g. if the search term ** is 4.9 and the integer approximation 5. ** ** (x > 4.9) -> (x >= 5) ** (x <= 4.9) -> (x < 5)
66297. Number of elements in the data array
66298. ** Cache data in the Fts3MultiSegReader.aBuffer[] buffer (overwriting any ** existing data). Grow the buffer if required. *** If successful, return SQLITE_OK. Otherwise, if an OOM error is encountered ** trying to resize the buffer, return SQLITE_NOMEM.
66299. Array of sub-lists
66300. 0 is not a legal page number and page 1 cannot be an ** overflow page. Therefore if ovflPgno<2 or past the end of the ** file the database must be corrupt.
66301. Old value of pPg->pgno, if sync is required
66302. ** Name of the default collating sequence
66303. ** Restore the cursor to the position it was in (or as close to as possible) ** when saveCursorPosition() was called. Note that this call deletes the ** saved position info stored by saveCursorPosition(), so there can be ** at most one effective restoreCursorPosition() call after each ** saveCursorPosition().
66304. Used to write the new, merged, segment
66305. SQLITE_ENABLE_MEMORY_MANAGEMENT

66306. ** Delete an IdList.
66307. Disable pWC->a[iParent] when this term disabled
66308. The complete hash table
66309. ** Check to see if the i-th bit is set. Return true or false. ** If p is NULL (if the bitmap has not been created) or if ** i is out of range, then return false.
66310. x=EXPR
66311. ** Set the pFile->lastErrno. Do this in a subroutine as that provides ** a convenient place to set a breakpoint.
66312. State information for the randomness gatherer.
66313. Page containing pPayload
66314. Bytes in string, or negative
66315. transtype ::= EXCLUSIVE
66316. If pIdx is NULL, then the parent key is the INTEGER PRIMARY KEY ** column of the parent table (table pTab).
66317. !defined(SQLITE_OMIT_DATETIME_FUNCS)
66318. Justification of ALWAYS(): A correct vtab constructor must allocate ** the sqlite3_vtab object if successful.
66319. TK_GE
66320. The index that might be used for coverage
66321. Allocate an extra register for limit+offset
66322. Jump here on any error
66323. ** Do a comparison between a 64-bit signed integer and a 64-bit floating-point ** number. Return negative, zero, or positive if the first (i64) is less than, ** equal to, or greater than the second (double).
66324. Extra space associated with each page
66325. Existing nearset, or NULL
66326. Update the state of the lock has held in the file descriptor then ** return the appropriate result code.
66327. SQLITE_OMIT_GET_TABLE
66328. ** By default, this module parses the legacy syntax that has been ** traditionally used by fts3. Or, if SQLITE_ENABLE_FTS3_PARENTHESIS ** is defined, then it uses the new syntax. The differences between ** the new and the old syntaxes are: *** a) The new syntax supports parenthesis. The old does not. *** b) The new syntax supports the AND and NOT operators. The old does not. *** c) The old syntax supports the "-" token qualifier. This is not ** supported by the new syntax (it is replaced by the NOT operator). *** d) When using the old syntax, the OR operator has a greater precedence ** than an implicit AND. When using the new, both implicit and explicit ** AND operators have a higher precedence than OR. *** If compiled with SQLITE_TEST defined, then this module exports the ** symbol "int sqlite3_fts3_enable_parentheses". Setting this variable ** to zero causes the module to use the old syntax. If it is set to ** non-zero the new syntax is activated. This is so both syntaxes can ** be tested using a single build of testfixture. *** The following describes the syntax supported by the fts3 MATCH ** operator in a similar format to that used by the lemon parser ** generator. This module does not use actually lemon, it uses a ** custom parser. *** query ::= andexpr (OR andexpr)*. *** andexpr ::= notexpr (AND? notexpr)*. *** notexpr ::= nearexpr (NOT nearexpr|TOKEN)*. *** notexpr ::= LP query RP. *** nearexpr ::= phrase (NEAR distance_opt nearexpr)*. *** distance_opt ::= . *** distance_opt ::= / INTEGER. *** phrase ::= TOKEN. *** phrase ::= COLUMN:TOKEN. ** phrase ::= "TOKEN TOKEN TOKEN...".
66329. **comment:** Opcode: IdxInsert P1 P2 P3 P4 P5 ** Synopsis: key=r[P2] *** Register P2 holds an SQL index key made using the ** MakeRecord instructions. This opcode writes that key ** into the index P1. Data for the entry is nil. *** If P4 is not zero, then it is the number of values in the unpacked ** key of reg(P2). In that case, P3 is the index of the first register ** for the unpacked key. The availability of the unpacked key can sometimes ** be an optimization. *** If P5 has the OPFLAG_APPEND bit set, that is a hint to the b-tree layer ** that this insert is likely to be an append. *** If P5 has the OPFLAG_NCHANGE bit set, then the change counter is ** incremented by this instruction. If the OPFLAG_NCHANGE bit is clear, ** then the change counter is unchanged. *** If the OPFLAG_USESEEKRESULT flag of P5 is set, the implementation might ** run faster by avoiding an unnecessary seek on cursor P1. However, ** the OPFLAG_USESEEKRESULT flag must only be set if there have been no prior ** seeks on the cursor or if the most recent seek used a key equivalent ** to P2. ** This instruction only works for indices. The equivalent instruction ** for tables is OP_Insert.
label: code-design
66330. 1st input operand
66331. Used to convert UTF16 parameters into UTF8 for display
66332. ** The MSVC CRT on Windows CE may not have a localtime() function. ** So declare a substitute. The substitute function itself is ** defined in "os_win.c".
66333. **comment:** ** Arguments alIdx, aDistance and aSpare all point to arrays of size ** nIdx. The alIdx array contains the set of integers from 0 to ** (nIdx-1) in no particular order. This function sorts the values ** in alIdx according to the indexed values in aDistance. For ** example, assuming the inputs: *** alIdx = { 0, 1, 2, 3 } ** aDistance = { 5.0, 2.0, 7.0, 6.0 } *** this function sets the alIdx array to contain: *** alIdx = { 0, 1, 2, 3 } *** The aSpare array is used as temporary working space by the ** sorting algorithm.
label: code-design
66334. aSample[] for the lower bound
66335. Total size of iBestCol in tokens
66336. ** Attempt to set a system-lock on the file pFile. The lock is ** described by pLock. *** If the pFile was opened read/write from unix-excl, then the only lock ** ever obtained is an exclusive lock, and it is obtained exactly once ** the first time any lock is attempted. All subsequent system locking ** operations become no-ops. Locking operations still happen internally, ** in order to coordinate access between separate database connections ** within this process, but all of that is handled in memory and the ** operating system does not participate. *** This function is a pass-through to fcntl(F_SETLK) if pFile is using ** any VFS other than "unix-excl" or if pFile is opened on "unix-excl" ** and is read-only. *** Zero is returned if the call completes successfully, or -1 if a call ** to fcntl() fails. In this case, errno is set appropriately (by fcntl()).
66337. Error condition
66338. **comment:** ** Add a new name/number pair to a VList. This might require that the ** VList object be reallocated, so return the new VList. If an OOM ** error occurs, the original VList returned and the ** db->mallocFailed flag is set. *** A VList is really just an array of integers. To destroy a VList, ** simply pass it to sqlite3DbFree(). *** The first integer is the number of integers allocated for the whole ** VList. The second integer is the number of integers actually used. ** Each name/number pair is encoded by subsequent groups of 3 or more ** integers. *** Each name/number pair starts with two integers which are the numeric ** value for the pair and the size of the name/number pair, respectively. ** The text name overlays one or more following integers. The text name ** is always zero-terminated. *** Conceptually: *** struct VList { ** int nAlloc; // Number of allocated slots ** int nUsed; // Number of used slots ** struct VListEntry { ** int iValue; // Value for this entry ** int nSlot; // Slots used by this entry ** ... variable name goes here ** } a[0]; ** } *** During code generation, pointers to the variable names within the ** VList are taken. When that happens, nAlloc is set to zero as an ** indication that the VList may never again be enlarged, since the ** accompanying realloc() would invalidate the pointers.
label: code-design
66339. sqlite3_free(pCur);
66340. ** Generate a single line of output for the tree, with a prefix that contains ** all the appropriate tree lines
66341. ** A specially optimized version of vdbeSorterCompare() that assumes that ** the first field of each key is a TEXT value and that the collation ** sequence to compare them with is BINARY.
66342. ** Interpret the given string as a boolean value.
66343. All foreign keys by referenced table name
66344. Consider tokens with this root node
66345. !defined(SQLITE_OMIT_CTE)
66346. ** If pNode currently points to a match, this function returns SQLITE_OK ** without modifying it. Otherwise, pNode is advanced until it does point ** to a match or EOF is reached.
66347. Thread procedure Win32 compatibility shim
66348. For a WITHOUT ROWID table, create an ephemeral table used to ** hold all primary keys for rows to be deleted.
66349. Open every index that needs updating.
66350. ** If the expression is always either TRUE or FALSE (respectively), ** then return 1. If one cannot determine the truth value of the ** expression at compile-time return 0. *** This is an optimization. If is OK to return 0 here even if ** the expression really is always false or false (a false negative). ** But it is a bug to return 1 if the expression might have different ** boolean values in different circumstances (a false positive.) *** Note that if the expression is part of conditional for a ** LEFT JOIN, then we cannot determine at compile-time whether or not ** is it true or false, so always return 0.
66351. The size of a Mem5Link object must be a power of two. Verify that ** this is case.

66352. ** Creates the lock file and any missing directories in lockPath
66353. True if attempting to delete from a view
66354. ** 2014 August 30 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil.
 ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
 ***** This file contains the public interface for the RBU extension.

66355. xDisconnect
66356. FROM subqueries have Table metadata
66357. ** Configure an sqlite3_wal_hook() callback to automatically checkpoint ** a database after committing a transaction if there are nFrame or ** more frames in the log file. Passing zero or a negative value as the ** nFrame parameter disables automatic checkpoints entirely. *** The callback registered by this function replaces any existing callback ** registered using sqlite3_wal_hook(). Likewise, registering a callback ** using sqlite3_wal_hook() disables the automatic checkpoint mechanism ** configured by this function.

66358. 4 -> ORDER
66359. Comparison information for merging rows
66360. Remove a single entry from the hash table given a pointer to that ** element and a hash on the element's key.
66361. Opcode: Close P1 * * * * * Close a cursor previously opened as P1. If P1 is not ** currently open, this instruction is a no-op.
66362. **comment:** Clear the temporary register cache, thereby ensuring that each ** co-routine has its own independent set of registers, because co-routines ** might expect their registers to be preserved across an OP_Yield, and ** that could cause problems if two or more co-routines are using the same ** temporary register.
label: code-design

66363. 450
66364. ** Implementation of the sqlite3_pcache.xInit method.
66365. The sqlite3BtreeCursor() routine can only fail for the first cursor ** opened for a database. Since there is already an open cursor when this ** opcode is run, the sqlite3BtreeCursor() cannot fail
66366. Nesting level
66367. ** Implementation of the QUOTE() function. This function takes a single ** argument. If the argument is numeric, the return value is the same as ** the argument. If the argument is NULL, the return value is the string ** "NULL". Otherwise, the argument is enclosed in single quotes with ** single-quote escapes.
66368. ** Free any buffer allocated by pBuf. Zero the structure before returning.
66369. If that was the last leaf node, break out of the loop
66370. File handle here
66371. adjust exponent for shifting decimal point
66372. ** 2008 May 26 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. **
 May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
 ***** This header file is used by programs that want to link against the ** ICU extension. All it does is declare the sqlite3IcuInit() interface.

66373. *****
66374. **comment:** ** This is the routine that actually formats the sqlite3_log() message. ** We house it in a separate routine from sqlite3_log() to avoid using ** stack space on small-stack systems when logging is disabled. *** sqlite3_log() must render into a static buffer. It cannot dynamically ** allocate memory because it might be called while the memory allocator ** mutex is held. *** sqlite3VXPrintf() might ask for *temporary* memory allocations for ** certain format characters (%q) or for very large precisions or widths. ** Care must be taken that any sqlite3_log() calls that occur while the ** memory mutex is held do not use these mechanisms.
label: code-design

66375. pUserData
66376. **comment:** Remove the xxx_node entry.
label: code-design

66377. ** Append the hash of the data type passed as the second argument to the ** hash-key value passed as the first. Return the new hash-key value.
66378. ** This routine is called by the parser while in the middle of ** parsing a CREATE TABLE statement. A "NOT NULL" constraint has ** been seen on a column. This routine sets the notNull flag on ** the column currently under construction.
66379. ** Streaming version of sqlite3session_changeshet().
66380. Hash table for fast lookup by key
66381. Number of header fields parsed so far
66382. EVIDENCE-OF: R-01849-26079 Value is a big-endian 32-bit ** twos-complement integer.
66383. ***** Include opcodes.h in the middle of vdbe.h *****
66384. cmd ::= DETACH database_kw_opt expr
66385. ** Discard any data stored in the in-memory hash tables. Do not write it ** to the database. Additionally, assume that the contents of the %_data ** table may have changed on disk. So any in-memory caches of %_data ** records must be invalidated.
66386. Keep running the loop until the Queue is empty
66387. For looping over tables in pSrc
66388. ** The char() function takes zero or more arguments, each of which is ** an integer. It constructs a string where each character of the string ** is the unicode character for the corresponding integer argument.
66389. Opcode: NullRow P1 * * * * * Move the cursor P1 to a null row. Any OP_Column operations ** that occur while the cursor is on the null row will always ** write a NULL.
66390. If pPayload is part of the data area of pPage, then make sure pPage ** is still writeable
66391. Mapping from Index fields to RHS fields
66392. Position list (no 0x00 term)
66393. The VDBE has been deallocated
66394. Sorter key to compare pVal with
66395. The parser to be shifted
66396. 1.0e+22 is the largest power of 10 than can be ** represented exactly.
66397. Local cache of pPage->aData
66398. whereLoopAddBtree() always generates and inserts the automatic index ** case first. Hence compatible candidate WhereLoops never have a larger ** rSetup. Call this SETUP-INVARIANT
66399. **comment:** In theory this code could just pass SQLITE_TRANSIENT as the final ** argument to sqlite3ValueSetStr() and have the copy created ** automatically. But doing so makes it difficult to detect any OOM ** error. Hence the code to create the copy externally.
label: code-design

66400. If this phrase is applies specifically to some column other than ** column iCol, return a NULL pointer.
66401. Byte offset within current leaf
66402. True to extend file if necessary
66403. Complete log message
66404. ** The proxy lock file path for the database at dbPath is written into lPath, ** which must point to valid, writable memory large enough for a maxLen length ** file path.
66405. nearest ::= phrase
66406. Opcode: ShiftLeft P1 P2 P3 * * * * Synopsis: r[P3]=r[P2]<<r[P1] ** ** Shift the integer value in register P2 to the left by the ** number of bits specified by the integer in register P1. ** Store the result in register P3. ** If either input is NULL, the result is NULL.
66407. cmd ::= ALTER TABLE add_column_fullname ADD kwcolumn_opt columnname carglist
66408. Page number to fetch
66409. ON CONFLICT algorithm.
66410. Convert the ORDER BY term into an integer column number iCol, ** taking care to preserve the COLLATE clause if it exists
66411. If auto-vacuum is disabled in this build and this is an auto-vacuum ** database, mark the database as read-only.
66412. Child nodes. For a NOT node, this array always contains 2 entries. For ** AND or OR nodes, it contains 2 or more entries.

66413. Number of entries in aMem
66414. OUT: Return code
66415. True to use OPFLAG_SEEKRESULT
66416. ** Check that the contents of the FTS index match that of the %_content ** table. Return SQLITE_OK if they do, or SQLITE_CORRUPT if not. Return ** some other SQLite error code if an error occurs while attempting to ** determine this.
66417. True to interpret filenames as URIs
66418. Size of aDoclist in bytes
66419. Cursor seek operation
66420. 248
66421. Pointer to current term
66422. ** Decrement the r-tree reference count. When the reference count reaches ** zero the structure is deleted.
66423. synopsis: r[P1@P3] <-> r[P2@P3]
66424. Start of loop body
66425. Argument to the thread
66426. Aggregate function
66427. Position of term for entry to add
66428. If a term is the BETWEEN operator, create two new virtual terms ** that define the range that the BETWEEN implements. For example: ** ** a BETWEEN b AND c ** ** is converted into: ** ** (a BETWEEN b AND c) AND (a>=b) AND (a<=c) ** ** The two new terms are added onto the end of the WhereClause object. ** The new terms are "dynamic" and are children of the original BETWEEN ** term. That means that if the BETWEEN term is coded, the children are ** skipped. Or, if the children are satisfied by an index, the original ** BETWEEN term is skipped.
66429. ** CAPI3REF: Setting The Subtype Of An SQL Function ** METHOD: sqlite3_context ** ** The sqlite3_result_subtype(C,T) function causes the subtype of ** the result from the [application-defined SQL function] with ** [sqlite3_context] C to be the value T. Only the lower 8 bits ** of the subtype T are preserved in current versions of SQLite; ** higher order bits are discarded. ** The number of subtype bytes preserved by SQLite might increase ** in future releases of SQLite.
66430. Move page iDbPage from its current location to page number iFreePage
66431. ** This function is used to merge two position lists into one. When it is ** called, *pp1 and *pp2 must both point to position lists. A position-list is ** the part of a doclist that follows each document id. For example, if a row ** contains: ** ** 'a b c'|x y z|'a b a' ** ** Then the position list for this row for token 'b' would consist of: ** ** 0x02 0x01 0x02 0x03 0x00 ** ** When this function returns, both *pp1 and *pp2 are left pointing to the ** byte following the 0x00 terminator of their respective position lists. ** ** If isSaveLeft is 0, an entry is added to the output position list for ** each position in *pp2 for which there exists one or more positions in ** *pp1 so that (pos(*pp2)>pos(*pp1) && pos(*pp2)-pos(*pp1)<=nToken). i.e. ** when the *pp1 token appears before the *pp2 token, but not more than nToken ** slots before it. ** ** e.g. nToken=1 searches for adjacent positions.
66432. Array of Fts3SegReader objects
66433. Generate auto-index WhereLoops
66434. Needed for schema access
66435. An instance of this object represents a JSON string ** under construction. Really, this is a generic string accumulator ** that can be and is used to create strings other than JSON.
66436. 39
66437. **comment:** ** This function is called after an incrmental-merge operation has run to ** merge (or partially merge) two or more segments from absolute level ** iAbsLevel. ** ** Each input segment is either removed from the db completely (if all of ** its data was copied to the output segment by the incrmerge operation) ** or modified in place so that it no longer contains those entries that ** have been duplicated in the output segment.
label: code-design
66438. Table change pertains to
66439. ** Advance the MergeEngine to its next entry. ** Set *pbEof to true there is no next entry because ** the MergeEngine has reached the end of all its inputs. ** ** Return SQLITE_OK if successful or an error code if an error occurs.
66440. New list element
66441. #include "windows.h"
66442. Integer key for tables. Size of pKey for indices
66443. ***** SQL functions used for testing and debugging*****
66444. If none of the above, then the result destination must be ** SRT_Output. This routine is never called with any other ** destination other than the ones handled above or SRT_Output. ** ** For SRT_Output, results are stored in a sequence of registers. ** Then the OP_ResultRow opcode is used to cause sqlite3_step() to ** return the next row of result.
66445. Token value. Zero terminated and dequoted
66446. CHECK constraint uses a changing column
66447. Determine the BM25 score for the current row.
66448. The name context of the SELECT statement
66449. ** Free all memory belonging to the PmaReader object passed as the ** argument. All structure fields are set to zero before returning.
66450. Finite column numbers
66451. Read the term data for the next term
66452. 160
66453. **comment:** ** This routine is called to handle SQL of the following forms: ** ** insert into TABLE (IDLIST) values(EXPRLIST),(EXPRLIST),... ** insert into TABLE (IDLIST) select ** insert into TABLE (IDLIST) default values ** ** The IDLIST following the table name is always optional. If omitted, ** then a list of all (non-hidden) columns for the table is substituted. ** The IDLIST appears in the pColumn parameter. pColumn is NULL if IDLIST ** is omitted. ** ** For the pSelect parameter holds the values to be inserted for the ** first two forms shown above. A VALUES clause is really just short-hand ** for a SELECT statement that omits the FROM clause and everything else ** that follows. If the pSelect parameter is NULL, that means that the ** DEFAULT VALUES form of the INSERT statement is intended. ** ** The code generated follows one of four templates. For a simple ** insert with data coming from a single-row VALUES clause, the code executes ** once straight down through. Pseudo-code follows (we call this ** the "1st template"): ** ** open write cursor to <table> and its indices ** put VALUES clause expressions into registers ** write the resulting record into <table> ** cleanup ** ** The three remaining templates assume the statement is of the form ** ** INSERT INTO <table> SELECT ... ** ** If the SELECT clause is of the restricted form "SELECT * FROM <table2>" - ** in other words if the SELECT pulls all columns from a single table ** and there is no WHERE or LIMIT or GROUP BY or ORDER BY clauses, and ** if <table2> and <table1> are distinct tables but have identical ** schemas, including all the same indices, then a special optimization ** is invoked that copies raw records from <table2> over to <table1>. ** See the xferOptimization() function for the implementation of this ** template. This is the 2nd template. ** ** open a write cursor to <table> ** open read cursor on <table2> ** transfer all records in <table2> over to <table> ** close cursors ** foreach index on <table> ** open a write cursor on the <table> index ** open a read cursor on the corresponding <table2> index ** transfer all records from the read to the write cursors ** close cursors ** end foreach ** ** The 3rd template is for when the second template does not apply ** and the SELECT clause does not read from <table> at any time. ** The generated code follows this template: ** ** X <- A ** goto B ** A: setup for the SELECT ** loop over the rows in the SELECT ** load values into registers R..R+n ** yield X ** end loop ** cleanup after the SELECT ** end-coroutine X ** B: open write cursor to <table> and its indices ** C: yield X, at EOF goto D ** insert the select result into <table> from R..R+n ** goto C ** D: cleanup ** ** The 4th template is used if the insert statement takes its ** values from a SELECT but the data is being inserted into a table ** that is also read as part of the SELECT. In the third form, ** we have to use an intermediate table to store the results of ** the select. The template is like this: ** ** X <- A ** goto B ** A: setup for the SELECT ** loop over the tables in the SELECT ** load value into register R..R+n ** yield X ** end loop ** cleanup after the SELECT ** end co-routine R ** B: open temp table ** L: yield X, at EOF goto M ** insert row from R..R+n into temp table ** goto L ** M: open write cursor to <table> and its indices ** rewind temp table ** C: loop over rows of intermediate table ** transfer values from intermediate table into <table> ** end loop ** D: cleanup
label: code-design
66454. Register to hold packed record
66455. Saved values of p->nLimit and p->nOffset
66456. Data comes from this temporary cursor if >=0
66457. ** An instance of the following structure can be declared on a stack and used ** to save the Parse.zAuthContext value so that it can be restored later.
66458. ** julianday(TIMESTRING, MOD, MOD, ...) ** ** Return the julian day number of the date specified in the arguments

66459. User authentication information
66460. Which column to return
66461. SELECT from sessionSelectStmt()
66462. ** Lower the locking level on file descriptor id to locktype. locktype ** must be either NO_LOCK or SHARED_LOCK. *** If the locking level of the file descriptor is already at or below ** the requested locking level, this routine is a no-op. *** It is not possible for this routine to fail if the second argument ** is NO_LOCK. If the second argument is SHARED_LOCK then this routine ** might return SQLITE_IOERR;
66463. Do not output
66464. True if table-valued-function syntax
66465. ** Discard the contents of the pending-terms hash tables.
66466. TO
66467. CASE
66468. 263
66469. Must be right after "base"
66470. **comment:** Warnings from sqlite3_log()
 label: code-design
66471. Name of db (e.g. "main")
66472. Unlink the session from the linked list of sessions attached to the ** database handle. Hold the db mutex while doing so.
66473. ** Close an iterator opened by sqlite3Fts5IndexQuery().
66474. If shared memory could not be created, then close the mutex and fail
66475. Size of pList in bytes
66476. True to skip deleted entries
66477. NOTE: Be sure to update mkopcodeh.tcl when adding or removing ** cases from this switch!
66478. Invoke BEFORE DELETE trigger programs.
66479. Hash score: 182
66480. Compiled version of zSql
66481. Language-id to search
66482. tab2 must be NOT NULL if tab1 is
66483. available free spot. check to see if this is going to
66484. Non-threadsafe build, use strerror().
66485. Remove any entries of the sqlite_sequence table associated with ** the table being dropped. This is done before the table is dropped ** at the btree level, in case the sqlite_sequence table needs to ** move as a result of the drop (can happen in auto-vacuum mode).
66486. Opcode: VBegin *** P4 * *** P4 may be a pointer to an sqlite3_vtab structure. If so, call the ** xBegin method for that table. *** Also, whether or not P4 is set, check that this is not being called from ** within a callback to a virtual table xSync() method. If it is, the error ** code will be set to SQLITE_LOCKED.
66487. Total time spent executing this instruction
66488. Prerequisites
66489. Name of %_segments table
66490. **comment:** The shared-lock has just been acquired then check to ** see if the database has been modified. If the database has changed, ** flush the cache. The hasHeldSharedLock flag prevents this from ** occurring on the very first access to a file, in order to save a ** single unnecessary sqlite3OsRead() call at the start-up. *** Database changes are detected by looking at 15 bytes beginning ** at offset 24 into the file. The first 4 of these 16 bytes are ** a 32-bit counter that is incremented with each change. The ** other bytes change randomly with each file change when ** a codec is in use. *** There is a vanishingly small chance that a change will not be ** detected. The chance of an undetected change is so small that ** it can be neglected.
 label: code-design
66491. Debug print SQL as it executes
66492. Number of SQL function arguments
66493. 4,5
66494. ** Write data to a blob handle.
66495. Index of candidate output segment
66496. Wal connection
66497. Invoke the authorization callback.
66498. ** Set the value stored in *pMem should already be a NULL. ** Also store a pointer to go with it.
66499. UTF-16 Little-endian -> UTF-8
66500. ** Evaluate an expression (either a vector or a scalar expression) and store ** the result in contiguous temporary registers. Return the index of ** the first register used to store the result. *** If the returned result register is a temporary scalar, then also write ** that register number into *piFreeable. If the returned result register ** is not a temporary or if the expression is a vector set *piFreeable ** to 0.
66501. ** Extract the next token from a tokenization cursor. The cursor must ** have been opened by a prior call to porterOpen().
66502. If no destructor action specified: do nothing
66503. LIMIT if wctrlFlags has WHERE_USE_LIMIT
66504. Ptr to head of pages backup list
66505. Cursor to iterate through node
66506. The VM that owns this context
66507. Size of allocation rounded up to power of 2
66508. ** This function interprets the string at (*pp) as a non-negative integer ** value. It reads the integer and sets *pnOut to the value read, then ** sets *pp to point to the byte immediately following the last byte of ** the integer value. *** Only decimal digits ('0'..'9') may be part of an integer value. *** If *pp does not point to a decimal digit SQLITE_ERROR is returned and ** the output value undefined. Otherwise SQLITE_OK is returned. *** This function is used when parsing the "prefix=" FTS4 parameter.
66509. Initialize the contents of sLocal to avoid a compiler warning.
66510. Change in the number of documents
66511. Argument
66512. The first element of an INTARRAY is always the ** count of the number of elements to follow
66513. True if key size is 2^32 or more
66514. ** Set the i-th bit. Return 0 on success and an error code if ** anything goes wrong. *** This routine might cause sub-bitmaps to be allocated. Failing ** to get the memory needed to hold the sub-bitmap is the only ** that can go wrong with an insert, assuming p and i are valid. *** The calling function must ensure that p is a valid Bitvec object ** and that the value for "i" is within range of the Bitvec object. ** Otherwise the behavior is undefined.
66515. ** A single step of an incremental checkpoint - frame iWalFrame of the wal ** file should be copied to page iDbPage of the database file.
66516. Keyword text
66517. OUT: Pointer to table name
66518. An OOM - finalize() below returns S_NOMEM
66519. If the query is DISTINCT with an ORDER BY but is not an aggregate, and ** if the select-list is the same as the ORDER BY list, then this query ** can be rewritten as a GROUP BY. In other words, this: *** SELECT DISTINCT xyz FROM ... ORDER BY xyz ** ** is transformed to: *** SELECT xyz FROM ... GROUP BY xyz ORDER BY xyz ** ** The second form is preferred as a single index (or temp-table) may be ** used for both the ORDER BY and DISTINCT processing. As originally ** written the query must use a temp-table for at least one of the ORDER ** BY and DISTINCT, and an index or separate temp-table for the other.
66520. Information about each nest loop in WHERE
66521. If z is a NULL pointer, set pMem to contain an SQL NULL.
66522. Size of aNode in bytes
66523. ** This variant of sqlite3BtreePayload() works even if the cursor has not ** in the CURSOR_VALID state. It is only used by the sqlite3_blob_read() ** interface.
66524. On the upwards pass, or...
66525. Check that this node should not be FTS5_TERM

66526. Estimated size of value in this column. sizeof(INT)==1
66527. True for a "natural" join
66528. If there are DELETE triggers on this table and the ** recursive-triggers flag is set, call GenerateRowDelete() to ** remove the conflicting row from the table. This will fire ** the triggers and remove both the table and index b-tree entries. *** Otherwise, if there are no triggers or the recursive-triggers ** flag is not set, but the table has one or more indexes, call ** GenerateRowIndexDelete(). This removes the index b-tree entries ** only. The table b-tree entry will be replaced by the new entry ** when it is inserted. *** If either GenerateRowDelete() or GenerateRowIndexDelete() is called, ** also invoke MultiWrite() to indicate that this VDBE may require ** statement rollback (if the statement is aborted after the delete ** takes place). Earlier versions called sqlite3MultiWrite() regardless, ** but being more selective here allows statements like: *** REPLACE INTO t(rowid) VALUES(\$newrowid) *** to run without a statement journal if there are no indexes on the ** table.
66529. ** Define HAVE_FULLFSYNC to 0 or 1 depending on whether or not ** the F_FULLFSYNC macro is defined. F_FULLFSYNC is currently ** only available on Mac OS X. But that could change.
66530. The MATCH or rowid constraint, if any
66531. Index structure
66532. Read the total number of levels and segments from the start of the ** structure record.
66533. ** Possible values for the sqlite.magic field. ** The numbers are obtained at random and have no special meaning, other ** than being distinct from one another.
66534. xNext
66535. This loop determines (a) if the commit hook should be invoked and ** (b) how many database files have open write transactions, not ** including the temp database. (b) is important because if more than ** one database file has an open write transaction, a master journal ** file is required for an atomic commit.
66536. stmt_cursor is a subclass of sqlite3_vtab_cursor which will ** serve as the underlying representation of a cursor that scans ** over rows of the result
66537. The minor type of the error token
66538. Destructor for the data
66539. Database is open
66540. IN/OUT: Pointer to PendingList struct
66541. If the column has REAL affinity, it may currently be stored as an ** integer. Use OP_RealAffinity to make sure it is really real. *** EVIDENCE-OF: R-60985-57662 SQLite will convert the value back to ** floating point when extracting it from the record.
66542. ** Are most of the Win32 Unicode APIs available (i.e. with certain exceptions ** based on the sub-platform)?
66543. Report an out-of-memory (OOM) condition
66544. Make sure initialized even if FullPathname() fails
66545. trigger_event ::= UPDATE
66546. A
66547. If the entry in aPgno[] is already set, then the previous writer ** must have exited unexpectedly in the middle of a transaction (after ** writing one or more dirty pages to the WAL to free up memory). ** Remove the remnants of that writers uncommitted transaction from ** the hash-table before writing any new entries.
66548. Assert that the mutex (if any) associated with the BtShared database ** that contains table p is held by the caller. See header comments ** above function sqlite3VtabUnlockList() for an explanation of why ** this makes it safe to access the sqlite3.pDisconnect list of any ** database connection that may have an entry in the p->pVTable list.
66549. amalgamator: keep
66550. ** 2008 October 07 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the C functions that implement mutexes. ***
This implementation in this file does not provide any mutual ** exclusion and is thus suitable for use only in applications ** that use SQLite in a single thread. The routines defined ** here are place-holders. Applications can substitute working ** mutex routines at start-time using the ***
sqlite3_config(SQLITE_CONFIG_MUTEX,...) *** interface. *** If compiled with SQLITE_DEBUG, then additional logic is inserted ** that does error checking on mutexes to make sure they are being ** called correctly.
66551. Opcode: Compare P1 P2 P3 P4 P5 ** Synopsis: r[P1@P3] <-> r[P2@P3] ** ** Compare two vectors of registers in reg(P1)..reg(P1+P3-1) (call this ** vector "A") and in reg(P2)..reg(P2+P3-1) ("B"). Save the result of ** the comparison for use by the next OP_Jump instruct. *** If P5 has the OPFLAG_PERMUTE bit set, then the order of comparison is ** determined by the most recent OP_Permutation operator. If the ** OPFLAG_PERMUTE bit is clear, then register are compared in sequential ** order. *** P4 is a KeyInfo structure that defines collating sequences and sort ** orders for the comparison. The permutation applies to registers ** only. The KeyInfo elements are used sequentially. *** The comparison is a sort comparison, so NULLs compare equal, ** NULLs are less than numbers, numbers are less than strings, ** and strings are less than blobs.
66552. Page size according to the log
66553. ** Lock the file with the lock specified by parameter locktype - one ** of the following: *** (1) SHARED_LOCK ** (2) RESERVED_LOCK ** (3)
PENDING_LOCK ** (4) EXCLUSIVE_LOCK *** Sometimes when requesting one lock state, additional lock states ** are inserted in between. The locking might fail on one of the later ** transitions leaving the lock state different from what it started but ** still short of its goal. The following chart shows the allowed ** transitions and the inserted intermediate states: *** UNLOCKED -> SHARED ** SHARED -> RESERVED ** SHARED -> (PENDING) -> EXCLUSIVE
** RESERVED -> (PENDING) -> EXCLUSIVE ** PENDING -> EXCLUSIVE *** This routine will only increase a lock. The winUnlock() routine ** erases all locks at once and returns us immediately to locking level 0. ** It is not possible to lower the locking level one step at a time. You ** must go straight to locking level 0.
66554. OUT: Pointer to position list
66555. Make sure a mutex is held on the table to be accessed
66556. setlist
66557. Check if there exists a MATCH constraint - even an unusable one. If there ** is, do not consider the lookup-by-rowid plan as using such a plan would ** require the VDBE to evaluate the MATCH constraint, which is not currently ** possible.
66558. Check to see if the new rowid already exists in the table. Skip ** the following conflict logic if it does not.
66559. Remove the shared lock before trying the range. we'll need to ** reestablish the shared lock if we can't get the afpUnlock
66560. If only SQLITE_ENABLE_STAT3_OR_STAT4 is on, then the largest input ** possible to this routine is 310, resulting in a maximum x of 31
66561. 232
66562. ** The second argument, argv[], is an array of pointers to nul-terminated ** strings. This function makes a copy of the array and strings into a ** single block of memory. It then dequotes any of the strings that appear ** to be quoted. *** If successful, output parameter *pazDequote is set to point at the ** array of dequoted strings and SQLITE_OK is returned. The caller is ** responsible for eventually calling sqlite3_free() to free the array ** in this case. Or, if an error occurs, an SQLite error code is returned. ** The final value of *pazDequote is undefined in this case.
66563. ATTACH
66564. Temp space for PoslistNearMerge()
66565. True if the GROUP BY and ORDER BY are the same
66566. ** Allocate VdbeCursor number iCur. Return a pointer to it. Return NULL ** if we run out of memory.
66567. Conflict algorithm. (OE_Abort, etc)
66568. (272) input ::= cmdlist
66569. Generate code for an unconditional jump to instruction iDest
66570. CSE1 from below
66571. Set the threshold at which OP_Once counters reset back to zero. ** By default this is 0x7fffffff (over 2 billion), but that value is ** too big to test in a reasonable amount of time, so this control is ** provided to set a small and easily reachable reset value.
66572. Root page number of the table to be locked
66573. a short read or version format mismatch means we need to create a new ** conch file.
66574. ** Try to read the wal-index header. Return 0 on success and 1 if ** there is a problem. *** The wal-index is in shared memory. Another thread or process might ** be writing the header at the same time this procedure is trying to ** read it, which might result in inconsistency. A dirty read is detected ** by verifying that both copies of the header are the same and also by ** a checksum on the header. *** If and only if the read is consistent and the header is different from ** pWal->hdr, then pWal->hdr is updated to the content of the new header ** and *pChanged is set to 1. *** If the checksum cannot be verified return non-zero. If the header ** is read successfully and the checksum verified, return zero.
66575. Calculate the phrase frequency (symbol "f(qi,D)" in the documentation) ** for each phrase in the query for the current row.

66576. First byte of the Cell
66577. ** Delete all entries in the %_segments table associated with the segment ** opened with seg-reader pSeg. This function does not affect the contents ** of the %_segdir table.
66578. ** Assuming mem5.zPool is divided up into an array of Mem5Link ** structures, return a pointer to the idx-th such link.
66579. ** For a compound SELECT statement, make sure p->pPrior->pNext==p for ** all elements in the list. And make sure list length does not exceed **
SQLITE_LIMIT_COMPOUND_SELECT.
66580. Initialize the output buffer
66581. FROM clause pLvl reads data from
66582. For looping over existing file IDs
66583. Disable the pCache->nPage validity check
66584. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Main file for the SQLite library. The routines in this file **
implement the programmer interface to the library. Routines in ** other files are for internal use by SQLite and should not be ** accessed by users of the library.
66585. ** Rtree virtual table module xBestIndex method. There are three ** table scan strategies to choose from (in order from most to ** least desirable): *** *** idxNum
idxStr Strategy ** ----- ** 1 Unused Direct lookup by rowid. ** 2 See below R-tree query or full-table scan. ** -----
----- ** If strategy 1 is used, then idxStr is not meaningful. If strategy ** 2 is used, idxStr is formatted to contain 2 bytes for each **
constraint used. The first two bytes of idxStr correspond to ** the constraint in sqlite3_index_info.aConstraintUsage[] with ** (argvIndex==1) etc. *** *** The first
of each pair of bytes in idxStr identifies the constraint ** operator as follows: *** *** Operator Byte Value ** ----- ** = 0x41 ('A') ** <= 0x42 ('B') **
< 0x43 ('C') ** >= 0x44 ('D') ** > 0x45 ('E') ** MATCH 0x46 ('F') ** ----- ** The second of each pair of bytes identifies the coordinate column
** to which the constraint applies. The leftmost coordinate column ** is 'a', the second from the left 'b' etc.
66586. Operator(s) to scan for
66587. !defined(SQLITE_HWTIME_H)
66588. Unsigned integer types. These are already defined in the sqliteInt.h, ** but the definitions need to be repeated for separate compilation.
66589. Many of the flag-pragmas modify the code generated by the SQL ** compiler (eg. count_changes). So add an opcode to expire all ** compiled SQL statements
after modifying a pragma value.
66590. ***** End of wal.c *****
66591. Length of z in bytes
66592. Segment id
66593. Number of entries in argv[]
66594. non-ASCII codepoint read from input
66595. ** Constructor for a new stmt_cursor object.
66596. ** Locate or create an AutoincInfo structure associated with table pTab ** which is in database iDb. Return the register number for the register ** that holds the
maximum rowid. Return zero if pTab is not an AUTOINCREMENT ** table. (Also return zero when doing a VACUUM since we do not want to ** update the
AUTOINCREMENT counters during a VACUUM.) *** There is at most one AutoincInfo structure per table even if the ** same table is autoincremented
multiple times due to inserts within ** triggers. A new AutoincInfo structure is created if this is the ** first use of table pTab. On 2nd and subsequent uses, the
original ** AutoincInfo structure is used. *** Three memory locations are allocated: *** *** (1) Register to hold the name of the pTab table. ** (2) Register to
hold the maximum ROWID of pTab. ** (3) Register to hold the rowid in sqlite_sequence of pTab *** *** The 2nd register is the one that is returned. That is all the
*** insert routine needs to know about.
66597. ** Destroy a tokenizer allocated by unicodeCreate().
66598. Releasing a reserved lock
66599. ***** End of resolve.c *****
66600. ** Advance the cursor to the next row in the table.
66601. The top of the parser's stack
66602. Make sure the allocated memory does not assume that it is set to zero ** or retains a value from a previous allocation
66603. 197
66604. defer MT decisions to system malloc
66605. Appended to wal file
66606. Size of list.aMemory allocation in bytes
66607. If the cursor is not at EOF, read the next character
66608. sqlite3_test_control(SQLITE_TESTCTRL_IMPOSTER, db, dbName, onOff, tnum); *** This test control is used to create imposter tables. "db" is a pointer **
to the database connection. dbName is the database name (ex: "main" or ** "temp") which will receive the imposter. "onOff" turns imposter mode on ** or off.
"tnum" is the root page of the b-tree to which the imposter ** table should connect. *** Enable imposter mode only when the schema has already been parsed.
Then ** run a single CREATE TABLE statement to construct the imposter table in ** the parsed schema. Then turn imposter mode back off again. *** If
onOff==0 and tnum>0 then reset the schema for all databases, causing ** the schema to be reparsed the next time it is needed. This has the ** effect of erasing all
imposter tables.
66609. Index within level of segment to modify
66610. State variables used for validating that the transaction control ** methods of the virtual table are called at appropriate times. These ** values do not contribute to
FTS functionality; they are used for ** verifying the operation of the SQLite core.
66611. Required lock type (READ_LOCK or WRITE_LOCK)
66612. Foreign key for which pTab is the child
66613. SQLITE_FORCE_PROXY_LOCKING==1 means force always use proxy, 0 means ** never use proxy, NULL means use proxy for non-local files only.
66614. **comment:** ** Generate code to construct the Index object for an automatic index ** and to set up the WhereLevel object pLevel so that the code generator **
makes use of the automatic index.
label: code-design
66615. For the purposes of this routine, disable the mutex
66616. See if we are dealing with a keyword.
66617. True if vacuuming a :memory: database
66618. ** Compare the key in memory cell pVal with the key that the sorter cursor ** passed as the first argument currently points to. For the purposes of ** the
comparison, ignore the rowid field at the end of each record. *** If the sorter cursor key contains any NULL values, consider it to be ** less than pVal. Even if
pVal also contains NULL values. *** If an error occurs, return an SQLite error code (i.e. SQLITE_NOMEM). ** Otherwise, set *pRes to a negative, zero or
positive value if the ** key in pVal is smaller than, equal to or larger than the current sorter ** key. *** This routine forms the core of the OP_SorterCompare
opcode, which in ** turn is used to verify uniqueness when constructing a UNIQUE INDEX.
66619. Create the statement to insert index entries
66620. Index of term=? value in apVal
66621. Max rowid register
66622. xRollback - rollback transaction
66623. Non-negative integer value if EP_IntValue
66624. Comparison collation sequence
66625. OUT: Rowid value
66626. Commit by deleting journal file
66627. ** The database opened by the first argument is an auto-vacuum database ** nOrig pages in size containing nFree free pages. Return the expected ** size of the
database in pages following an auto-vacuum operation.
66628. ** Find the appropriate action for a parser given the non-terminal ** look-ahead token iLookAhead.
66629. Vector to extract element from
66630. Cursor number for the sorter
66631. Source-file line number of first opcode
66632. ** This routine is called to do the work of a DROP TABLE statement. ** pName is the name of the table to be dropped.

66633. Mask of bits to set or clear.
66634. Memory mapped region
66635. Sorting and partial sorting costs
66636. ** Return the name of a directory in which to put temporary files. ** If no suitable temporary file directory can be found, return NULL.
66637. Page size can no longer be changed
66638. The location to move pDbPage to
66639. ** Here ends the implementation of all sqlite3_file methods. ** **** End sqlite3_file Methods ****

66640. Used to iterate through p->apValue[]
66641. 230
66642. Must be element [2]
66643. UNCOMPRESS
66644. Mask of columns used by this cursor
66645. The SQL to be executed
66646. If pIdx is an index on one or more expressions, then look through ** all the expressions in pWInfo and try to transform matching expressions ** into reference to index columns.
66647. If this is a rollback of a savepoint and data was not written to ** the database and the page is not in-memory, there is a potential ** problem. When the page is next fetched by the b-tree layer, it ** will be read from the database file, which may or may not be ** current. ** ** There are a couple of different ways this can happen. All are quite ** obscure. When running in synchronous mode, this can only happen ** if the page is on the free-list at the start of the transaction, then ** populated, then moved using sqlite3PagerMovepage(). ** ** The solution is to add an in-memory page to the cache containing ** the data just read from the sub-journal. Mark the page as dirty ** and if the pager requires a journal-sync, then mark the page as ** requiring a journal-sync before it is written.
66648. The cursor pointing to the table
66649. **comment:** If the FULLFSYNC failed, fall back to attempting an fsync(). ** It shouldn't be possible for fullfsync to fail on the local ** file system (on OSX), so failure indicates that FULLFSYNC ** isn't supported for this file system. So, attempt an fsync ** and (for now) ignore the overhead of a superfluous fcntl call. ** It'd be better to detect fullfsync support once and avoid ** the fcntl call every time sync is called.
label: code-design
66650. The VDBE cursor for the index
66651. If SQLite is already completely initialized, then this call ** to sqlite3_initialize() should be a no-op. But the initialization ** must be complete. So isInit must not be set until the very end ** of this routine.
66652. Allocate the TriggerPrg and SubProgram objects. To ensure that they ** are freed if an error occurs, link them into the Parse.pTriggerPrg ** list of the top-level Parse object sooner rather than later.
66653. ** Buffer object for the incremental building of string data.
66654. ** 2006 June 7 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used to dynamically load extensions into ** the SQLite library.
66655. !. Part of !=
66656. ** Register a callback to be invoked each time a transaction is rolled ** back by this database connection.
66657. ** Append N copies of character c to the given string buffer.
66658. ** Transfer eligible terms from the HAVING clause of a query, which is ** processed after grouping, to the WHERE clause, which is processed before ** grouping. For example, the query: ** ** SELECT * FROM <tables> WHERE a=? GROUP BY b HAVING b=? AND c=? ** ** can be rewritten as: ** ** SELECT * FROM <tables> WHERE a=? AND b=? GROUP BY b HAVING c=? ** ** A term of the HAVING expression is eligible for transfer if it consists ** entirely of constants and expressions that are also GROUP BY terms that ** use the "BINARY" collation sequence.
66659. If the bRetry flag is set, the change has not been applied due to an ** SQLITE_CHANGESET_DATA problem (i.e. this is an UPDATE or DELETE and ** a row with the correct PK is present in the db, but one or more other ** fields do not contain the expected values) and the conflict handler ** returned SQLITE_CHANGESET_REPLACE. In this case retry the operation, ** but pass NULL as the final argument so that sessionApplyOneOp() ignores ** the SQLITE_CHANGESET_DATA problem.
66660. ** Flush any data currently held in-memory to disk.
66661. ** Set the safety_level and pager flags for pager iDb. Or if iDb<0 ** set these values for all pagers.
66662. An FTS5_CONTENT value
66663. 15-18
66664. ** Find the largest relative level number in the table. If successful, set ** *pnMax to this value and return SQLITE_OK. Otherwise, if an error occurs, ** set *pnMax to zero and return an SQLite error code.
66665. OUT: Malloc'd doclist
66666. 970
66667. Number of characters in zCharSet
66668. case_operand
66669. ** This function is used to read or overwrite payload information ** for the entry that the pCur cursor is pointing to. The eOp ** argument is interpreted as follows: ** ** 0: The operation is a read. Populate the overflow cache. ** 1: The operation is a write. Populate the overflow cache. ** ** A total of "amt" bytes are read or written beginning at "offset". ** Data is read to or from the buffer pBuf. ** ** The content being read or written might appear on the main page ** or be scattered out on multiple overflow pages. ** ** If the current cursor entry uses one or more overflow pages ** this function may allocate space for and lazily populate ** the overflow page-list cache array (BtCursor.aOverflow). ** Subsequent calls use this cache to make seeking to the supplied offset ** more efficient. ** ** Once an overflow page-list cache has been allocated, it must be ** invalidated if some other cursor writes to the same table, or if ** the cursor is moved to a different row. Additionally, in auto-vacuum ** mode, the following events may invalidate an overflow page-list cache. ** ** * An incremental vacuum, ** * A commit in auto_vacuum="full" mode, ** * Creating a table (may require moving an overflow page).
66670. Increase the size of the aTo set by one
66671. ** Initialize the first page of the database file (creating a database ** consisting of a single page and no schema objects). Return SQLITE_OK ** if successful, or an SQLite error code otherwise.
66672. ** Invoke the xCommit method of all virtual tables in the ** sqlite3.aVTrans array. Then clear the array itself.
66673. ** Begin a read transaction on the WAL. ** ** This routine used to be called "pagerOpenSnapshot()" because it essentially ** makes a snapshot of the database at the current point in time and preserves ** that snapshot for use by the reader in spite of concurrently changes by ** other writers or check pointers.
66674. ** This routine checks if there is a RESERVED lock held on the specified ** file by this or any other process. If such a lock is held, set *pResOut ** to a non-zero value otherwise *pResOut is set to zero. The return value ** is set to SQLITE_OK unless an I/O error occurs during lock checking. ** ** In dotfile locking, either a lock exists or it does not. So in this ** variation of CheckReservedLock(), *pResOut is set to true if any lock ** is held on the file and false if the file is unlocked.
66675. ** Possible error returns from patternMatch()
66676. \$ => nothing
66677. BEGIN => ID
66678. Quasi-random value added to every checksum
66679. Internal-use-only converters allowed
66680. 222
66681. P5 value for OP_Column + FLAGS
66682. pScratch
66683. Register for RowSet object
66684. SQLITE_OS_WIN
66685. Previous rowid written to current leaf
66686. same as TK_SLASH, synopsis: r[P3]=r[P2]/r[P1]
66687. SELECT statement from sessionSelectRow()

66688. Bitmask of all WhereLoop objects in this path
66689. If the allocation succeeded, populate the new object.
66690. Level to merge
66691. SQLITE_OMIT_INTEGRITY_CHECK
66692. Record changes in the file format
66693. sqlite3VdbeSetColName() installs column names as UTF8 ** strings so there is no way for sqlite3_column_name() to fail.
66694. Mem.z points to a static string
66695. Fill in the azColumn array
66696. Number of columns in the data
66697. 68
66698. ** This routine is used to allocate sufficient space for an UnpackedRecord ** structure large enough to be used with sqlite3VdbeRecordUnpack() if ** the first argument is a pointer to KeyInfo structure pKeyInfo. *** The space is either allocated using sqlite3DbMallocRaw() or from within ** the unaligned buffer passed via the second and third arguments (presumably ** stack space). If the former, then *ppFree is set to a pointer that should ** be eventually freed by the caller using sqlite3DbFree(). Or, if the ** allocation comes from the pSpace/szSpace buffer, *ppFree is set to NULL ** before returning. *** If an OOM error occurs, NULL is returned.
66699. 99
66700. Verify that the cursor and the BtShared agree about what is the current ** database connection. This is important in shared-cache mode. If the database ** connection pointers get out-of-sync, it is possible for routines like ** btreeInitPage() to reference an stale connection pointer that references a ** a connection that has already closed. This routine is used inside assert() ** statements only and for the purpose of double-checking that the btree code ** does keep the database connection pointers up-to-date.
66701. ** Cause any pending operation to stop at its earliest opportunity.
66702. Current merge state (see above)
66703. Pointer to hash table
66704. 5
66705. Offset of this term from read positions
66706. Matched CTE (or NULL if no match)
66707. ** The idea is that this function works like a combination of ** GetLastError() and FormatMessage() on Windows (or errno and ** strerror_r() on Unix). After an error is returned by an OS ** function, SQLite calls this function with zBuf pointing to ** a buffer of nBuf bytes. The OS layer should populate the ** buffer with a nul-terminated UTF-8 encoded error message ** describing the last IO error to have occurred within the calling ** thread. *** If the error message is too large for the supplied buffer, ** it should be truncated. The return value of xGetLastError ** is zero if the error message fits in the buffer, or non-zero ** otherwise (if the message was truncated). If non-zero is returned, ** then it is not necessary to include the nul-terminator character ** in the output buffer. *** Not supplying an error message will have no adverse effect ** on SQLite. It is fine to have an implementation that never ** returns an error message: *** int xGetLastError(sqlite3_vfs *pVfs, int nBuf, char *zBuf){ ** assert(zBuf[0]=='0'); ** return 0; ** } *** However if an error message is supplied, it will be incorporated ** by sqlite into the error message available to the user using ** sqlite3_errmsg(), possibly making IO errors easier to debug.
66708. Virtual tables with open transactions
66709. ***** Begin file ctime.c *****
66710. ** Set the pointer-map entries for all children of page pPage. Also, if ** pPage contains cells that point to overflow pages, set the pointer ** map entries for the overflow pages as well.
66711. ** Sync the file. *** If the real file has been created, call its xSync method. Otherwise, ** syncing an in-memory journal is a no-op.
66712. ** Defer sourcing vdbe.h and btree.h until after the "u8" and ** "BusyHandler" typedefs. vdbe.h also requires a few of the opaque ** pointer types (i.e. FuncDef) defined above.
66713. The real implementation of xFetch and xUnfetch
66714. Get the column number in the table (iColumn) and sort order ** (revIdx) for the j-th column of the index.
66715. there was a collision, check to see if it's already
66716. ** This routine is called on a collation sequence before it is used to ** check that it is defined. An undefined collation sequence exists when ** a database is loaded that contains references to collation sequences ** that have not been defined by sqlite3_create_collation() etc. *** If required, this routine calls the 'collation needed' callback to ** request a definition of the collating sequence. If this doesn't work, ** an equivalent collating sequence that uses a text encoding different ** from the main database is substituted, if one is available.
66717. **comment:** An error occurred. Delete the contents of the apLeaf[] array ** and pFree list. Everything else is cleaned up by the call to **
sqlite3Fts3ExprFree(pRoot) below.
label: code-design
66718. **comment:** ** CAPI3REF: Tracing And Profiling Functions ** METHOD: sqlite3 *** These routines are deprecated. Use the [sqlite3_trace_v2()] interface ** instead of the routines described here. *** These routines register callback functions that can be used for ** tracing and profiling the execution of SQL statements. ** ^The callback function registered by sqlite3_trace() is invoked at ** various times when an SQL statement is being run by [sqlite3_step()]. ** ^The sqlite3_trace() callback is invoked with a UTF-8 rendering of the ** SQL statement text as the statement first begins executing. ** ^(Additional sqlite3_trace() callbacks might occur ** as each triggered subprogram is entered. The callbacks for triggers ** contain a UTF-8 SQL comment that identifies the trigger.)^ ** ^The [SQLITE_TRACE_SIZE_LIMIT] compile-time option can be used to limit ** the length of [bound parameter] expansion in the output of sqlite3_trace(). ** ^The callback function registered by sqlite3_profile() is invoked ** as each SQL statement finishes. ^The profile callback contains ** the original statement text and an estimate of wall-clock time ** of how long that statement took to run. ^The profile callback ** time is in units of nanoseconds, however the current implementation ** is only capable of millisecond resolution so the six least significant ** digits in the time are meaningless. Future versions of SQLite ** might provide greater resolution on the profiler callback. The ** sqlite3_profile() function is considered experimental and is ** subject to change in future versions of SQLite.
label: code-design
66719. Number of def. constraints when stmt started
66720. Max page number of 2147483648
66721. jump: P2 holds jmp target
66722. The integer values
66723. Write output here
66724. Depth of the tree so far
66725. 243
66726. ROWID of most recent insert (see above)
66727. ** Based on the contents of the AggInfo structure indicated by the first ** argument, this function checks if the following are true: *** * the query contains just a single aggregate function, ** * the aggregate function is either min() or max(), and *** the argument to the aggregate function is a column value. *** If all of the above are true, then WHERE_ORDERBY_MIN or WHERE_ORDERBY_MAX ** is returned as appropriate. Also, *ppMinMax is set to point to the ** list of arguments passed to the aggregate before returning. *** Or, if the conditions above are not met, *ppMinMax is set to 0 and ** WHERE_ORDERBY_NORMAL is returned.
66728. ** Merge the doclist aDoclist/nDoclist into the TermSelect object passed ** as the first argument. The merge is an "OR" merge (see function ** fts3DoclistOrMerge() for details). *** This function is called with the doclist for each term that matches ** a queried prefix. It merges all these doclists into one, the doclist ** for the specified prefix. Since there can be a very large number of ** doclists to merge, the merging is done pair-wise using the TermSelect ** object. *** This function returns SQLITE_OK if the merge is successful, or an ** SQLite error code (SQLITE_NOMEM) if an error occurs.
66729. star_opt ::= STAR
66730. The value list on the RHS of "x IN (v1,v2,v3,...)"
66731. ***** Include rtree.h in the middle of main.c *****
66732. ** Compare two UTF-8 strings for equality where the first string is ** a "LIKE" expression. Return true (1) if they are the same and ** false (0) if they are different.
66733. Test if comparison i of pTerm is compatible with column (i+nEq) ** of the index. If not, exit the loop.
66734. ** This function is called when ** the transaction opened by database db has just finished. Locks held ** by database connection db have been released. *** This function loops through each entry in the blocked connections ** list and does the following: *** 1) If the sqlite3_pBlockingConnection member of a list

entry is ** set to db, then set pBlockingConnection=0. *** 2) If the sqlite3.pUnlockConnection member of a list entry is ** set to db, then invoke the configured unlock-notify callback and ** set pUnlockConnection=0. *** 3) If the two steps above mean that pBlockingConnection==0 and ** pUnlockConnection==0, remove the entry from the blocked connections ** list.

66735. 123456789 123456

66736. PRAGMA => ID

66737. ** Set output variable *ppStmt to point to an UPDATE statement that may ** be used to update the imposter table for the main table b-tree of the ** table object that pIter currently points to, assuming that the ** rbu_control column of the data_xyz table contains zMask. *** If the zMask string does not specify any columns to update, then this ** is not an error. Output variable *ppStmt is set to NULL in this case.

66738. size of the BLOB

66739. synopsis: r[P2]=cursor[P1].ctr++

66740. Hash table of connection functions

66741. ** The common case is for a varint to be a single byte. They following ** macros handle the common case without a procedure call, but then call ** the procedure for larger varints.

66742. ** All current savepoints are stored in a linked list starting at ** sqlite3.pSavepoint. The first element in the list is the most recently ** opened savepoint. Savepoints are added to the list by the vdbe ** OP_Savepoint instruction.

66743. Rows less than the upper bound

66744. Output the single row in Current

66745. Need to reorder the LHS fields according to aiMap

66746. ON CONFLICT policy

66747. ** A write-transaction must be opened before calling this function. ** It performs a single unit of work towards an incremental vacuum. *** If the incremental vacuum is finished after this function has run, ** SQLITE_DONE is returned. If it is not finished, but no error occurred, ** SQLITE_OK is returned. Otherwise an SQLite error code.

66748. OK to return IN_INDEX_NOOP

66749. Name of the table into which we are inserting

66750. Generate code to remove the table from the master table ** on disk.

66751. The inner query or "subquery"

66752. Scan definitions for sqlite3_stmt_scanstatus()

66753. ** Move the iterator to the next entry. *** If an error occurs, an error code is left in Fts5Index.rc. It is not ** considered an error if the iterator reaches EOF, or if it is already at ** EOF when this function is called.

66754. ***** End of vdbeblob.c *****

66755. Find the lowest cost path. pFrom will be left pointing to that path

66756. ** CAPI3REF: Commit And Rollback Notification Callbacks ** METHOD: sqlite3 *** ^The sqlite3_commit_hook() interface registers a callback ** function to be invoked whenever a transaction is [COMMIT | committed]. ** ^Any callback set by a previous call to sqlite3_commit_hook() ** for the same database connection is overridden. ** ^The sqlite3_rollback_hook() interface registers a callback ** function to be invoked whenever a transaction is [ROLLBACK | rolled back]. ** ^Any callback set by a previous call to sqlite3_rollback_hook() ** for the same database connection is overridden. ** ^The pArg argument is passed through to the callback. ** ^If the callback on a commit hook function returns non-zero, ** then the commit is converted into a rollback. *** ^The sqlite3_commit_hook(D,C,P) and sqlite3_rollback_hook(D,C,P) functions ** return the P argument from the previous call of the same function ** on the same [database connection] D, or NULL for ** the first call for each function on D. *** ^The commit and rollback hook callbacks are not reentrant. ** The callback implementation must not do anything that will modify ** the database connection that invoked the callback. Any actions ** to modify the database connection must be deferred until after the ** completion of the [sqlite3_step()] call that triggered the commit ** or rollback hook in the first place. ** Note that running any other SQL statements, including SELECT statements, ** or merely calling [sqlite3_prepare_v2()] and [sqlite3_step()] will modify ** the database connections for the meaning of "modify" in this paragraph. *** ^Registering a NULL function disables the callback. *** ^When the commit hook callback routine returns zero, the [COMMIT] ** operation is allowed to continue normally. ^If the commit hook ** returns non-zero, then the [COMMIT] is converted into a [ROLLBACK]. ** ^The rollback hook is invoked on a rollback that results from a commit ** hook returning non-zero, just as it would be with any other rollback. *** ^For the purposes of this API, a transaction is said to have been ** rolled back if an explicit "ROLLBACK" statement is executed, or ** an error or constraint causes an implicit rollback to occur. ** ^The rollback callback is not invoked if a transaction is ** automatically rolled back because the database connection is closed. *** See also the [sqlite3_update_hook()] interface.

66757. 0x18

66758. SQL_SELECT_MXLEVEL ** Return the largest relative level in the FTS index or indexes.

66759. Reduction in fragmentation

66760. Pointer to term buffer

66761. Base class. Must be first. pBuf & pExtra

66762. Results stored in register inReg

66763. Modify the page-size after the cache has been created.

66764. ** 2012 July 21 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file presents a simple cross-platform threading interface for ** use internally by SQLite. *** A "thread" can be created using sqlite3ThreadCreate(). This thread ** runs independently of its creator until it is joined using ** sqlite3ThreadJoin(), at which point it terminates. *** Threads do not have to be real. It could be that the work of the ** "thread" is done by the main thread at either the sqlite3ThreadCreate() ** or sqlite3ThreadJoin() call. This is, in fact, what happens in ** single threaded systems. Nothing in SQLite requires multiple threads. ** This interface exists so that applications that want to take advantage ** of multiple cores can do so, while also allowing applications to stay ** single-threaded if desired.

66765. The FROM clause of the outer query

66766. 0 normally. Positive == commit flag

66767. ** Encode N integers as varints into a blob.

66768. Opcode: Remainder P1 P2 P3 *** Synopsis: r[P3]=r[P2]%-r[P1] *** Compute the remainder after integer register P2 is divided by ** register P1 and store the result in register P3. ** If the value in register P1 is zero the result is NULL. ** If either operand is NULL, the result is NULL.

66769. **comment:** ** 2007 August 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains low-level memory allocation drivers for when ** SQLite will use the standard C-library malloc/realloc/free interface ** to obtain the memory it needs while adding lots of additional debugging ** information to each allocation in order to help detect and fix memory ** leaks and memory usage errors. *** This file contains implementations of the low-level memory allocation ** routines specified in the sqlite3_mem_methods object.

label: code-design

66770. XOR next input into a[i]

66771. Slots needed in p->azResult[]

66772. Extra dependencies on LEFT JOIN

66773. This OP_Insert is an sql UPDATE

66774. 1==UNIQUE, 2==PRIMARY KEY, 0==CREATE INDEX

66775. End the database scan loop.

66776. ** Forward reference required as the vdbeIncrMergeInit() and ** vdbePmaReaderIncrInit() routines are called mutually recursively when ** building a merge tree.

66777. ** When doing a search of an r-tree, instances of the following structure ** record intermediate results from the tree walk. *** The id is always a node-id. For iLevel>=1 the id is the node-id of ** the node that the RtreeSearchPoint represents. When iLevel==0, however, ** the id is of the parent node and the cell that RtreeSearchPoint ** represents is the iCell-th entry in the parent node.

66778. callback can use this, if desired

66779. ** 2007 May 6 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** \$Id: icu.c,v 1.7 2007/12/13 21:54:11 drh Exp \$ *** This file

implements an integration between the ICU library ("International Components for Unicode", an open-source library for handling unicode data) and SQLite. The integration uses ICU to provide the following to SQLite: An implementation of the SQL regexp() function (and hence REGEXP operator) using the ICU uregex_XX() APIs. Implementations of the SQL scalar upper() and lower() functions for case mapping. Integration of ICU and SQLite collation sequences. An implementation of the LIKE operator that uses ICU to provide case-independent matching.

66780. Add the page to the PGroup LRU list.

66781. No digits

66782. Remove entries from the hash table that point to WAL slots greater than pWal->hdr.mxFrame. This function is called whenever pWal->hdr.mxFrame is decreased due to a rollback or savepoint. At most only the hash table containing pWal->hdr.mxFrame needs to be updated. Any later hash tables will be automatically cleared when pWal->hdr.mxFrame advances to the point where those hash tables are actually needed.

66783. Write search results here

66784. If the update-hook or pre-update-hook will be invoked, set zDb to the name of the db to pass as to it. Also set local pTab to a copy of p4.pTab. Finally, if p5 is true, indicating that this cursor was last moved with OP_Next or OP_Prev, not Seek or NotFound, set VdbeCursor.moveToTarget to the current rowid.

66785. Return true if the page is already in the journal file.

66786. FTS5_STRING node this phrase is part of

66787. Length of string argv[2]

66788. All code in this file should access the global structure above via the alias "pcache1". This ensures that the WSD emulation is used when compiling for systems that do not support real WSD.

66789. If any data has been written (but not committed) to the log file, this function moves the write-pointer back to the start of the transaction. Additionally, the callback function is invoked for each frame written to the WAL since the start of the transaction. If the callback returns other than SQLITE_OK, it is not invoked again and the error code is returned to the caller. Otherwise, if the callback function does not return an error, this function returns SQLITE_OK.

66790. We only need to generate a select expression if there is a limit/offset term to enforce.

66791. Free the pBt->pTmpSpace allocation

66792. EQ seeks only - no range seeks

66793. Sorter cursor

66794. The methods of the json_tree virtual table.

66795. Lower bound on the range. ex: "x>123" Might be NULL

66796. Length of the name

66797. Where clause to locate temp triggers

66798. The following routines are implementations of the MemPage.xCellSize method. Compute the total number of bytes that a Cell needs in the cell data area of the btree-page. The return number includes the cell data header and the local payload, but not any overflow page or the space used by the cell pointer. cellSizePtrNoPayload() => table internal nodes cellSizePtr() => all index nodes & table leaf nodes

66799. Set the variable isMemdb to true for an in-memory database, or false for a file-based database.

66800. ON clause terms may not be used with an index on left table of a LEFT JOIN. Ticket #3015

66801. file handle

66802. Checksum used for sanity checking

66803. Used to iterate through attached tables

66804. See comment below

66805. A table in the database is locked

66806. Database filename (UTF-8)

66807. 0x1a

66808. pTask->list.aMemory can only be non-zero if it was handed memory from the main thread. That only occurs SQLITE_MAX_WORKER_THREADS>0

66809. Insert a record into the %_docszie table. Specifically, do: INSERT OR REPLACE INTO %_docszie(id, sz) VALUES(iRowid, pBuf); If there is no %_docszie table (as happens if the columnsize=0 option is specified when the FTS5 table is created), this function is a no-op.

66810. Check to ensure that the cursor is valid. Restore the cursor if need be. Return any I/O error from the restore operation.

66811. The current column index in pp2

66812. Flags passed through to the VFS

66813. Used by: table_info

66814. Length of the zJson string in bytes

66815. An rbu VFS is implemented using an instance of this structure.

66816. Decoded coordinates

66817. **comment:** The VdbeCoverage macros are used to set a coverage testing point for VDBE branch instructions. The coverage testing points are line numbers in the sqlite3.c source file. VDBE branch coverage testing only works with an amalgamation build. That's ok since a VDBE branch coverage build designed for testing the test suite only. No application should ever ship with VDBE branch coverage measuring turned on. VdbeCoverage(v) // Mark the previously coded instruction as a branch *** VdbeCoverageIf(v, conditional) // Mark previous if conditional true *** VdbeCoverageAlwaysTaken(v) // Previous branch is always taken *** VdbeCoverageNeverTaken(v) // Previous branch is never taken *** Every VDBE branch operation must be tagged with one of the macros above. If not, then when "make test" is run with -DSQLITE_VDBE_COVERAGE and -DSQLITE_DEBUG then an ALWAYS() will fail in the vdbeTakeBranch() routine in vdbe.c, alerting the developer to the missed tag.

label: test

66818. HAVE_LOCALTIME_R

66819. SQLITE_IGNORE_AFP_LOCK_ERRORS

66820. Patch with JsonNode.u.pPatch

66821. If this is the first row, then generate an extra row containing the names of all columns.

66822. Buffer used to hold a poslist

66823. Database connection currently using this Btree

66824. Counters used by sqlite3_stmt_status()

66825. Check if the output file is full or if the input has been exhausted. In either case exit the loop.

66826. **comment:** pExpr points to an expression which implements a function. If it is appropriate to apply the LIKE optimization to that function then set aWc[0] through aWc[2] to the wildcard characters and return TRUE. If the function is not a LIKE-style function then return FALSE. *pIsNocase is set to true if uppercase and lowercase are equivalent for the function (default for LIKE). If the function makes the distinction between uppercase and lowercase (as does GLOB) then *pIsNocase is set to false.

label: code-design

66827. The database size was written into the offset 28 of the header when the transaction started, so we know that the value at offset 28 is nonzero.

66828. Size of array aParam[]

66829. 0x0b

66830. **comment:** Elements above, except pCache, are public. All that follow are private to pcache.c and should not be accessed by other modules. pCache is grouped with the public elements for efficiency.

label: code-design

66831. 279

66832. ORDER BY of joins via index

66833. (290) typename ::= ID|STRING

66834. Database pages read

66835. Coordinate value convert to a double

66836. Expr struct EXPR_TOKENONLYSIZE bytes only

66837. The 2-byte case

66838. The xConnect() and xCreate() methods for the virtual table. All the work is done in function fts3InitVtab().

66839. Integers of known sizes. These typedefs might change for architectures where the sizes vary. Preprocessor macros are available so that the types can be conveniently redefined at compile-type. Like this: *** cc '-DUINTPTR_TYPE=long long int' ...

66840. **comment:** ** Execute SQL code. Return one of the SQLITE_ success/failure ** codes. Also write an error message into memory obtained from ** malloc() and make *pzErrMsg point to that message. *** If the SQL is a query, then for each row in the query result ** the xCallback() function is called. pArg becomes the first ** argument to xCallback(). If xCallback=NULL then no callback ** is invoked, even for queries.
label: code-design

66841. we can skip these cause they were (effectively) done above ** while calculating s
66842. Record pRec is equal to sample i
66843. SQLITE_OMIT_SCHEMA_PRAGMAS
66844. PRIMARY KEY array
66845. ** This constant determines the maximum depth of an FTS expression tree ** that the library will create and use. FTS uses recursion to perform ** various operations on the query tree, so the disadvantage of a large ** limit is that it may allow very large queries to use large amounts ** of stack space (perhaps causing a stack overflow).
66846. ** This function is called while writing an FTS segment each time a leaf o ** node is finished and written to disk. The key (zTerm/nTerm) is guaranteed ** to be greater than the largest key on the node just written, but smaller ** than or equal to the first key that will be written to the next leaf ** node. *** The block id of the leaf node just written to disk may be found in ** (pWriter->aNodeWriter[0].iBlock) when this function is called.
66847. **comment:** Here is inserted the actions which take place when a ** terminal or non-terminal is destroyed. This can happen ** when the symbol is popped from the stack during a ** reduce or during error processing or when a parser is ** being destroyed before it is finished parsing. *** Note: during a reduce, the only symbols destroyed are those ** which appear on the RHS of the rule, but which are *not* used ** inside the C code.
label: code-design

66848. The expression that defines the start bound
66849. 283
66850. ** Unlink a VFS from the linked list
66851. First argument to compare function
66852. ***** End of fts3_tokenizer1.c *****
66853. The cursor number for the corresponding table
66854. Bytes of reserved space at the end of each page
66855. If no database changes have been made, return early.
66856. create_table ::= createkw temp TABLE ifnotexists nm dbnm
66857. Index of VdbeCursor that writes the sqlite_stat1 table
66858. ** Default permissions when creating auto proxy dir
66859. "idx" or "tbl"
66860. ** Break the OR clause into its separate subterms. The subterms are ** stored in a WhereClause structure containing within the WhereOrInfo ** object that is attached to the original OR clause term.
66861. **comment:** ** Type SegmentNode is used by the following three functions to create ** the interior part of the segment b+-tree structures (everything except ** the leaf nodes). These functions and type are only ever used by code ** within the fts3SegWriterXXX() family of functions described above. ***
fts3NodeAddTerm() ** fts3NodeWrite() ** fts3NodeFree() *** When a b+-tree is written to the database (either as a result of a merge ** or the pending-terms table being flushed), leaves are written into the ** database file as soon as they are completely populated. The interior of ** the tree is assembled in memory and written out only once all leaves have ** been populated and stored. This is Ok, as the b+-tree fanout is usually ** very large, meaning that the interior of the tree consumes relatively ** little memory.
label: code-design

66862. Which column of pTab is desired
66863. OUT: Space to write result to
66864. The prepared statement under construction
66865. Cache is valid if this matches Vdbe.cacheCtr
66866. Type of checkpoint
66867. Ensure DROP TABLE is not used on a view, and DROP VIEW is not used ** on a table.
66868. Always instruction 0
66869. ** Make a new pointer to a KeyInfo object
66870. ERROR: insert command gives an output larger than predicted
66871. Invoke the xBegin method. If successful, add the vtab to the ** sqlite3.aVTrans[] array.
66872. ** An instance of the following data structure is used to build doclists ** incrementally. See function fts3PendingListAppend() for details.
66873. The FROM clause of the recursive query
66874. Special case of a FROM clause subquery implemented as a co-routine
66875. Check that MEMDB implies noSync. And an in-memory journal. Since ** this means an in-memory pager performs no IO at all, it cannot encounter ** either SQLITE_IOERR or SQLITE_FULL during rollback or while finalizing ** a journal file. (although the in-memory journal implementation may ** return SQLITE_IOERR_NOMEM while the journal file is being written). It ** is therefore not possible for an in-memory pager to enter the ERROR ** state.
66876. (292) signed ::= minus_num (OPTIMIZED OUT)
66877. Offset to the page header
66878. If the WAL is not empty, return the size of the database.
66879. ** Allocate a new RowSetEntry object that is associated with the ** given RowSet. Return a pointer to the new and completely uninitialized ** objected. *** In an OOM situation, the RowSet.db->mallocFailed flag is set and this ** routine returns NULL.
66880. Move existing slots that come after the newly inserted slots ** out of the way
66881. Output values. Valid only after Fts3SegReaderStep() returns SQLITE_ROW.
66882. If the bPreserve flag is set to true, then the memory cell must already ** contain a valid string or blob value.
66883. Start scanning the virtual table
66884. Node must have already been started. There must be a doclist for a ** leaf node, and there must not be a doclist for an internal node.
66885. ** Create a new FTS5 expression by cloning phrase iPhrase of the ** expression passed as the second argument.
66886. **comment:** Clean up and return
label: code-design

66887. Database size before freeing
66888. EVIDENCE-OF: R-23882-45353 The cell pointer array of a b-tree page ** immediately follows the b-tree page header.
66889. Start a write transaction on these databases
66890. Playback and delete the journal. Drop the database write ** lock and reacquire the read lock. Purge the cache before ** playing back the hot-journal so that we don't end up with ** an inconsistent cache. Sync the hot journal before playing ** it back since the process that crashed and left the hot journal ** probably did not sync it and we are required to always sync ** the journal before playing it back.
66891. Where to start scanning
66892. If the calls to xBestIndex() have so far failed to find a plan ** that requires no source tables at all and does not use an IN(...) ** operator, make a final call to obtain one here.
66893. Number of NULL aLTerm[] entries
66894. Ephemeral. Delete with VDBE
66895. Name of db (i.e. "main", "temp" etc.)
66896. trigger_cmd ::= DELETE FROM trnm tridxby where_opt
66897. Position list data following iTail
66898. Number instances seen so far
66899. ** Global variables.
66900. Default value of this column
66901. **comment:** Column names are determined by the left-most term of a compound select
label: code-design

66902. ** Unbind the value bound to variable i in virtual machine p. This is the ** same as binding a NULL value to the column. If the "i" parameter is ** out of range, then SQLITE_RANGE is returned. Otherwise SQLITE_OK. *** A successful evaluation of this routine acquires the mutex on p. ** the mutex is released if any kind of error occurs. *** The error code stored in database p->db is overwritten with the return ** value in any case.

66903. ** This function is called by the wal module when writing page content ** into the log file. *** This function returns a pointer to a buffer containing the encrypted ** page content. If a malloc fails, this function may return NULL.

66904. 6,7,8

66905. ** Call sqlite3_declare_vtab() based on the contents of the configuration ** object passed as the only argument. Return SQLITE_OK if successful, or ** an SQLite error code if an error occurs.

66906. Number of retries

66907. mxParserStack

66908. Initialize the 4-byte leaf-page header to 0x00.

66909. Figure out which index to search and set iIdx accordingly. If this ** is a prefix query for which there is no prefix index, set iIdx to ** greater than pConfig->nPrefix to indicate that the query will be ** satisfied by scanning multiple terms in the main index. *** If the QUERY_TEST_NOIDX flag was specified, then this must be a ** prefix-query. Instead of using a prefix-index (if one exists), ** evaluate the prefix query using the main FTS index. This is used ** for internal sanity checking by the integrity-check in debug ** mode only.

66910. True if cache must be reset

66911. Second key value passed to hook

66912. Count the number of additional columns needed to create a ** covering index. A "covering index" is an index that contains all ** columns that are needed by the query. With a covering index, the ** original table never needs to be accessed. Automatic indices must ** be a covering index because the index will not be updated if the ** original table changes and the index and table cannot both be used ** if they go out of sync.

66913. ** Main sorter structure. A single instance of this is allocated for each ** sorter cursor created by the VDBE. *** mxKeysize: ** As records are added to the sorter by calls to sqlite3VdbeSorterWrite(), ** this variable is updated so as to be set to the size on disk of the ** largest record in the sorter.

66914. endif aggregate query

66915. Coordinates of node or entry to check

66916. 265

66917. ** Set *pnMax to the largest segment level in the database for the index ** iIndex. *** Segment levels are stored in the 'level' column of the %_segdir table. ** ** Return SQLITE_OK if successful, or an SQLite error code if not.

66918. Unmap the database file. It is possible that external processes ** may have truncated the database file and then extended it back ** to its original size while this process was not holding a lock. ** In this case there may exist a Pager.pMap mapping that appears ** to be the right size but is not actually valid. Avoid this ** possibility by unmapping the db here.

66919. The output line number is small enough that we are still in the ** main program.

66920. ** sqlite3_test_control(SQLITE_TESTCTRL_ALWAYS, int X) *** This action provides a run-time test to see how the ALWAYS and ** NEVER macros were defined at compile-time. *** The return value is ALWAYS(X). *** The recommended test is X==2. If the return value is 2, that means ** ALWAYS() and NEVER() are both no-op pass-through macros, which is the ** default setting. If the return value is 1, then ALWAYS() is either ** hard-coded to true or else it asserts if its argument is false. ** The first behavior (hard-coded to true) is the case if ** SQLITE_TESTCTRL_ASSERT shows that assert() is disabled and the second ** behavior (assert if the argument to ALWAYS() is false) is the case if ** SQLITE_TESTCTRL_ASSERT shows that assert() is enabled. *** The run-time test procedure might look something like this: *** ** if(sqlite3_test_control(SQLITE_TESTCTRL_ALWAYS, 2)==2){ ** // ALWAYS() and NEVER() are no-op pass-through macros ** }else if(sqlite3_test_control(SQLITE_TESTCTRL_ASSERT, 1)){ ** // ALWAYS(x) asserts that x is true. NEVER(x) asserts x is false. ** }else{ ** // ALWAYS(x) is a constant 1. NEVER(x) is a constant 0. ** }

66921. Root BTree page for this table

66922. Offset to the start of the cell pointer array

66923. SELECT statement to encode

66924. ** The input pointer currently points to the first byte of the first field ** of a record consisting of nCol columns. This function ensures the entire ** record is buffered. It does not move the input pointer. *** If successful, SQLITE_OK is returned and *pnByte is set to the size of ** the record in bytes. Otherwise, an SQLite error code is returned. The ** final value of *pnByte is undefined in this case.

66925. True if ok to populate

66926. The unix errno from last I/O error

66927. ** x BETWEEN y AND z *** This is equivalent to ** ** x>=y AND x<=z ** ** X is stored in pExpr->pLeft. ** Y is stored in pExpr->pList->a[0].pExpr. ** Z is stored in pExpr->pList->a[1].pExpr.

66928. The column that is wanted

66929. sqlite3_test_control(SQLITE_TESTCTRL_ISKEYWORD, const char *zWord) *** If zWord is a keyword recognized by the parser, then return the ** number of keywords. Or if zWord is not a keyword, return 0. *** This test feature is only available in the amalgamation since ** the SQLITE_N_KEYWORD macro is not defined in this file if SQLite ** is built using separate source files.

66930. Number of registers to allocate

66931. **comment:** Initialized only to suppress erroneous warning from Clang
label: code-design

66932. ** Parse a timezone extension on the end of a date-time. ** The extension is of the form: ** ** (+-)HH:MM ** ** Or the "zulu" notation: ** ** Z ** ** If the parse is successful, write the number of minutes ** of change in p->tz and return 0. If a parser error occurs, ** return non-zero. *** A missing specifier is not considered an error.

66933. Output values used by 'row' and 'col' tables

66934. Rowid changed in a normal table

66935. **comment:** Cursor for the index, if pIdx!=0. Unused otherwise
label: code-design

66936. **comment:** Case 4: A scan using an index. *** The WHERE clause may contain zero or more equality ** terms ("==" or "IN" operators) that refer to the N ** left-most columns of the index. It may also contain ** inequality constraints (>, <, >= or <=) on the indexed ** column that immediately follows the N equalities. Only ** the right-most column can be an inequality - the rest must ** use the "==" and "IN" operators. For example, if the ** index is on (x,y,z), then the following clauses are all ** optimized: *** x=5 ** x=5 AND y=10 ** x=5 AND y<10 ** x=5 AND y>5 AND y<10 ** x=5 AND y=5 AND z<=10 *** The z<10 term of the following cannot be used, only ** the x=5 term: *** x=5 AND z<10 ** ** N may be zero if there are inequality constraints. ** If there are no inequality constraints, then N is at ** least one. *** This case is also used when there are no WHERE clause ** constraints but an index is selected anyway, in order ** to force the output order to conform to an ORDER BY.
label: code-design

66937. Encoding for the data

66938. If this is an INSERT or UPDATE, read the new.* record.

66939. OP_Found will use a composite key

66940. Iterator error code

66941. The various pragma types

66942. 0xff means setting unknown

66943. If opening the main database, set ENC(db).

66944. Special handling for a compound-select that originates as a VALUES clause.

66945. ** Create an Fts5Iter that iterates through the doclist provided ** as the second argument.

66946. ** Called by the parser to compile an ATTACH statement. *** ATTACH p AS pdbname KEY pKey

66947. ** Return the file handle for the journal file (if it exists). ** This will be either the rollback journal or the WAL file.

66948. The base for radix conversion

66949. 157

66950. same as TK_EQ, synopsis: IF r[P3]==r[P1]

66951. ** 2009 January 28 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the implementation of the sqlite3_backup_XXX()

** API functions and the related features.

66952. P1==1 for end-of-loop

66953. Confirm that aiMap[] contains nVector integer values between 0 and ** nVector-1.

66954. Append a comma separator to the output buffer, if the previous ** character is not '[' or ']'.
66955. 0x1b
66956. List of indexed columns
66957. ALWAYS() justification: eOp is an equality operator due to the ** j<pLoop->u.btree.nEq constraint above. Any equality other ** than WO_IN is captured by the previous "if". So this one ** always has to be WO_IN.
66958. end Loop over all index columns
66959. ** Get a page from the pager. Initialize the MemPage.pBt and ** MemPage.aData elements if needed. See also: btreeGetUnusedPage(). *** If the PAGER_GET_NOCONTENT flag is set, it means that we do not care ** about the content of the page at this time. So do not go to the disk ** to fetch the content. Just fill in the content with zeros for now. ** If in the future we call sqlite3PagerWrite() on this page, that ** means we have started to be concerned about content and the disk ** read should occur at that point.
66960. ** Available fault injectors. Should be numbered beginning with 0.
66961. Size of record header in bytes
66962. ORDERBY+LIMIT on the inner loop
66963. Name of variables
66964. .n =
66965. 17
66966. Logarithm of table size
66967. True if trailing zeros should be removed
66968. The wal-index is divided into pages of WALINDEX_PGSZ bytes each.
66969. Number of XOR operations done
66970. The table does not exist.
66971. Used when p4type is P4_KEYINFO
66972. Assuming no error has occurred, run a "restart" checkpoint with the ** sqlite3rbu.eStage variable set to CAPTURE. This turns on the following ** special behaviour in the rru VFS: *** If the exclusive shm WRITER or READ0 lock cannot be obtained, ** the checkpoint fails with SQLITE_BUSY (normally SQLite would ** proceed with running a passive checkpoint instead of failing). *** Attempts to read from the *-wal file or write to the database file ** do not perform any IO. Instead, the frame/page combinations that ** would be read/written are recorded in the sqlite3rbu.aFrame[] ** array. *** Calls to xShmLock(UNLOCK) to release the exclusive shm WRITER, ** READ0 and CHECKPOINT locks taken as part of the checkpoint are ** no-ops. These locks will not be released until the connection ** is closed. *** Attempting to xSync() the database file causes an SQLITE_INTERNAL ** error. *** As a result, unless an error (i.e. OOM or SQLITE_BUSY) occurs, the ** checkpoint below fails with SQLITE_INTERNAL, and leaves the aFrame[] ** array populated with a set of (frame -> page) mappings. Because the ** WRITER, CHECKPOINT and READ0 locks are still held, it is safe to copy ** data from the wal file into the database file according to the ** contents of aFrame[].
66973. Set to zero if any values are modified
66974. Opcode: ReadCookie P1 P2 P3 * * * * * Read cookie number P3 from database P1 and write it into register P2. ** P3==1 is the schema version. P3==2 is the database format. ** P3==3 is the recommended pager cache size, and so forth. P1==0 is ** the main database file and P1==1 is the database file used to store ** temporary tables. *** There must be a read-lock on the database (either a transaction ** must be started or there must be an open cursor) before ** executing this instruction.
66975. ** If the most recently coded instruction is a constant range constraint ** (a string literal) that originated from the LIKE optimization, then ** set P3 and P5 on the OP_String opcode so that the string will be cast ** to a BLOB at appropriate times. *** The LIKE optimization tries to evaluate "x LIKE 'abc%'" as a range ** expression: "x>='ABC' AND x<'abd'". But this requires that the range ** scan loop run twice, once for strings and a second time for BLOBS. ** The OP_String opcodes on the second pass convert the upper and lower ** bound string constants to blobs. This routine makes the necessary changes ** to the OP_String opcodes for that to happen. ** ** Except, of course, if SQLITE_LIKE_DOESNT_MATCH_BLOBS is defined, then ** only the one pass through the string space is required, so this routine ** becomes a no-op.
66976. Cursor used for index scans (if any)
66977. Size of apPhrase array
66978. The above might be initialized to non-zero. The following need to always ** initially be zero, however.
66979. ** Append a rowid and position-list size field to the writers output.
66980. Test if fcntl() is supported and use POSIX style locks. ** Otherwise fall back to the named semaphore method.
66981. 840
66982. Register holding the offset counter
66983. **comment:** If [0] is unused then [1] is also unused. So we can ** always safely abort as soon as the first unused slot is found
 label: code-design
66984. Next free argvIndex value
66985. Full-text table
66986. selectnowith
66987. Connection handle
66988. The 1-byte case. Overwhelmingly the most common.
66989. Pointer to buffer containing changeset A
66990. ONEPASS defaults to OFF
66991. Encoding of z. 0 for BLOBS
66992. defined(SQLITE_USE_MALLOC_H)
66993. True if db currently has pBt locked
66994. **comment:** LIMIT expression. NULL means not used.
 label: code-design
66995. ** Read the next nByte bytes of data from the PMA p. ** If successful, set *ppOut to point to a buffer containing the data ** and return SQLITE_OK. Otherwise, if an error occurs, return an SQLite ** error code. *** The buffer returned in *ppOut is only valid until the ** next call to this function.
66996. ** The argument to this macro is a file descriptor (type sqlite3_file*). ** Return 0 if it is not open, or non-zero (but not 1) if it is. *** This is so that expressions can be written as: *** if(isOpen(pPager->jfd)){ ... } instead of *** if(pPager->jfd->pMethods){ ... }
66997. Mark the expression is being from the ON or USING clause of a join ** so that the sqlite3ExprCodeTarget() routine will not attempt to move ** it into the Parse.pConstExpr list. We should use a new bit for this, ** for clarity, but we are out of bits in the Expr.flags field so we ** have to reuse the EP_FromJoin bit. Bummer.
66998. The current schema
66999. ** Memory cell pMem contains the context of an aggregate function. ** This routine calls the finalize method for that function. The ** result of the aggregate is stored back into pMem. *** Return SQLITE_ERROR if the finalizer reports an error. SQLITE_OK ** otherwise.
67000. Figure out if we have any triggers and if the table being ** inserted into is a view
67001. Integer value if p4type==P4_INT32
67002. Address of the A>B subroutine
67003. Flags allowed as part of the 4th argument to SegmentReaderIterate()
67004. SQLITE_IOERR
67005. LIKE opt constraints
67006. If the column value is a string, we need a persistent value, not ** a MEM_Ephem value. This branch is a fast short-cut that is equivalent ** to calling sqlite3VdbeSerialGet() and sqlite3VdbeDeepHemeralize().
67007. Release/Acquire the system-level lock
67008. An unrecognized directive. Return an error message.
67009. Function to possibly overload

67010. If pFree is not NULL, it points to the pSpace buffer used ** by a previous call to balance_nonroot(). Its contents are ** now stored either on real database pages or within the ** new pSpace buffer, so it may be safely freed here.

67011. A negative nCol is a special marker meaning that we are currently ** trying to compute the column names. If we enter this routine with ** a negative nCol, it means two or more views form a loop, like this: ** ** CREATE VIEW one AS SELECT * FROM two; ** CREATE VIEW two AS SELECT * FROM one; ** ** Actually, the error above is now caught prior to reaching this point. ** But the following test is still important as it does come up ** in the following: ** ** CREATE TABLE main.ex1(a); ** CREATE TEMP VIEW ex1 AS SELECT a FROM ex1; ** SELECT * FROM temp.ex1;

67012. Segment id to load from

67013. If the target table column is an "INTEGER PRIMARY KEY", add ** "PRIMARY KEY" to the imposter table column declaration.

67014. same as TK_GE, synopsis: IF r[P3]>=r[P1]

67015. SQLITE_OS_WIN_H

67016. **comment:** The P4 parameter is not used
label: code-design

67017. ** Append a DELETE change to the buffer passed as the first argument. Use ** the changeset format if argument bPatchset is zero, or the patchset ** format otherwise.

67018. Remove the FK from the fkeyHash hash table.

67019. The first in an array of registers (see above)

67020. ** Compute the parentage of all nodes in a completed parse.

67021. ***** Begin file random.c *****

67022. 116

67023. ** Rtree virtual table module xColumn method.

67024. Since the names of named mutexes, semaphores, file mappings etc are ** case-sensitive, take advantage of that by uppercasing the mutex name ** and using that as the shared filemapping name.

67025. Name of the table

67026. 231

67027. Fill the index with data and reparse the schema. Code an OP_Expire ** to invalidate all pre-compiled statements.

67028. REGEXP

67029. Value to return in *pnEntry

67030. The spec says there are three possible values for flags. But only ** two of them are actually used

67031. Copy the b-tree node content from page pFrom to page pTo.

67032. The parser action.

67033. Sub-vdbe for trigger program

67034. sqlite3_randomness()

67035. #include "sqlite3.h"

67036. Number of bytes in current term

67037. Convert indexed expr to column

67038. lock the RESERVED byte

67039. A copy of pExpr is used instead of the original, as pExpr contains ** tokens that point to volatile memory. The 'span' of the expression ** is required by pragma table_info.

67040. ** pRoot is the root of an incremental merge-tree with depth nDepth (according ** to vdbeSorterTreeDepth()). pLeaf is the iSeq'th leaf to be added to the ** tree, counting from zero. This function adds pLeaf to the tree. ** ** If successful, SQLITE_OK is returned. If an error occurs, an SQLite error ** code is returned and pLeaf is freed.

67041. SQLITE_INTERRUPT

67042. ** If a ReadFile() or WriteFile() error occurs, invoke this routine ** to see if it should be retried. Return TRUE to retry. Return FALSE ** to give up with an error.

67043. There is no reason any cursor should have an outstanding reference ** to an overflow page belonging to a cell that is being deleted/updated. ** So if there exists more than one reference to this page, then it ** must not really be an overflow page and the database must be corrupt. ** It is helpful to detect this before calling freePage2(), as ** freePage2() may zero the page contents if secure-delete mode is ** enabled. If this 'overflow' page happens to be a page that the ** caller is iterating through or using in some other way, this ** can be problematic.

67044. ** If the following function pointer is not NULL and if ** SQLITE_ENABLE_IOTRACE is enabled, then messages describing ** I/O active are written using this function. These messages ** are intended for debugging activity only.

67045. Number of loaded extensions

67046. Cursor that holds the new sorter

67047. Case-sensitive LIKE-type function

67048. String, possibly obtained from sqlite3_malloc

67049. Current checkout, including internal fragmentation

67050. Rowid that P1 current points to

67051. Make the IdChar function accessible from ctime.c

67052. varintRecordCompareInt() and varintRecordCompareString() both assume ** that the size-of-header varint that occurs at the start of each record ** fits in a single byte (i.e. is 127 or less). varintRecordCompareInt() ** also assumes that it is safe to overread a buffer by at least the ** maximum possible legal header size plus 8 bytes. Because there is ** guaranteed to be at least 74 (but not 136) bytes of padding following each ** buffer passed to varintRecordCompareInt() this makes it convenient to ** limit the size of the header to 64 bytes in cases where the first field ** is an integer. ** ** The easiest way to enforce this limit is to consider only records with ** 13 fields or less. If the first field is an integer, the maximum legal ** header size is (12*5 + 1 + 1) bytes.

67053. ** Destructor for a stmt_cursor.

67054. ** Return the size of an outstanding allocation, in bytes. The ** size returned omits the 8-byte header overhead. This only ** works for chunks that are currently checked out.

67055. IN/OUT: Point to read varint from

67056. Next session object on same db.

67057. True if a plan with no prereqs seen

67058. Figure out how many memory cells we will need then allocate them.

67059. ** We say the WhereLoop is "one-row" if it generates no more than one ** row of output. A WhereLoop is one-row if all of the following are true: ** (a) All index columns match with WHERE_COLUMN_EQ. ** (b) The index is unique ** Any WhereLoop with an WHERE_COLUMN_EQ constraint on the rowid is one-row. ** Every one-row WhereLoop will have the WHERE_ONEROW bit set in wsFlags. ** ** We say the WhereLoop is "order-distinct" if the set of columns from ** that WhereLoop that are in the ORDER BY clause are different for every ** row of the WhereLoop. Every one-row WhereLoop is automatically ** order-distinct. A WhereLoop that has no columns in the ORDER BY clause ** is not order-distinct. To be order-distinct is not quite the same as being ** UNIQUE since a UNIQUE column or index can have multiple rows that ** are NULL and NULL values are equivalent for the purpose of order-distinct. ** To be order-distinct, the columns must be UNIQUE and NOT NULL. ** ** The rowid for a table is always UNIQUE and NOT NULL so whenever the ** rowid appears in the ORDER BY clause, the corresponding WhereLoop is ** automatically order-distinct.

67060. The Current table

67061. <tbl> MATCH ?

67062. Bytes consumed per cell

67063. Column constrained. -1 for ROWID

67064. **comment:** COVERAGE: The (pVal->z==0) branch is never true using current versions ** of SQLite. If a malloc fails in an sqlite3_value_xxx() function, either ** the (pVal->z) variable remains as it was or the type of the value is ** set to SQLITE_NULL.
label: code-design

67065. Number of a[] slots allocated

67066. ** The methods above are in versions 1 and 2 of the sqlite_vfs object. ** Those below are for version 3 and greater.

67067. ** Return the number of bytes currently allocated at address p.

67068. ** This object holds a record which has been parsed out into individual ** fields, for the purposes of doing a comparison. ** ** A record is an object that contains one or more fields of data. ** Records are used to store the content of a table row and to store ** the key of an index. A blob encoding of a record is created by **

the OP_MakeRecord opcode of the VDBE and is disassembled by the ** OP_Column opcode. *** An instance of this object serves as a "key" for doing a search on ** an index b+tree. The goal of the search is to find the entry that ** is closed to the key described by this object. This object might hold ** just a prefix of the key. The number of fields is given by ** pKeyInfo->nField. *** The r1 and r2 fields are the values to return if this key is less than ** or greater than a key in the btree, respectively. These are normally ** -1 and +1 respectively, but might be inverted to +1 and -1 if the b-tree ** is in DESC order. *** The key comparison functions actually return default_rc when they find ** an equals comparison. default_rc can be -1, 0, or +1. If there are ** multiple entries in the b-tree with the same key (when only looking ** at the first pKeyInfo->nFields,) then default_rc can be set to -1 to ** cause the search to find the last match, or +1 to cause the search to ** find the first match. *** The key comparison functions will set eqSeen to true if they ever ** get equal results when comparing this structure to a b-tree record. ** When default_rc!=0, the search might end up on the record immediately ** before the first match or immediately after the last match. The ** eqSeen field will indicate whether or not an exact match exists in the ** b-tree.

67069. The following table contains information about every rule that ** is used during the reduce.

67070. IN: Elements in array *paLeft

67071. Set up the required files for pIncr. A multi-threaded IncrMerge object ** requires two temp files to itself, whereas a single-threaded object ** only requires a region of pTask->file2.

67072. If an error occurs during a ROLLBACK, we can no longer trust the pager ** cache. So call pager_error() on the way out to make any error persistent.

67073. Used when p4type is P4_MEMORY

67074. An SQLITE_MISUSE error occurred

67075. 1 -> main journal. 0 -> sub-journal.

67076. OUT: Final number of checkpointed frames

67077. Number of bytes in key

67078. Address of subroutine to manifest a subquery

67079. Length of the source file

67080. Disable lookaside memory allocation

67081. Generate a select expression tree to enforce the limit/offset ** term for the DELETE or UPDATE statement. For example: ** DELETE FROM table_a WHERE col1=1 ORDER BY col2 LIMIT 1 OFFSET 1 ** becomes: ** DELETE FROM table_a WHERE rowid IN (** SELECT rowid FROM table_a WHERE col1=1 ORDER BY col2 LIMIT 1 OFFSET 1 **);

67082. ** Generate code for a boolean expression such that a jump is made ** to the label "dest" if the expression is true but execution ** continues straight thru if the expression is false. *** If the expression evaluates to NULL (neither true nor false), then ** take the jump if the jumpIfNull flag is SQLITE_JUMPNULL. ** This code depends on the fact that certain token values (ex: TK_EQ) ** are the same as opcode values (ex: OP_Eq) that implement the corresponding ** operation. Special comments in vdbe.c and the mkopcodeh.awk script in ** the make process cause these values to align. Assert(s) in the code ** below verify that the numbers are aligned correctly.

67083. Reserved space for locks

67084. MEM cell holding key for the record

67085. ** Unless it is NULL, the argument must be an UnpackedRecord object returned ** by an earlier call to sqlite3Stat4ProbeSetValue(). This call deletes ** the object.

67086. If the database supports auto-vacuum, write an entry in the pointer-map ** to indicate that the page is free.

67087. EXISTS => nothing

67088. OUT: Block number in next layer down

67089. on_opt ::=

67090. ** The sqlite3_mutex_held() and sqlite3_mutex_notheld() routine are ** intended for use inside assert() statements.

67091. Flags for the fsync

67092. ***** End of fts3_expr.c *****

67093. Index of iDelete cell in pLeaf

67094. ***** Include btree.h in the middle of sqliteInt.h *****

67095. Position just read from pList

67096. Beginning of the header. 0 most pages. 100 page 1

67097. Property flags associated with various pragma.

67098. Matchinfo format string

67099. Translate the schema name in zDb into a pointer to the corresponding ** schema. If not found, pSchema will remain NULL and nothing will match ** resulting in an appropriate error message toward the end of this routine

67100. The "wsdAutoext" macro will resolve to the autoextension ** state vector. If writable static data is unsupported on the target, ** we have to locate the state vector at run-time. In the more common ** case where writable static data is supported, wsdStat can refer directly ** to the "sqlite3Autoext" state vector declared above.

67101. ** The file-handle passed as the only argument is open on a journal file. ** Return true if this "journal file" is currently stored in heap memory, ** or false otherwise.

67102. ** The json_parse(JSON) function returns a string which describes ** a parse of the JSON provided. Or it returns NULL if JSON is not ** well-formed.

67103. True if rbu_count exists

67104. **comment:** Use register aRegIdx[i] for index i. 0 for unused

label: code-design

67105. eidlist ::= eidlist COMMA nm collate sortorder

67106. If the current number of pages in the file is greater than the ** configured maximum pager number, increase the allowed limit so ** that the file can be read.

67107. 0x4

67108. Size of doclist in current entry

67109. ** Populate the buffer pInfo->aMatchinfo[] with an array of integers to ** be returned by the matchinfo() function. Argument zArg contains the ** format string passed as the second argument to matchinfo (or the ** default value "pcx" if no second argument was specified). The format ** string has already been validated and the pInfo->aMatchinfo[] array ** is guaranteed to be large enough for the output. *** If bGlobal is true, then populate all fields of the matchinfo() output. ** If it is false, then assume that those fields that do not change between ** rows (i.e. FTS3_MATCHINFO_NPHRASE, NCOL, NDOC, AVGLENGTH and part of HITS) ** have already been populated. *** Return SQLITE_OK if successful, or an SQLite error code if an error ** occurs. If a value other than SQLITE_OK is returned, the state the ** pInfo->aMatchinfo[] buffer is left in is undefined.

67110. ** Initialize an array of N Mem element.

67111. ** When this function is called, *ppRec points to the start of a record ** that contains nCol values. This function advances the pointer *ppRec ** until it points to the byte immediately following that record.

67112. IMPLEMENTATION-OF: R-63124-39300 The sqlite3_sourceid() function returns a ** pointer to a string constant whose value is the same as the ** SQLITE_SOURCE_ID C preprocessor macro.

67113. nearphrases ::= nearphrases phrase

67114. Largest payload of any cell on this page

67115. A frame is only valid if the page number is greater than zero.

67116. select

67117. Callback for SELECTs

67118. No-op versions of the explainXXX() functions and macros.

67119. ** Make sure virtual table pTab is contained in the pParse->apVirtualLock[] ** array so that an OP_VBegin will get generated for it. Add pTab to the ** array if it is missing. If pTab is already in the array, this routine ** is a no-op.

67120. Opcode: NoConflict P1 P2 P3 P4 *** Synopsis: key=r[P3@P4] *** If P4==0 then register P3 holds a blob constructed by MakeRecord. If ** P4>0 then register P3 is the first of P4 registers that form an unpacked ** record. *** Cursor P1 is on an index btree. If the record identified by P3 and P4 ** contains any NULL value, jump immediately to P2. If all terms of the ** record are not-NULL then a check is done to determine if any row in the ** P1 index btree has a matching key prefix. If there are no matches, jump ** immediately to P2. If there is a match, fall through and leave the P1 ** cursor pointing to the matching row. *** This opcode is similar to OP_NotFound with the exceptions that the ** branch is always taken if any part of the search key input is NULL. *** This operation leaves the cursor in a state where it cannot be ** advanced in either direction. In other words, the Next and Prev ** opcodes do not work after this operation. *** See also: NotFound, Found, NotExists

67121. ROWID value for this entry

67122. If the above was successful, each component iterators now points ** to the first entry in its segment. In this case initialize the ** aFirst[] array. Or, if an error has occurred, free the iterator ** object and set the output variable to NULL.

67123. Functions used to configure a Pager object.

67124. ** Return a 32-bit integer value extracted from a string. If the ** string is not an integer, just return 0.

67125. If we succeeded in making the shared memory handle, map it.

67126. If there is more header available for parsing in the record, try ** to extract additional fields up through the p2+1-th field

67127. Callback function to invoke for phrases

67128. ** Compile the UTF-16 encoded SQL statement zSql into a statement handle.

67129. 0x19

67130. SQL used to determine first block

67131. Number of args

67132. ** Set a flag in the vdbe to update the change counter when it is finalised ** or reset.

67133. One of several kinds of errors

67134. ** Return the number of rows inserted. If this routine is ** generating code because of a call to sqlite3NestedParse(), do not ** invoke the callback function.

67135. The following was already done in fts5WriteInit():

67136. OUT: Hash value

67137. Number of pages on the overflow chain

67138. Shifts left before out of the error

67139. Resolve names in all CHECK constraint expressions.

67140. TRIGGER => ID

67141. ***** Include os_common.h in the middle of os_win.c *****

67142. **comment:** Cannot be both CLEAN and DIRTY
label: code-design

67143. ** This function is called to edit the position list associated with ** the phrase object passed as the fifth argument according to a NEAR ** condition. For example: *** abc NEAR/5 "def ghi" *** Parameter nNear is passed the NEAR distance of the expression (5 in ** the example above). When this function is called, *paPoslist points to ** the position list, and *pnToken is the number of phrase tokens in, the ** phrase on the other side of the NEAR operator to pPhrase. For example, ** if pPhrase refers to the "def ghi" phrase, then *paPoslist points to ** the position list associated with phrase "abc". *** All positions in the pPhrase position list that are not sufficiently ** close to a position in the *paPoslist position list are removed. If this ** leaves 0 positions, zero is returned. Otherwise, non-zero. *** Before returning, *paPoslist is set to point to the position lsit ** associated with pPhrase. And *pnToken is set to the number of tokens in ** pPhrase.

67144. segdir.start_block

67145. Structure of specific index

67146. ** Install the pre-update hooks on the session object passed as the only ** argument.

67147. Size of argv array

67148. Return the total number of outstanding page references

67149. return if parameter is missing

67150. ** Delete a session object previously allocated using sqlite3session_create().

67151. Update the private copy of the header.

67152. ** An object of this type is created for each virtual table present in ** the database schema. *** If the database schema is shared, then there is one instance of this ** structure for each database connection (sqlite3*) that uses the shared ** schema. This is because each database connection requires its own unique ** instance of the sqlite3_vtab* handle used to access the virtual table ** implementation. sqlite3_vtab* handles can not be shared between ** database connections, even when the rest of the in-memory database ** schema is shared, as the implementation often stores the database ** connection handle passed to it via the xConnect() or xCreate() method ** during initialization internally. This database connection handle may ** then be used by the virtual table implementation to access real tables ** within the database. So that they appear as part of the callers ** transaction, these accesses need to be made via the same database ** connection as that used to execute SQL operations on the virtual table. *** All VTable objects that correspond to a single table in a shared ** database schema are initially stored in a linked-list pointed to by ** the Table.pVTable member variable of the corresponding Table object. ** When an sqlite3_prepare() operation is required to access the virtual ** table, it searches the list for the VTable that corresponds to the ** database connection doing the preparing so as to use the correct ** sqlite3_vtab* handle in the compiled query. *** When an in-memory Table object is deleted (for example when the ** schema is being reloaded for some reason), the VTable objects are not ** deleted and the sqlite3_vtab* handles are not xDisconnect()ed ** immediately. Instead, they are moved from the Table.pVTable list to ** another linked list headed by the sqlite3_pDisconnect member of the ** corresponding sqlite3 structure. They are then deleted/xDisconnected ** next time a statement is prepared using said sqlite3*. This is done ** to avoid deadlock issues involving multiple sqlite3.mutex mutexes. ** Refer to comments above function sqlite3VtabUnlockList() for an ** explanation as to why it is safe to add an entry to an sqlite3.pDisconnect ** list without holding the corresponding sqlite3.mutex mutex. *** The memory for objects of this type is always allocated by ** sqlite3DbMalloc(), using the connection handle stored in VTable.db as ** the first argument.

67153. Jump here if we hit the LIMIT

67154. ** Remove the cell with index iCell from node pNode.

67155. ** Initialize and deinitialize the operating system interface.

67156. Malloc'd master-journal file descriptor

67157. 226

67158. ***** The json_each virtual table*****

67159. Jump here to skip this query

67160. If work was actually accomplished...

67161. ** Set the busy handler function. *** The pager invokes the busy-handler if sqlite3OsLock() returns ** SQLITE_BUSY when trying to upgrade from no-lock to a SHARED lock, ** or when trying to upgrade from a RESERVED lock to an EXCLUSIVE ** lock. It does *not* invoke the busy handler when upgrading from ** SHARED to RESERVED, or when upgrading from SHARED to EXCLUSIVE ** (which occurs during hot-journal rollback). Summary: *** Transition | Invokes xBusyHandler ** ----- ** NO_LOCK -> SHARED_LOCK | Yes ** SHARED_LOCK -> RESERVED_LOCK | No ** SHARED_LOCK -> EXCLUSIVE_LOCK | No ** RESERVED_LOCK -> EXCLUSIVE_LOCK | Yes *** If the busy-handler callback returns non-zero, the lock is ** retried. If it returns zero, then the SQLITE_BUSY error is ** returned to the caller of the pager API function.

67162. At this point the client has a lock on an aReadMark[] slot holding ** a value equal to or smaller than pSnapshot->mxFrame, but pWal->hdr ** is populated with the wal-index header corresponding to the head ** of the wal file. Verify that pSnapshot is still valid before ** continuing. Reasons why pSnapshot might no longer be valid: *** (1) The WAL file has been reset since the snapshot was taken. ** In this case, the salt will have changed. *** (2) A checkpoint as been attempted that wrote frames past ** pSnapshot->mxFrame into the database file. Note that the ** checkpoint need not have completed for this to cause problems.

67163. Apply the requested affinity to all inputs

67164. The cursor used for this query

67165. Fall through into OP_Rewind

67166. The value to apply affinity to

67167. Term to search for

67168. !defined(SQLITE_OMIT_VIRTUALTABLE)

67169. This function is only called for rollback pagers in WRITER_DBMOD state.

67170. If the DISTINCT keyword was present on the SELECT statement ** and this row has been seen before, then do not make this row ** part of the result.

67171. Pointer to hash table to clear

67172. The VDBE cursor number used to access this table

67173. Generate code to evaluate all constraint terms using == or IN ** and store the values of those terms in an array of registers ** starting at regBase.

67174. Populate the new SessionTable object and link it into the list. ** The new object must be linked onto the end of the list, not ** simply added to the start of it in order to ensure that tables ** appear in the correct order when a changeset or patchset is ** eventually generated.

67175. Pointer to filter object

67176. The node to render

67177. Nr. of entries in aPgn[] and aIndex[]

67178. True to run in heap-memory mode
67179. RHS is null
67180. 0x03
67181. ** Write iAmt bytes of content into the WAL file beginning at iOffset. ** Do a sync when crossing the p->iSyncPoint boundary. ** ** In other words, if iSyncPoint is in between iOffset and iOffset+iAmt, ** first write the part before iSyncPoint, then sync, then write the ** rest.
67182. (300) defer_subclause_opt ::= defer_subclause (OPTIMIZED OUT)
67183. ** The default size of a disk sector
67184. Current ordered scan item
67185. Initialize a term iterator for each term in the phrase
67186. Available memory locations begin here
67187. Cost of running each loop
67188. ** This routine is used for testing and analysis only.
67189. If the PRIMARY KEY expression is NULL, then use OP_NewRowid ** to generate a unique primary key value.
67190. 9x
67191. **comment:** ** This function is called after invoking an sqlite3_value_XXX function on a ** column value (i.e. a value returned by evaluating an SQL expression in the ** select list of a SELECT statement) that may cause a malloc() failure. If ** malloc() has failed, the threads mallocFailed flag is cleared and the result ** code of statement pStmt set to SQLITE_NOMEM. ** ** Specifically, this is called from within: ** ** sqlite3_column_int() ** sqlite3_column_int64() ** sqlite3_column_text() ** sqlite3_column_text16() ** sqlite3_column_real() ** sqlite3_column_bytes() ** sqlite3_column_bytes16() ** sqlite3_column_blob()
label: code-design
67192. ** This routine does per-connection function registration. Most ** of the built-in functions above are part of the global function set. ** This routine only deals with those that are not global.
67193. **comment:** ** The following group of routines make deep copies of expressions, ** expression lists, ID lists, and select statements. The copies can ** be deleted (by being passed to their respective ...Delete() routines) ** without effecting the originals. ** ** The expression list, ID, and source lists return by sqlite3ExprListDup(), ** sqlite3IdListDup(), and sqlite3SrcListDup() can not be further expanded ** by subsequent calls to sqlite*ListAppend() routines. ** ** Any tables that the SrcList might point to are not duplicated. ** ** The flags parameter contains a combination of the EXPRDUP_XXX flags. ** If the EXPRDUP_REDUCE flag is set, then the structure returned is a ** truncated version of the usual Expr structure that will be stored as ** part of the in-memory representation of the database schema.
label: code-design
67194. ** This routine frees the space the sqlite3_get_table() malloced.
67195. Allocated size of apCell, szCell, aFrom.
67196. Append "UPDATE tbl SET "
67197. Free a prior allocation
67198. no, really, unlock.
67199. Result returned from sqlite3_get_table()
67200. Name assigned to this module
67201. Check that the stack is large enough to grow by a single entry ** if the RHS of the rule is empty. This ensures that there is room ** enough on the stack to push the LHS value
67202. Arguments to agg function
67203. Round up request size to allocation size
67204. zFilename encoded in UTF-8 instead of UTF-16
67205. Operators and special symbols
67206. Alphabetics or '_'. Usable in a keyword
67207. UPDATE
67208. ** Unlink the chunk at mem3.aPool[i] from list it is currently ** on. *pRoot is the list that i is a member of.
67209. #ifndef SQLITE OMIT_WAL
67210. The lookup key
67211. Number of cells in apCell[]
67212. One character past the end of input text
67213. 280
67214. Return true if index X is a UNIQUE index
67215. OOM will cause exit after sqlite3Select()
67216. Size of buffer zDoc in bytes
67217. If this was page 1, then restore the value of Pager.dbFileVers. ** Do this before any decoding.
67218. Exit the mutexes and free the backup context structure.
67219. Iterator object to initialize
67220. ***** Include pcache.h in the middle of sqliteInt.h *****
67221. Test all CHECK constraints
67222. Add sign
67223. When the new root page was allocated, page 1 was made writable in ** order either to increase the database filesize, or to decrement the ** freelist count. Hence, the sqlite3BtreeUpdateMeta() call cannot fail.
67224. Maximum bit index. Max iSize is 4,294,967,296.
67225. The %p conversion
67226. Try to match the ORDER BY expression against an expression ** in the result set. Return an 1-based index of the matching ** result-set entry.
67227. Opcode: IncrVacuum P1 P2 * * * * * Perform a single step of the incremental vacuum procedure on ** the P1 database. If the vacuum has finished, jump to instruction ** P2. Otherwise, fall through to the next instruction.
67228. Handle the common case of integer comparison here, as an ** optimization, to avoid a call to sqlite3MemCompare()
67229. Content of the node, as should be on disk
67230. Modify the sqlite_master table to use the new table name.
67231. Previous index in the loop
67232. IMP: R-49053-54554
67233. 105
67234. ** Free the phrase object passed as the second argument.
67235. Initialize the shared memory if we're supposed to
67236. Parse structure for main program (or NULL)
67237. Length of the string so far
67238. Enlarge pJson->zBuf so that it can hold at least N more bytes. ** Return zero on success. Return non-zero on an OOM error
67239. ** Check if another file-handle holds a RESERVED lock on an rbuVfs-file.
67240. A stat() call may fail for various reasons. If this happens, it is ** almost certain that an open() call on the same path will also fail. ** For this reason, if an error occurs in the stat() call here, it is ** ignored and -1 is returned. The caller will try to open a new file ** descriptor on the same path, fail, and return an error to SQLite. ** ** Even if a subsequent open() call does succeed, the consequences of ** not searching for a reusable file descriptor are not dire.
67241. Populate the output variable and return success.
67242. The page size
67243. Opcode: CreateTable P1 P2 * * * * * Synopsis: r[P2]=root iDb=P1 ** ** Allocate a new table in the main database file if P1==0 or in the ** auxiliary database file if P1==1 or in an attached database if ** P1>1. Write the root page number of the new table into ** register P2 ** ** The difference between a table and an index is this: A table must ** have a 4-byte integer key and can have arbitrary data. An index ** has an arbitrary key but no data. ** ** See also: CreateIndex
67244. True while initialization in progress
67245. ** If cursors, triggers, views and subqueries are all omitted from ** the build, then none of the following routines, except for ** sqlite3SelectDup(), can be called. sqlite3SelectDup() is sometimes ** called with a NULL argument.

67246. ** Return a simple checksum value based on the arguments.

67247. **comment:** ** CAPI3REF: Finalize A Changeset Iterator *** This function is used to finalize an iterator allocated with ** [sqlite3changeset_start()]. *** This function should only be called on iterators created using the ** [sqlite3changeset_start()] function. If an application calls this ** function with an iterator passed to a conflict-handler by ** [sqlite3changeset_apply()], [SQLITE_MISUSE] is immediately returned and the ** call has no effect. *** If an error was encountered within a call to an sqlite3changeset_xxx() ** function (for example an [SQLITE_CORRUPT] in [sqlite3changeset_next()]) or an ** [SQLITE_NOMEM] in [sqlite3changeset_new()] then an error code corresponding ** to that error is returned by this function. Otherwise, SQLITE_OK is ** returned. This is to allow the following pattern (pseudo-code): *** ** sqlite3changeset_start(); ** while(SQLITE_ROW==sqlite3changeset_next()){ ** // Do something with change. ** } ** rc = sqlite3changeset_finalize(); ** if(rc!=SQLITE_OK) { ** // An error has occurred ** }

label: code-design

67248. ** Allocate nByte bytes of memory.

67249. ** This function is called from within the xConnect() or xCreate() method to ** determine the node-size used by the rtree table being created or connected ** to. If successful, pRtree->iNodeSize is populated and SQLITE_OK returned. ** Otherwise, an SQLite error code is returned. *** If this function is being called as part of an xConnect(), then the rtree ** table already exists. In this case the node-size is determined by inspecting ** the root node of the tree. *** Otherwise, for an xCreate(), use 64 bytes less than the database page-size. ** This ensures that each node is stored on a single database page. If the ** database page-size is so large that more than RTREE_MAXCELLS entries ** would fit in a single node, use a smaller node-size.

67250. 680

67251. File handle for the database file

67252. Generate code that will load into register regOut a value that is ** appropriate for the iIdxCol-th column of index pIdx.

67253. Check for the rbu_count table. If it does not exist, or if an error ** occurs, nPhaseOneStep will be left set to -1.

67254. List of tables in current patch

67255. aiRowLogEst values come from sqlite_stat1

67256. All nFrom paths at the previous level

67257. Information about each column

67258. There already exists a WhereLoop on the list that is better ** than pTemplate, so just ignore pTemplate

67259. Buffer containing string or blob data

67260. IMP: R-61914-48074

67261. Btree

67262. ** Disable MMAP on platforms where it is known to not work

67263. The ram filesystem has no write behind ** so it is ordered

67264. Canonical filename

67265. Opcode: Concat P1 P2 P3 * * * Synopsis: r[P3]=r[P2]+r[P1] *** Add the text in register P1 onto the end of the text in ** register P2 and store the result in register P3. ** If either the P1 or P2 text are NULL then store NULL in P3. *** P3 = P2 || P1 *** It is illegal for P1 and P3 to be the same register. Sometimes, ** if P3 is the same register as P2, the implementation is able ** to avoid a memcpy().

67266. 7 -> NOTINDEXED

67267. Number of dimensions

67268. ** Clear the column names from every VIEW in database idx.

67269. ** Return true if it desirable to avoid allocating a new page cache ** entry. *** If memory was allocated specifically to the page cache using ** SQLITE_CONFIG_PAGECACHE but that memory has all been used, then ** it is desirable to avoid allocating a new page cache entry because ** presumably SQLITE_CONFIG_PAGECACHE was suppose to be sufficient ** for all page cache needs and we should not need to spill the ** allocation onto the heap. *** Or, the heap is used for all page cache memory but the heap is ** under memory pressure, then again it is desirable to avoid ** allocating a new page cache entry in order to avoid stressing ** the heap even further.

67270. ** Streaming version of sqlite3changeset_start().

67271. The Btree to which this cursor belongs

67272. Forward references to VFS methods

67273. Figure out the name of the primary key index for the current table. ** This is needed for the argument to "PRAGMA index_xinfo". Set ** zIdx to point to a null-terminated string containing this name.

67274. ** Add a single new WhereTerm entry to the WhereClause object pWC. ** The new WhereTerm object is constructed from Expr p and with wtFlags. ** The index in pWC->a[] of the new WhereTerm is returned on success. ** 0 is returned if the new WhereTerm could not be added due to a memory ** allocation error. The memory allocation failure will be recorded in ** the db->mallocFailed flag so that higher-level functions can detect it. *** This routine will increase the size of the pWC->a[] array as necessary. ** If the wtFlags argument includes TERM_DYNAMIC, then responsibility ** for freeing the expression p is assumed by the WhereClause object pWC. ** This is true even if this routine fails to allocate a new WhereTerm. *** WARNING: This routine might reallocate the space used to store ** WhereTerms. All pointers to WhereTerms should be invalidated after ** calling this routine. Such pointers may be reinitialized by referencing ** the pWC->a[] array.

67275. SQL_CHOMP_SEGDIR ** Update the start_block (:1) and root (:2) fields of the %_segdir ** entry located on absolute level :3 with index :4.

67276. pPager->errMask = 0;

67277. For looping over existing elements

67278. ** Values used as part of the flags argument passed to IndexQuery().

67279. ** Return the size of a pcache allocation

67280. Top of the loop

67281. ***** End of treeview.c *****

67282. os_win.c

67283. pC->aRow does not have to hold the entire row, but it does at least ** need to cover the header of the record. If pC->aRow does not contain ** the complete header, then set it to zero, forcing the header to be ** dynamically allocated.

67284. idx = (lwr+upr)/2

67285. ** Close the dynamic library handle pHandle.

67286. ** This function is used to create delta-encoded serialized lists of FTS3 ** varints. Each call to this function appends a single varint to a list.

67287. Check that the wal file has not been wrapped. Assuming that it has ** not, also check that no checkpointer has attempted to checkpoint any ** frames beyond pSnapshot->mxFrame. If either of these conditions are ** true, return SQLITE_BUSY_SNAPSHOT. Otherwise, overwrite pWal->hdr ** with *pSnapshot and set *pChanged as appropriate for opening the ** snapshot.

67288. Add a field to the old.* record. This is omitted if this modules is ** currently generating a patchset.

67289. expr ::= expr IS expr

67290. Offsets of each character in utf-8 input

67291. Restriction (2a)

67292. ** Return the pPager->iDataVersion value

67293. For a LEFT OUTER JOIN, generate code that will record the fact that ** at least one row of the right table has matched the left table.

67294. If there is a doclist-index

67295. ** The mutex object

67296. Estimate that a[1] is 10, a[2] is 9, a[3] is 8, a[4] is 7, a[5] is ** 6 and each subsequent value (if any) is 5.

67297. Defer right-most ORDER BY of a compound

67298. Show full column names on SELECT

67299. Invoke the pre-update-hook if required.

67300. ** Generate VDBE code for a BEGIN statement.

67301. ** The implementation of user-defined scalar function fts5_rowid().

67302. If rowid is changing, make sure the new rowid does not previously ** exist in the table.

67303. ** Generate code to evaluate an expression and store the results ** into a register. Return the register number where the results ** are stored. *** If the register is a temporary register that can be deallocated, ** then write its number into *pReg. If the result register is not ** a temporary, then set *pReg to zero. *** If pExpr is a constant, then this routine might generate this ** code to fill the register in the initialization section of the ** VDBE program, in order to factor it out of the evaluation loop.

67304. **comment:** Clean up and report errors.
label: code-design
67305. i0, i1, i2... such that aPgn0[iN] ascend
67306. ** NDEBUG and SQLITE_DEBUG are opposites. It should always be true that ** defined(NDEBUG)==!defined(SQLITE_DEBUG). If this is not currently true, ** make it true by defining or undefining NDEBUG. *** Setting NDEBUG makes the code smaller and faster by disabling the ** assert() statements in the code. So we want the default action ** to be for NDEBUG to be set and NDEBUG to be undefined only if SQLITE_DEBUG ** is set. Thus NDEBUG becomes an opt-in rather than an opt-out ** feature.
67307. The SQLITE_ColumnCache flag disables the column cache. This is used ** for testing only - to verify that SQLite always gets the same answer ** with and without the column cache.
67308. Sizeof aRowid[]
67309. **comment:** TUNING: Because uncertainties in the estimates for skip-scan queries, ** add a 1.375 fudge factor to make skip-scan slightly less likely.
label: code-design
67310. Value of 'pagetype' column
67311. The state-number, or reduce action in SHIFTREDUCE
67312. Value for 'level' column of %_segdir
67313. ** If the subquery is the right operand of a LEFT JOIN, then the ** subquery may not be a join itself. Example of why this is not allowed: ** t1 LEFT OUTER JOIN (t2 JOIN t3) ** ** If we flatten the above, we would get ** (t1 LEFT OUTER JOIN t2) JOIN t3 ** ** which is not at all the same thing. ** ** If the subquery is the right operand of a LEFT JOIN, then the outer ** query cannot be an aggregate. This is an artifact of the way aggregates ** are processed - there is no mechanism to determine if the LEFT JOIN ** table should be all-NULL. ** ** See also tickets #306, #350, and #3300.
67314. The eMode parameter is always valid
67315. **comment:** ** CAPI3REF: Mutex Types ** ** The [sqlite3_mutex_alloc()] interface takes a single argument ** which is one of these integer constants. ** ** The set of static mutexes may change from one SQLite release to the ** next. Applications that override the built-in mutex logic must be ** prepared to accommodate additional static mutexes.
label: code-design
67316. ** Compare two search points. Return negative, zero, or positive if the first ** is less than, equal to, or greater than the second. ** ** The rScore is the primary key. Smaller rScore values come first. ** If the rScore is a tie, then use iLevel as the tie breaker with smaller ** iLevel values coming first. In this way, if rScore is the same for all ** SearchPoints, then iLevel becomes the deciding factor and the result ** is a depth-first search, which is the desired default behavior.
67317. ** One object per %_data table.
67318. **comment:** For a long-term use prepared statement avoid the use of ** lookaside memory.
label: code-design
67319. Shadow table to create (e.g. "content")
67320. ** CAPI3REF: Virtual Table Instance Object ** KEYWORDS: sqlite3_vtab ** ** Every [virtual table module] implementation uses a subclass ** of this object to describe a particular instance ** of the [virtual table]. Each subclass will ** be tailored to the specific needs of the module implementation. ** The purpose of this superclass is to define certain fields that are ** common to all module implementations. ** ** ^Virtual tables methods can set an error message by assigning a ** string obtained from [sqlite3_mprintf()] to zErrMsg. The method should ** take care that any prior string is freed by a call to [sqlite3_free()] ** prior to assigning a new string to zErrMsg. ^After the error message ** is delivered up to the client application, the string will be automatically ** freed by sqlite3_free() and the zErrMsg field will be zeroed.
67321. Scan multiple terms in the main index
67322. Outer conjunction
67323. Incremental merger
67324. Dummy parameter to exprCodeVector()
67325. #include "sqliteInt.h" ** Requires access to internal data structures **
67326. Remove the cell from the node. This call just moves bytes around ** the in-memory node image, so it cannot fail.
67327. 1350
67328. synopsis: if r[P1]!=0 then r[P1]--, goto P2
67329. The OP_OpenEphemeral for the sorting index
67330. If the parent table is the same as the child table, and we are about ** to increment the constraint-counter (i.e. this is an INSERT operation), ** then check if the row being inserted matches itself. If so, do not ** increment the constraint-counter.
67331. **comment:** Rowid for current doc being written
label: documentation
67332. same as TK_AND, in1, in2, out3
67333. 77
67334. 0xffff
67335. Memory cell used for change counting
67336. ** Extract the next token from a tokenization cursor.
67337. Used only when flags==MEM_RowSet
67338. synopsis: affinity(r[P1])
67339. If another database handle has already opened a write transaction ** on this shared-btree structure and a second write transaction is ** requested, return SQLITE_LOCKED.
67340. If there is a GROUP BY clause we might need a sorting index to ** implement it. Allocate that sorting index now. If it turns out ** that we do not need it after all, the OP_SorterOpen instruction ** will be converted into a Noop.
67341. ** The following are used as the second parameter to sqlite3Savepoint(), ** and as the P1 argument to the OP_Savepoint instruction.
67342. A record to be insert into the new table
67343. Maximum number of simultaneous paths tracked
67344. iAuxArg is always set if to a positive value if ONEPASS is possible
67345. CURTYPE_PSEUDO. Reg holding content.
67346. ** Free all allocations associated with the iterator passed as the ** second argument.
67347. ** Pragma virtual table module xFilter method.
67348. SCAN
67349. ** FTS3 and FTS4 both require virtual table support
67350. Database connection associated with this table
67351. Page number for this page
67352. **comment:** Not needed. Only to silence a warning.
label: requirement
67353. composite PRIMARY KEY value
67354. ** A "phrase" is a sequence of one or more tokens that must match in ** sequence. A single token is the base case and the most common case. ** For a sequence of tokens contained in double-quotes (i.e. "one two three") ** nToken will be the number of tokens in the string.
67355. xfer opt does not play well with PRAGMA count_changes
67356. Function arguments
67357. What to do in case of uniqueness conflict on iPKey
67358. Restriction (23)
67359. OUT: SegReader for pending-terms
67360. The journal file exists and no other connection has a reserved ** or greater lock on the database file. Now check that there is ** at least one non-zero bytes at the start of the journal file. ** If there is, then we consider this journal to be hot. If not, ** it can be ignored.
67361. PARTLY_WITHIN or FULLY_WITHIN
67362. User visible part of object (see fts5.h)
67363. ** Implementation of randomblob(N). Return a random blob ** that is N bytes long.
67364. The "rows less-than" for the rowid column must be greater than that ** for the last sample in the p->a[] array. Otherwise, the samples would ** be out of order.

67365. OR
67366. SQLITE_MUTEX_PTHREADS
67367. Used to get maximum path length and AppData
67368. For the OP_NoConflict opcode, take the jump if any of the ** input fields are NULL, since any key with a NULL will not ** conflict
67369. In-memory database
67370. EVIDENCE-OF: R-14374-42468 This option sets the threading mode to ** Multi-thread.
67371. Check to see if a partial index with pPartIndexWhere can be used ** in the current query. Return true if it can be and false if not.
67372. SQL function call context
67373. Set of contexts for which prohibited
67374. If true, discard no data
67375. Attach the hook to this db handle
67376. Function to invoke on each commit
67377. sqlite3BtreeOpen()
67378. space allocated to zToken buffer
67379. Return results here
67380. Do not do auto-vacuum
67381. OUT: Write new node image here
67382. Possible if pIn1==pIn3
67383. Must be UNION ALL
67384. ON DELETE
67385. Node ID
67386. 11
67387. selcollist
67388. Column expressions
67389. ** Initialize API pointer table, if required.
67390. Look for an existing WhereLoop to replace with pTemplate
67391. Query planner decisions affected by ** Index.aiRowLogEst[] values
67392. **comment:** *** This function - winLogErrorAtLine() - is only ever called via the macro ** winLogError(). *** This routine is invoked after an error occurs in an OS function. ** It logs a message using sqlite3_log() containing the current value of ** error code and, if possible, the human-readable equivalent from ** FormatMessage. *** The first argument passed to the macro should be the error code that ** will be returned to SQLite (e.g. SQLITE_IOERR_DELETE, SQLITE_CANTOPEN). ** The two subsequent arguments should be the name of the OS function that ** failed and the associated file-system path, if any.
label: code-design
67393. SQL statement, UTF-16 encoded
67394. IN/OUT: Increment by bytes written
67395. Index of the parameter to bind
67396. ** Return the size of an outstanding allocation, in bytes. ** This only works for chunks that are currently checked out.
67397. PGroup this cache belongs to
67398. Array of remaining idx values
67399. phrase ::= STRING star_opt
67400. ** Structure containing global configuration data for the SQLite library. *** This structure also contains some state information.
67401. No error if the table already exists
67402. ** Add pSelect to the Expr.x.pSelect field. Or, if pExpr is NULL (due ** do a memory allocation failure) then delete the pSelect object.
67403. Explicit use of the CROSS keyword
67404. Max level number for this index/langid
67405. **comment:** ** This function is called as part of the transition from PAGER_OPEN ** to PAGER_READER state to determine the size of the database file ** in pages (assuming the page size currently stored in Pager.pageSize). *** If no error occurs, SQLITE_OK is returned and the size of the database ** in pages is stored in *pnPage. Otherwise, an error code (perhaps ** SQLITE_IOERR_FSTAT) is returned and *pnPage is left unmodified.
label: code-design
67406. Pointer to list of ongoing backup processes
67407. ** A pointer to a structure of the following type is passed as the first ** argument to callbacks registered using rtree_geometry_callback().
67408. deprecated names
67409. Term to seek to
67410. FULL
67411. One of PASSIVE, FULL or RESTART
67412. **comment:** If the read is unsuccessful, set the dbFileVers[] to something ** that will never be a valid file version. dbFileVers[] is a copy ** of bytes 24..39 of the database. Bytes 28..31 should always be ** zero or the size of the database in page. Bytes 32..35 and 35..39 ** should be page numbers which are never 0xffffffff. So filling ** pPager->dbFileVers[] with all 0xff bytes should suffice. *** For an encrypted database, the situation is more complex: bytes ** 24..39 of the database are white noise. But the probability of ** white noise equaling 16 bytes of 0xff is vanishingly small so ** we should still be ok.
label: code-design
67413. Allocate memory from the gap in between the cell pointer array ** and the cell content area. The btreeInitPage() call has already ** validated the freelist. Given that the freelist is valid, there ** is no way that the allocation can extend off the end of the page. ** The assert() below verifies the previous sentence.
67414. **comment:** Allocate and initialize the WhereInfo structure that will become the ** return value. A single allocation is used to store the WhereInfo ** struct, the contents of WhereInfo.a[], the WhereClause structure ** and the WhereMaskSet structure. Since WhereClause contains an 8-byte ** field (type Bitmask) it must be aligned on an 8-byte boundary on ** some architectures. Hence the ROUND8() below.
label: code-design
67415. DELETE + INSERT
67416. beginning-of-error-codes
67417. Filename argument to pass to BtreeOpen()
67418. ** By default, allocate this many bytes of memory for each FileChunk object.
67419. Record that this module has started
67420. Number of phrases seen so far
67421. Cursor id
67422. **comment:** OUT: Pointer to unused portion of zSql
label: code-design
67423. Jump to the end of the loop if the LIMIT is reached. Except, if ** there is a sorter, in which case the sorter has already limited ** the output for us.
67424. Abort due to constraint violation
67425. AFP locking uses the file path so it needs to be included in ** the afpLockingContext.
67426. SET
67427. Pre-allocated buffer
67428. ** Return a pointer to an sqlite3_value structure containing the value bound ** parameter iVar of VM v. Except, if the value is an SQL NULL, return ** 0 instead. Unless it is NULL, apply affinity aff (one of the SQLITE_AFF_* ** constants) to the value before returning it. *** The returned value must be freed by the caller using sqlite3ValueFree().
67429. Name of the column of the table
67430. No commit or rollback needed if the program never started or if the ** SQL statement does not read or write a database file.
67431. ***** Implementation of an eponymous virtual table that runs a pragma. **
67432. Offset in zDoc of end of token
67433. 21

67434. **comment:** Check to see if another process is holding the dead-man switch. ** If not, truncate the file to zero length.
label: code-design

67435. 51

67436. ** Prepare to begin tokenizing a particular string. The input ** string to be tokenized is zInput[0..nInput-1]. A cursor ** used to incrementally tokenize this string is returned in ** ppCursor.

67437. Optional WHERE clause to be added

67438. **comment:** ** This function is an optimized version of sqlite3VdbeRecordCompare() ** that (a) the first field of pPKey2 is an integer, and (b) the ** size-of-header varint at the start of (pKey1/nKey1) fits in a single ** byte (i.e. is less than 128). ** ** To avoid concerns about buffer overreads, this routine is only used ** on schemas where the maximum valid header size is 63 bytes or less.
label: code-design

67439. ** If the argument is a codepoint corresponding to a lowercase letter ** in the ASCII range with a diacritic added, return the codepoint ** of the ASCII letter only. For example, if passed 235 - "LATIN ** SMALL LETTER E WITH DIAERESIS" - return 65 ("LATIN SMALL LETTER ** E"). The results of passing a codepoint that corresponds to an ** uppercase letter are undefined.

67440. The list of freeblocks must be in ascending order. Find the ** spot on the list where iStart should be inserted.

67441. Used by extra internal tests only run if NDEBUG is not defined

67442. True if blob 'hint' has been modified

67443. Decomposition of the entire WHERE clause

67444. Find the value object that holds the new rowid value.

67445. Check sanity of left child page for internal pages

67446. Array of "seen instance" flags

67447. Used by PRAGMA temp_store_directory

67448. ** Initialize the operating system interface. ** ** This routine registers all VFS implementations for unix-like operating ** systems. This routine, and the sqlite3_os_end() routine that follows, ** should be the only routines in this file that are visible from other ** files. ** ** This routine is called once during SQLite initialization and by a ** single thread. The memory allocation and mutex subsystems have not ** necessarily been initialized when this routine is called, and so they ** should not be used.

67449. ** The stmtConnect() method is invoked to create a new ** stmt_vtab that describes the stmt virtual table. ** ** Think of this routine as the constructor for stmt_vtab objects. ** ** All this routine needs to do is: ** ** (1) Allocate the stmt_vtab object and initialize all fields. ** ** (2) Tell SQLite (via the sqlite3_declare_vtab() interface) what the ** result set of queries against stmt will look like.

67450. SQLITE_ENABLE_LOCKING_STYLE && defined(__APPLE__)

67451. If opening a non-empty database, check the text encoding. For the ** main database, set sqlite3.enc to the encoding of the main database. ** For an attached db, it is an error if the encoding is not the same ** as sqlite3.enc.

67452. cmd ::= REINDEX

67453. ** Obtain a pointer to a mapping of a single 32KiB page of the *-shm file.

67454. Restriction (1)

67455. IMP: R-07272-22309

67456. ** Append N bytes of text from z to the StrAccum object. Increase the ** size of the memory allocation for StrAccum if necessary.

67457. Path to rbu db

67458. How often to do a periodic sample

67459. Handle negative integers in a single step. This is needed in the ** case when the value is -9223372036854775808.

67460. 6

67461. Create a single sqlite_stat1 entry containing NULL as the index ** name and the row count as the content.

67462. The expression of the term

67463. The julian day number for 9999-12-31 23:59:59.999 is 5373484.4999999. ** Multiplying this by 86400000 gives 464269060799999 as the maximum value ** for Date/Time.iJD. ** ** But some older compilers (ex: gcc 4.2.1 on older Macs) cannot deal with ** such a large integer literal, so we have to encode it.

67464. Advance pointer p1 or p2 (whichever corresponds to the smaller of ** iCol1 and iCol2) so that it points to either the 0x00 that marks the ** end of the position list, or the 0x01 that precedes the next ** column-number in the position list.

67465. OUT: Malloc'd array of column names

67466. nfs lockd on OSX 10.3+ doesn't clear write locks when a read lock is set

67467. 92

67468. These constants specify the various numeric values for terminal symbols ** in a format understandable to "makeheaders". This section is blank unless ** "lemon" is run with the "-m" command-line option. ***** Begin makeheaders token definitions *****

67469. OUT: Allocated seg-reader cursor

67470. Increase the total number of bytes written to account for the new entry.

67471. **comment:** This is an extra SQLITE_TRACE macro that indicates "legacy" tracing ** in the style of sqlite3_trace()
label: code-design

67472. ** Default span for NEAR operators.

67473. ** Allocate a two-slot MatchinfoBuffer object.

67474. Number of cursors required

67475. Multiple by log(M) where M is the number of output rows. ** Use the LIMIT for M if it is smaller

67476. This indicates that no database name was specified as part ** of the PRAGMA command. In this case the locking-mode must be ** set on all attached databases, as well as the main db file. ** ** Also, the sqlite3.dfltLockMode variable is set so that ** any subsequently attached databases also use the specified ** locking mode.

67477. Reclaim all memory used by the VDBE

67478. Discard the contents of the cache

67479. ** LOCKFILE_FAIL_IMMEDIATELY is undefined on some Windows systems.

67480. Write result here

67481. Initial number of leaf cells on trunk page

67482. Like FULL, but wait for readers

67483. **comment:** The following goto is an optimization. It can be omitted and ** everything will still work. But OP_Column is measurably faster ** by skipping the subsequent conditional, which is always true.
label: code-design

67484. Size of the rowid

67485. ~'

67486. Extracted Key value

67487. Because P2 is always a static string, it is impossible for the ** sqlite3VdbeMemCopy() to fail

67488. 0x20..0x2F

67489. Largest valid xSavepoint integer

67490. Type of node in this tree

67491. True if there is a trailing "**"

67492. True if plan uses IN(..) operator

67493. Ignore TEMP and :memory: databases

67494. The next iteration of the do-loop balances the parent page.

67495. 880

67496. Query the value of journalmode

67497. ** Return the current time for a statement. If the current time ** is requested more than once within the same run of a single prepared ** statement, the exact same time is returned for each invocation regardless ** of the amount of time that elapses between invocations. In other words, ** the time returned is always the time of the first call.

67498. ** Return the number of bytes that will be needed to store the given ** 64-bit integer.

67499. .pFiller =
67500. ** Compute and return a new Expr object which when passed to ** sqlite3ExprCode() will generate all necessary code to compute ** the iField-th column of the vector expression pVector. ** ** It is ok for pVector to be a scalar (as long as iField==0). ** In that case, this routine works like sqlite3ExprDup(). ** ** The caller owns the returned Expr object and is responsible for ** ensuring that the returned value eventually gets freed. ** ** The caller retains ownership of pVector. If pVector is a TK_SELECT, ** then the returned object will reference pVector and so pVector must remain ** valid for the life of the returned object. If pVector is a TK_VECTOR ** or a scalar expression, then it can be deleted as soon as this routine ** returns. ** ** A trick to cause a TK_SELECT pVector to be deleted together with ** the returned Expr object is to attach the pVector to the pRight field ** of the returned TK_SELECT_COLUMN Expr object.
67501. reg[1] holds errors left
67502. ** Interchange two search points in a cursor.
67503. ccons ::= PRIMARY KEY sortorder onconf autoinc
67504. Search backwards for a varint with value zero (the end of the previous ** poslist). This is an 0x00 byte preceded by some byte that does not ** have the 0x80 bit set.
67505. ** If the argument p points to a MemJournal structure that is not an ** in-memory-only journal file (i.e. is one that was opened with a +ve ** nSpill parameter), and the underlying file has not yet been created, ** create it now.
67506. Save the positions of any other cursors open on this table. ** ** In some cases, the call to btreeMoveto() below is a no-op. For ** example, when inserting data into a table with auto-generated integer ** keys, the VDBE layer invokes sqlite3BtreeLast() to figure out the ** integer key to use. It then calls this function to actually insert the ** data into the intkey B-Tree. In this case btreeMoveto() recognizes ** that the cursor is already where it needs to be and returns without ** doing any work. To avoid thwarting these optimizations, it is important ** not to clear the cursor here.
67507. All fields above are zeroed when the cursor is allocated. See ** sqlite3BtreeCursorZero(). Fields that follow must be manually ** initialized.
67508. This loop terminates either when a readJournalHdr() or ** pager_playback_one_page() call returns SQLITE_DONE or an IO error ** occurs.
67509. Error state.
67510. No overflow pages. Return without doing anything
67511. this happens for a comment or white-space
67512. ** Return the ON CONFLICT resolution mode in effect for the virtual ** table update operation currently in progress. ** ** The results of this routine are undefined unless it is called from ** within an xUpdate method.
67513. 247
67514. SQLITE_ZERO_MALLOC
67515. **comment:** Set the output variable to NULL in case an error occurs.
label: code-design
67516. Collating sequence with native encoding, or NULL
67517. The following three functions, heightOfExpr(), heightOfExprList() ** and heightOfSelect(), are used to determine the maximum height ** of any expression tree referenced by the structure passed as the ** first argument. ** ** If this maximum height is greater than the current value pointed ** to by pnHeight, the second parameter, then set *pnHeight to that ** value.
67518. Restriction (21)
67519. Loop counters
67520. The "wsdHooks" macro will resolve to the appropriate BenignMallocHooks ** structure. If writable static data is unsupported on the target, ** we have to locate the state vector at run-time. In the more common ** case where writable static data is supported, wsHooks can refer directly ** to the "sqlite3Hooks" state vector declared above.
67521. First call xBestIndex() with all constraints usable.
67522. Index of docid>=x constraint, if present
67523. Size of allocation at aData
67524. ** CAPI3REF: OS Interface Open File Handle ** ** An [sqlite3_file] object represents an open file in the ** [sqlite3_vfs | OS interface layer]. Individual OS interface ** implementations will ** want to subclass this object by appending additional fields ** for their own use. The pMethods entry is a pointer to an ** [sqlite3_io_methods] object that defines methods for performing ** I/O operations on the open file.
67525. SQL used to determine iAbsLevel
67526. ** A VdbeCursor is an superclass (a wrapper) for various cursor objects: ** ** * A b-tree cursor ** - In the main database or in an ephemeral database ** - On either an index or a table ** * A sorter ** * A virtual table ** * A one-row "pseudotable" stored in a single register
67527. * The extra flags to use in calls to the Win32 heap APIs. This value may be * zero for the default behavior.
67528. ** The xConnect() and xCreate() methods for the virtual table. All the ** work is done in function fts5VocabInitVtab().
67529. List of operands to the LIKE operator
67530. It is not possible to create a database for which the final page ** is either a pointer-map page or the pending-byte page. If one ** is encountered, this indicates corruption.
67531. ** Check if the *-wal file that corresponds to the database opened by pPager ** exists if the database is not empty, or verify that the *-wal file does ** not exist (by deleting it) if the database file is empty. ** ** If the database is not empty and the *-wal file exists, open the pager ** in WAL mode. If the database is empty or if no *-wal file exists and ** if no error occurs, make sure Pager.journalMode is not set to ** PAGER_JOURNALMODE_WAL. ** ** Return SQLITE_OK or an error code. ** ** The caller must hold a SHARED lock on the database file to call this ** function. Because an EXCLUSIVE lock on the db file is required to delete ** a WAL on a non-empty database, this ensures there is no race condition ** between the xAccess() below and an xDelete() being executed by some ** other connection.
67532. ** Return the search point with the lowest current score.
67533. ***** Begin file vdbe.c *****
67534. The bottom of the loop
67535. ** CAPI3REF: Finding The Subtype Of SQL Values ** METHOD: sqlite3_value ** ** The sqlite3_value_subtype(V) function returns the subtype for ** an [application-defined SQL function] argument V. The subtype ** information can be used to pass a limited amount of context from ** one SQL function to another. Use the [sqlite3_result_subtype()] ** routine to set the subtype for the return value of an SQL function.
67536. Offset into aKey[] of next header element
67537. Any read-only or read-write transaction implies a read-lock on ** page 1. So if some other shared-cache client already has a write-lock ** on page 1, the transaction cannot be opened.
67538. ** The token *pName contains the name of a database (either "main" or ** "temp" or the name of an attached db). This routine returns the ** index of the named database in db->aDb[], or -1 if the named db ** does not exist.
67539. Locate the end of the CREATE VIEW statement. Make sEnd point to ** the end.
67540. Otherwise useTempTable is true
67541. 1 for the "l" flag, 2 for "ll", 0 by default
67542. Delete a row from a main table b-tree
67543. List of all unixInodeInfo objects
67544. Also store the SQLite version number in bytes 96..99 and in ** bytes 92..95 store the change counter for which the version number ** is valid.
67545. If this is an insert into a table b-tree, invalidate any incrblob ** cursors open on the row being replaced
67546. **comment:** For temporary filename, if necessary.
label: code-design
67547. ** If a write transaction is open, then all changes made within the ** transaction are reverted and the current write-transaction is closed. ** The pager falls back to PAGER_READER state if successful, or PAGER_ERROR ** state if an error occurs. ** ** If the pager is already in PAGER_ERROR state when this function is called, ** it returns Pager.errCode immediately. No work is performed in this case. ** ** Otherwise, in rollback mode, this function performs two functions: ** ** 1) It rolls back the journal file, restoring all database file and ** in-memory cache pages to the state they were in when the transaction ** was opened, and ** ** 2) It finalizes the journal file, so that it is not used for hot ** rollback at any point in the future. ** ** Finalization of the journal file (task 2) is only performed if the ** rollback is successful. ** ** In WAL mode, all cache-entries containing data modified within the ** current transaction are either expelled from the cache or reverted to ** their pre-transaction state by re-reading data from the database or ** WAL files. The WAL transaction is then closed.
67548. .enc =
67549. LIKE pattern

67550. At first glance you would think we could optimize out the ** ORDER BY in this case since the order of entries in the set ** does not matter. But there might be a LIMIT clause, in which ** case the order does matter

67551. LHS values prior to reordering by aiMap[]

67552. Cursor of table to get the new rowid

67553. Has no GROUP BY clause

67554. Number of bytes of prefix compression

67555. !defined(SQLITE_OMIT_TRIGGER)

67556. The three function arguments

67557. Partial index inappropriate for this query

67558. Jump to the appropriate pragma handler

67559. ** The following are allowed values for Vdbe.magic

67560. Use native byte order

67561. Column index on lhs of IN operator

67562. ** Look up an index by name. Or, if the name of a WITHOUT ROWID table ** is supplied instead, find the PRIMARY KEY index for that table.

67563. SQLite assumes that xFullPathname() nul-terminates the output buffer ** even if it returns an error.

67564. Table holding the result of the SELECT

67565. ** The most recently coded instruction was an OP_Column to retrieve the ** i-th column of table pTab. This routine sets the P4 parameter of the ** OP_Column to the default value, if any. ** ** The default value of a column is specified by a DEFAULT clause in the ** column definition. This was either supplied by the user when the table ** was created, or added later to the table definition by an ALTER TABLE ** command. If the latter, then the row-records in the table btree on disk ** may not contain a value for the column and the default value, taken ** from the P4 parameter of the OP_Column instruction, is returned instead. ** If the former, then all row-records are guaranteed to include a value ** for the column and the P4 value is not required. ** ** Column definitions created by an ALTER TABLE command may only have ** literal default values specified: a number, null or a string. (If a more ** complicated default expression value was provided, it is evaluated ** when the ALTER TABLE is executed and one of the literal values written ** into the sqlite_master table.) ** ** Therefore, the P4 parameter is only required if the default value for ** the column is a literal number, string or null. The sqlite3ValueFromExpr() ** function is capable of transforming these types of expressions into ** sqlite3_value objects. ** ** If parameter iReg is not negative, code an OP_RealAffinity instruction ** on register iReg. This is used when an equivalent integer value is ** stored in place of an 8-byte floating point value in order to save ** space.

67566. Ticket 2ea2425d34be

67567. ** The following macro defines an initializer for an sqlite3_vfs object. ** The name of the VFS is NAME. The pAppData is a pointer to a pointer ** to the "finder" function. (pAppData is a pointer to a pointer because ** silly C90 rules prohibit a void* from being cast to a function pointer ** and so we have to go through the intermediate pointer to avoid problems ** when compiling with -pedantic-errors on GCC.) ** ** The FINDERT parameter to this macro is the name of the pointer to the ** finder-function. The finder-function returns a pointer to the ** sqlite_io_methods object that implements the desired locking ** behaviors. See the division above that contains the IOMETHODS ** macro for addition information on finder-functions. ** ** Most finders simply return a pointer to a fixed sqlite3_io_methods ** object. But the "autolockIoFinder" available on MacOSX does a little ** more than that; it looks at the filesystem type that hosts the ** database file and tries to choose an locking method appropriate for ** that filesystem time.

67568. Number of significant digits returned

67569. True after mutexes are initialized

67570. Index of the first hidden column

67571. Define the yytestcase() macro to be a no-op if is not already defined ** otherwise. ** ** Applications can choose to define yytestcase() in the %include section ** to a macro that can assist in verifying code coverage. For production ** code the yytestcase() macro should be turned off. But it is useful ** for testing.

67572. **comment:** Smallest sample not yet tested
label: test

67573. 0x10..0x1F

67574. ** The maximum length of a TEXT or BLOB in bytes. This also ** limits the size of a row in a table or index. ** ** The hard limit is the ability of a 32-bit signed integer ** to count the size: 2^31-1 or 2147483647.

67575. Length in bytes of zFrom

67576. 920

67577. Manufactured x>NULL or x<=NULL term

67578. if we succeeded in taking the reserved lock, unlock it to restore ** the original state

67579. The assumed sector size for this process

67580. **comment:** ** Argument zFmt is a sqlite3_mprintf() style format string. The trailing ** arguments are the usual subsitution values. This function performs ** the printf() style substitutions and executes the result as an SQL ** statement on the RBU handles database. ** ** If an error occurs, an error code and error message is stored in the ** RBU handle. If an error has already occurred when this function is ** called, it is a no-op.
label: code-design

67581. Top of the index fill loop

67582. Size of prefix shared with previous term

67583. Analyze the min-heap looking for overlap between cells and/or ** freeblocks, and counting the number of untracked bytes in nFrag. ** ** Each min-heap entry is of the form: (start_address<<16)|end_address. ** There is an implied first entry the covers the page header, the cell ** pointer index, and the gap between the cell pointer index and the start ** of cell content. ** ** The loop below pulls entries from the min-heap in order and compares ** the start_address against the previous end_address. If there is an ** overlap, that means bytes are used multiple times. If there is a gap, ** that gap is added to the fragmentation count.

67584. groupby_opt ::= GROUP BY nxprlist

67585. Find the sibling pages to balance. Also locate the cells in pParent ** that divide the siblings. An attempt is made to find NN siblings on ** either side of pPage. More siblings are taken from one side, however, ** if there are fewer than NN siblings on the other side. If pParent ** has NB or fewer children then all children of pParent are taken. ** ** This loop also drops the divider cells from the parent page. This ** way, the remainder of the function does not have to deal with any ** overflow cells in the parent page, since if any existed they will ** have already been removed.

67586. Pre-allocated UnixUnusedFd

67587. Database was empty at trans start

67588. Because pFile->pInode is shared across threads

67589. Increment the "number of sequential leaves without a term" counter.

67590. assert(p->id==GetCurrentThreadId());

67591. ** Additional non-public SQLITE_PREPARE_* flags

67592. xCheckReservedLock

67593. Token text (not NULL terminated)

67594. ** Return UTF-16 encoded English language explanation of the most recent ** error.

67595. Default values must be the same for all columns

67596. ***** Begin file sqliteLimit.h *****

67597. Content of OLD.* table in triggers

67598. SQL statement, UTF-8 encoded

67599. unixInodeInfo that owns this SHM node

67600. Allowed flags for sqlite3BtreeDelete() and sqlite3BtreeInsert()

67601. Substitute the rowid (column -1) for the INTEGER PRIMARY KEY

67602. Locate the table which we want to update.

67603. ** Make a transient copy of expression pExpr and then code it using ** sqlite3ExprCode(). This routine works just like sqlite3ExprCode() ** except that the input expression is guaranteed to be unchanged.

67604. An INDEXED BY clause specifies a particular index to use

67605. ** Prepare the SQL statement in buffer zSql against database handle db. ** If successful, set *ppStmt to point to the new statement and return ** SQLITE_OK. ** ** Otherwise, if an error does occur, set *ppStmt to NULL and return ** an SQLite error code. Additionally, set output variable *pzerrmsg to ** point to a buffer containing an error message. It is the responsibility ** of the caller to (eventually) free this buffer using sqlite3_free().

67606. ** The interface to the LEMON-generated parser

67607. ** The "context" argument for an installable function. A pointer to an ** instance of this structure is the first argument to the routines used ** implement the SQL functions. *** There is a typedef for this structure in sqlite.h. So all routines, ** even the public interface to SQLite, can use a pointer to this structure. ** But this file is the only place where the internal details of this ** structure are known. *** This structure is defined inside of vdbeInt.h because it uses substructures ** (Mem) which are only defined there.

67608. ***** End Single-Threaded *****

67609. xUpdate

67610. **comment:** ** Temporary files are named starting with this prefix followed by 16 random ** alphanumeric characters, and no file extension. They are stored in the ** OS's standard temporary file directory, and are deleted prior to exit. ** If sqlite is being embedded in another program, you may wish to change the ** prefix to reflect your program's name, so that if your program exits ** prematurely, old temporary files can be easily identified. This can be done ** using -DSQLITE_TEMP_FILE_PREFIX=myprefix_ on the compiler command line. *** 2006-10-31: The default prefix used to be "sqlite_". But then ** McAfee started using SQLite in their anti-virus product and it ** started putting files with the "sqlite" name in the c:/temp folder. ** This annoyed many windows users. Those users would then do a ** Google search for "sqlite", find the telephone numbers of the ** developers and call to wake them up at night and complain. ** For this reason, the default name prefix is changed to be "sqlite" ** spelled backwards. So the temp files are still identified, but ** anybody smart enough to figure out the code is also likely smart ** enough to know that calling the developer will not help get rid ** of the file.

label: code-design

67611. ** Meanings of bits in of pWalker->eCode for checkConstraintUnchanged()

67612. Add "LIMIT -1 OFFSET \$nOffset" to SELECT

67613. Device characteristics

67614. Name of this virtual file system

67615. Uses an ephemeral index

67616. Obtain the page from this cache

67617. Record the number of times that we do a normal fsync() and ** FULLSYNC. This is used during testing to verify that this procedure ** gets called with the correct arguments.

67618. Number of rows in table

67619. The entry into which pNew is inserted

67620. SetEndOfFile() returns non-zero when successful, or zero when it fails.

67621. Key value (page number)

67622. Number of slots allocated for Vdbe.aOp[]

67623. ** Create a new sqlite3_value object.

67624. Name of file to delete when closing

67625. Database holding shared memory

67626. Size increases

67627. Gather the schema version number for checking: ** IMPLEMENTATION-OF: R-03189-51135 As each SQL statement runs, the schema ** version is checked to ensure that the schema has not changed since the ** SQL statement was prepared.

67628. The requested region is not mapped into this process's address space. ** Check to see if it has been allocated (i.e. if the wal-index file is ** large enough to contain the requested region).

67629. String inserted after highlighted term

67630. extended error code

67631. Check the number of columns in this xPreUpdate call matches the ** number of columns in the table.

67632. Expression to return via *ppNew

67633. Next on the Parse.pZombieTab list

67634. ** This routine is a copy of the sqlite3FileSuffix3() routine from the core. ** It is a no-op unless SQLITE_ENABLE_8_3_NAMES is defined. *** If SQLITE_ENABLE_8_3_NAMES is set at compile-time and if the database ** filename in zBaseFilename is a URI with the "8_3_names=1" parameter and ** if filename in z[] has a suffix (a.k.a. "extension") that is longer than ** three characters, then shorten the suffix on z[] to be the last three ** characters of the original suffix. *** If SQLITE_ENABLE_8_3_NAMES is set to 2 at compile-time, then always ** do the suffix shortening regardless of URI parameter. *** Examples: *** test.db-journal => test.nal ** test.db-wal => test.wal ** test.db-shm => test.shm ** test.db-mj7f3319fa => test.9fa

67635. Use index only - omit table

67636. First byte of available memory space

67637. synopsis: intkey=r[P3]

67638. ** Append a function parameter value to the JSON string under ** construction.

67639. term sans prefix-byte

67640. Start at the inner-most context and move outward until a match is found

67641. Set to true when merge is finished

67642. True if this is a TEMP table

67643. D

67644. ** Maximum nesting depth of JSON for this implementation. *** This limit is needed to avoid a stack overflow in the recursive ** descent parser. A depth of 2000 is far deeper than any sane JSON ** should go.

67645. Round the union size down to the nearest pointer boundary, since that's how ** it will be aligned within the Bitvec struct.

67646. ** xRowid - Return the current rowid for the cursor.

67647. ** Run a checkpoint on database iDb. This is a no-op if database iDb is ** not currently open in WAL mode. *** If a transaction is open on the database being checkpointed, this ** function returns SQLITE_LOCKED and a checkpoint is not attempted. If ** an error occurs while running the checkpoint, an SQLite error code is ** returned (i.e. SQLITE_IOERR). Otherwise, SQLITE_OK. *** The mutex on database handle db should be held by the caller. The mutex ** associated with the specific b-tree being checkpointed is taken by ** this function while the checkpoint is running. *** If iDb is passed SQLITE_MAX_ATTACHED, then all attached databases are ** checkpointed. If an error is encountered it is returned immediately - ** no attempt is made to checkpoint any remaining databases. *** Parameter eMode is one of SQLITE_CHECKPOINT_PASSIVE, FULL or RESTART.

67648. FuncDef wrapper for detachFunc() or attachFunc()

67649. #include <stddef.h>

67650. Number of remaining savepoints after this op.

67651. ** Start a statement subtransaction. The subtransaction can be rolled ** back independently of the main transaction. You must start a transaction ** before starting a subtransaction. The subtransaction is ended automatically ** if the main transaction commits or rolls back. *** Statement subtransactions are used around individual SQL statements ** that are contained within a BEGIN...COMMIT block. If a constraint ** error occurs within the statement, the effect of that one statement ** can be rolled back without having to rollback the entire transaction. *** A statement sub-transaction is implemented as an anonymous savepoint. The ** value passed as the second parameter is the total number of savepoints, ** including the new anonymous savepoint, open on the B-Tree. i.e. if there ** are no active savepoints and no other statement-transactions open, ** iStatement is 1. This anonymous savepoint can be released or rolled back ** using the sqlite3BtreeSavepoint() function.

67652. Whenever Mem contains a valid string or blob representation, one of ** the following flags must be set to determine the memory management ** policy for Mem.z. The MEM_Term flag tells us whether or not the ** string is \000 or \u0000 terminated

67653. ** Systems that support the isnan() library function should probably ** make use of it by compiling with -DSQLITE_HAVE_ISNAN. But we have ** found that many systems do not have a working isnan() function so ** this implementation is provided as an alternative. *** This NaN test sometimes fails if compiled on GCC with -ffast-math. ** On the other hand, the use of -ffast-math comes with the following ** warning: *** This option [-ffast-math] should never be turned on by any ** -O option since it can result in incorrect output for programs ** which depend on an exact implementation of IEEE or ISO ** rules/specifications for math functions. *** Under MSVC, this NaN test may fail if compiled with a floating- ** point precision mode other than /fp:precise. From the MSDN ** documentation: *** The compiler [with /fp:precise] will properly handle comparisons ** involving NaN. For example, x != x evaluates to true if x is NaN ** ...

67654. ** 2005 May 25 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the implementation of the sqlite3_prepare() ** interface, and routines that contribute to loading the database schema ** from disk.

67655. with ::=

67656. Opcode: Affinity P1 P2 * P4 * *** Synopsis: affinity(r[P1@P2]) *** Apply affinities to a range of P2 registers starting with P1. *** P4 is a string that is P2 characters long. The N-th character of the ** string indicates the column affinity that should be used for the N-th ** memory cell in the range.

67657. Only mark the value as an integer if *** (1) the round-trip conversion real->int->real is a no-op, and *** (2) The integer is neither the largest nor the smallest ** possible integer (ticket #3922) *** The second and third terms in the following conditional enforces ** the second condition under the assumption that addition overflow causes ** values to wrap around.

67658. 143

67659. ** Obtain a lock on the table whose root page is iTab. The ** lock is a write lock if isWritelock is true or a read lock ** if it is false.

67660. Type code for object to destroy

67661. Catch the case where the destination is an in-memory database and the ** page sizes of the source and destination differ.

67662. Result accumulator

67663. **comment:** The following "ifdef/elif/else/" block has the same structure as ** the one below. It is replicated here solely to avoid cluttering ** up the real code with the UNUSED_PARAMETER() macros.
label: code-design

67664. Iterator to seek

67665. Locale identifier - (eg. "jp_JP")

67666. ** This function is designed to be called from within a pre-update callback ** only. It returns zero if the change that caused the callback was made ** immediately by a user SQL statement. Or, if the change was made by a ** trigger program, it returns the number of trigger programs currently ** on the stack (1 for a top-level trigger, 2 for a trigger fired by a ** top-level trigger etc.). *** For the purposes of the previous paragraph, a foreign key CASCADE, SET NULL ** or SET DEFAULT action is considered a trigger.

67667. Calculate the first block to use in the output segment

67668. An SQL statement being run

67669. All tables in the FROM clause

67670. Number of table columns

67671. Saved value of db->nTotalChange

67672. Faults are not injected into COMMIT_PASETWO because, assuming SQLite ** is using a regular VFS, it is called after the corresponding ** transaction has been committed. Injecting a fault at this point ** confuses the test scripts - the COMMIT command returns SQLITE_NOMEM ** but the transaction is committed anyway. *** The core must call OsFileControl() though, not OsFileControlHint(), ** as if a custom VFS (e.g. zipvfs) returns an error here, it probably ** means the commit really has failed and an error should be returned ** to the user.

67673. Positions cannot be negative; we use -1 as a terminator internally. ** Tokens must have a non-zero length.

67674. Check to see if flattening is permitted. Return 0 if not.

67675. Ignore the MSVC warning about no initializer

67676. If true, bias the search to the high end

67677. Pointer to data for pPg

67678. ** 2008 Jan 22 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code to support the concept of "benign" **
malloc failures (when the xMalloc() or xRealloc() method of the ** sqlite3_mem_methods structure fails to allocate a block of memory ** and returns 0). *** **
Most malloc failures are non-benign. After they occur, SQLite ** abandons the current operation and returns an error code (usually ** SQLITE_NOMEM) to the user. However, sometimes a fault is not necessarily ** fatal. For example, if a malloc fails while resizing a hash table, this ** is completely recoverable simply by not carrying out the resize. The ** hash table will continue to function normally. So a malloc failure ** during a hash table resize is a benign fault.

67679. Number of open cursors

67680. Need to free the WhereTerm.u.pOrInfo object

67681. Current value of the system call

67682. xUnfetch

67683. **comment:** Bytes of memory needed
label: code-design

67684. Append a new column value, if necessary

67685. ** On systems with ample stack space and that support alloca(), make ** use of alloca() to obtain space for large automatic objects. By default, ** obtain space from malloc(). *** ** The alloca() routine never returns NULL. This will cause code paths ** that deal with sqlite3StackAlloc() failures to be unreachable.

67686. SQLITE_CONSTRAINT

67687. refargs ::= refargs refarg

67688. ** This routine does the first phase of a two-phase commit. This routine ** causes a rollback journal to be created (if it does not already exist) ** and populated with enough information so that if a power loss occurs ** the database can be restored to its original state by playing back ** the journal. Then the contents of the journal are flushed out to ** the disk. After the journal is safely on oxide, the changes to the ** database are written into the database file and flushed to oxide. ** At the end of this call, the rollback journal still exists on the ** disk and we are still holding all locks, so the transaction has not ** committed. See sqlite3BtreeCommitPhaseTwo() for the second phase of the ** commit process. *** ** This call is a no-op if no write-transaction is currently active on pBt. *** ** Otherwise, sync the database file for the btree pBt. zMaster points to ** the name of a master journal file that should be written into the ** individual journal file, or is NULL, indicating no master journal file ** (single database transaction). *** ** When this is called, the master journal should already have been ** created, populated with this journal pointer and synced to disk. *** ** Once this is routine has returned, the only thing required to commit ** the write-transaction for this database file is to delete the journal.

67689. Maximum allowed size of the database

67690. Size in bytes

67691. ***** End of global.c *****

67692. Add the "COLLATE" clause to this expression

67693. Non-zero to append " AND "

67694. pNew->aFirst[]

67695. Column to extract

67696. **comment:** The busy callback
label: requirement

67697. **comment:** Number of unused pcache slots
label: code-design

67698. Duplicate of trigger WHEN expression

67699. Jump here if LHS is not contained in the RHS

67700. Index of aTree[] to recalculate

67701. Cursor open on the btree to be searched

67702. Unreachable

67703. Cost of running this subquery

67704. Unlock-notify cb to invoke

67705. Information about an AUTOINCREMENT

67706. **comment:** ** This version of balance() handles the common special case where ** a new entry is being inserted on the extreme right-end of the ** tree, in other words, when the new entry will become the largest ** entry in the tree. *** ** Instead of trying to balance the 3 right-most leaf pages, just add ** a new page to the right-hand side and put the one new entry in ** that page. This leaves the right side of the tree somewhat ** unbalanced. But odds are that we will be inserting new entries ** at the end soon afterwards so the nearly empty page will quickly ** fill up. On average. *** ** pPage is the leaf page which is the right-most page in the tree. ** pParent is its parent. pPage must have a single overflow entry ** which is also the right-most entry on the page. *** ** The pSpace buffer is used to store a temporary copy of the divider ** cell that will be inserted into pParent. Such a cell consists of a 4 ** byte page number followed by a variable length integer. In other ** words, at most 13 bytes. Hence the pSpace buffer must be at ** least 13 bytes in size.
label: code-design

67707. RHS always has 2 or more terms... The parser ** changes "x IN (?)" into "x=?".

67708. ** The set of routines that implement the porter-stemmer tokenizer
67709. The page data
67710. Reset the return code so as not to report a checkpoint failure ** just because there are active readers.
67711. Open existing file, or create if it doesn't exist
67712. ** Add an INDEXED BY or NOT INDEXED clause to the most recently added ** element of the source-list passed as the second argument.
67713. ** The sqlite3_strglob() interface. Return 0 on a match (like strcmp()) and ** non-zero if there is no match.
67714. Destination journal mode
67715. Sorter for "ORDER BY rank" queries
67716. Number of non-transaction savepoints
67717. Variable iNewStart now contains the first valid leaf node.
67718. Number of rows in the table
67719. Value to append to data
67720. Call the parser to process a CREATE TABLE, INDEX or VIEW. ** But because db->init.busy is set to 1, no VDBE code is generated ** or executed. All the parser does is build the internal data ** structures that describe the table, index, or view.
67721. The underlying database file
67722. **comment:** FIXME: If Windows truly always prevents truncating or deleting a ** file while a mapping is held, then the following winUnmapfile() call ** is unnecessary can be omitted - potentially improving ** performance.
 label: code-design
67723. 234
67724. Largest block id written to database
67725. ** Free all resources associated with the IncrMerger object indicated by ** the first argument.
67726. Add any overflow cells
67727. ** Allowed values for Index.idxType
67728. Opcode: IsNull P1 P2 * * * * Synopsis: if r[P1]==NULL goto P2 ** * * Jump to P2 if the value in register P1 is NULL.
67729. The results of parsing supported FTS4 key=value options:
67730. But not in this one
67731. ** Given the page number of an overflow page in the database (parameter ** ovfl), this function finds the page number of the next page in the ** linked list of overflow pages. If possible, it uses the auto-vacuum ** pointer-map data instead of reading the content of page ovfl to do so. ** * * If an error occurs an SQLite error code is returned. Otherwise: ** * * The page number of the next overflow page in the linked list is ** written to *pPgnoNext. If page ovfl is the last page in its linked ** list, *pPgnoNext is set to zero. ** * * If ppPage is not NULL, and a reference to the MemPage object corresponding ** to page number pOvfl was obtained, then *ppPage is set to point to that ** reference. It is the responsibility of the caller to call releasePage() ** on *ppPage to free the reference. In no reference was obtained (because ** the pointer-map was used to obtain the value for *pPgnoNext), then ** *ppPage is set to zero.
67732. The replacement string C
67733. Current innermost WITH clause
67734. Database handle doing sort
67735. New tokenizer object
67736. 0 means "unknown at compile-time"
67737. ** The following are special cases for mutex enter routines for use ** in single threaded applications that use shared cache. Except for ** these two routines, all mutex operations are no-ops in that case and ** are null #defines in btree.h. ** * * If shared cache is disabled, then all btree mutex routines, including ** the ones below, are no-ops and are null #defines in btree.h.
67738. Compiled "PRAGMA %Q.page_size" statement
67739. Dummy argument used with xNext()
67740. Number of subqueries
67741. Restriction (14)
67742. Largest aReadMark[] value
67743. This module is only called on query plans that use an index.
67744. As it happens, the pending terms table is always empty here. This is ** because an "ALTER TABLE RENAME TABLE" statement inside a transaction ** always opens a savepoint transaction. And the xSavepoint() method ** flushes the pending terms table. But leave the (no-op) call to ** PendingTermsFlush() in in case that changes.
67745. Value of 'unused' column
67746. If none of the above worked, then we fail.
67747. Add the entry to the stat1 table.
67748. Expected number of pages in the list
67749. Bit values for the JsonNode.jnFlag field
67750. ** WinCE lacks native support for file locking so we have to fake it ** with some code of our own.
67751. ** Compute the column names for a SELECT statement. ** * * The only guarantee that SQLite makes about column names is that if the ** column has an AS clause assigning it a name, that will be the name used. ** That is the only documented guarantee. However, countless applications ** developed over the years have made baseless assumptions about column names ** and will break if those assumptions changes. Hence, use extreme caution ** when modifying this routine to avoid breaking legacy. ** * * See Also: sqlite3ColumnsFromExprList() ** * * The PRAGMA short_column_names and PRAGMA full_column_names settings are ** deprecated. The default setting is short=ON, full=OFF. 99.9% of all ** applications should operate this way. Nevertheless, we need to support the ** other modes for legacy: ** * * short=OFF, full=OFF: Column name is the text of the expression has it ** originally appears in the SELECT statement. In ** other words, the zSpan of the result expression. ** * * short=ON, full=OFF: (This is the default setting). If the result ** refers directly to a table column, then the result ** column name is just the table column name: COLUMN. ** Otherwise use zSpan. ** * * full=ON, short=ANY: If the result refers directly to a table column, ** then the result column name with the table name ** prefix, ex: TABLE.COLUMN. Otherwise use zSpan.
67752. ** PRAGMA [schema].journal_mode ** PRAGMA [schema].journal_mode = ** (delete|persist|off|truncate|memory|wal|off)
67753. A list of all open cursors
67754. The next table maps tokens (terminal symbols) into fallback tokens. ** If a construct like the following: ** * * %fallback ID X Y Z. ** * * appears in the grammar, then ID becomes a fallback token for X, Y, ** and Z. Whenever one of the tokens X, Y, or Z is input to the parser ** but it does not parse, the type of the token is changed to ID and ** the parse is retried before an error is thrown. ** * * This feature can be used, for example, to cause some keywords in a language ** to revert to identifiers if they keyword does not apply in the context where ** it appears.
67755. ** Token types used by the sqlite3_complete() routine. See the header ** comments on that procedure for additional information.
67756. Jump here to continue with next record
67757. Add the "eNNN" suffix
67758. Pointer type when MEM_Term|MEM_Subtype|MEM_Null
67759. Register holding an index key
67760. Number of pages still on the free-list
67761. The VDBE cursor for the table
67762. One byte past the end of usable data
67763. A view
67764. Dummy argument used with tokenizer
67765. Buffer to load data from wal file into
67766. ** Return the length of the data corresponding to the supplied serial-type.
67767. True if statement may throw an ABORT exception
67768. Remove the node from the in-memory hash table and link it into ** the Rtree.pDeleted list. Its contents will be re-inserted later on.
67769. Number of elements in argv[]
67770. Pointer to write buffer
67771. Current offset within aInput[]

67772. ** The default MMAP_SIZE is zero on all platforms. Or, even if a larger ** default MMAP_SIZE is specified at compile-time, make sure that it does ** not exceed the maximum mmap size.

67773. Called during startup to register a UDF with SQLite

67774. deadman switch

67775. True to use xFetch()

67776. Token pList/nList corresponds to

67777. Opcode: VCreate P1 P2 * * * * * P2 is a register that holds the name of a virtual table in database ** P1. Call the xCreate method for that table.

67778. Index structure corresponding to u1.zIndexedBy

67779. Database Name Table Name

67780. First argument passed to xFilter

67781. The whole SELECT statement

67782. Page iter.iLeaf must now be the rightmost leaf-page in the segment

67783. ** Return the number of cursors open on pBt. This is for use ** in assert() expressions, so it is only compiled if NDEBUG is not ** defined. ** * Only write cursors are counted if wrOnly is true. If wrOnly is ** false then all cursors are counted. ** * For the purposes of this routine, a cursor is any cursor that ** is capable of reading or writing to the database. Cursors that ** have been tripped into the CURSORFAULT state are not counted.

67784. Opcode: IdxRowid P1 P2 * * * * * Synopsis: r[P2]=rowid ** * Write into register P2 an integer which is the last entry in the record at ** the end of the index key pointed to by cursor P1. This integer should be ** the rowid of the table entry to which this index entry points. ** * See also: Rowid, MakeRecord.

67785. Number of fragments in this snippet

67786. FROM clause does not contain a real table

67787. ** An object of this type is used internally as an abstraction for ** input data. Input data may be supplied either as a single large buffer ** (e.g. sqlite3changeset_start()) or using a stream function (e.g. ** sqlite3changeset_start_strm()).

67788. (316) foreach_clause ::= FOR EACH ROW

67789. The "real" underlying VFS

67790. This is either a full-table scan (ePlan==FTS5_PLAN_SCAN) or a lookup ** by rowid (ePlan==FTS5_PLAN_ROWID).

67791. In this case, the RHS is the ROWID of table b-tree and so we also ** know that the RHS is non-NULL. Hence, we combine steps 3 and 4 ** into a single opcode.

67792. One or both operands are NULL

67793. Name of the file containing the BTree database

67794. pParent->pLeft==this or pParent->pRight==this

67795. ** Bits that make up the "idxNum" parameter passed indirectly by ** xBestIndex() to xFilter().

67796. **comment:** ** A version of printf() that understands %lld. Used for debugging. ** The printf() built into some versions of windows does not understand %lld ** and segfaults if you give it a long long int.

label: code-design

67797. One for every 32KB page in the wal-index

67798. At least one of the two values is a number

67799. ** Return the total number of segments in index structure pStruct. This ** function is only ever used as part of assert() conditions.

67800. Obscure case for non-leaf-data trees: If the cell at pCell was ** previously stored on a leaf node, and its reported size was 4 ** bytes, then it may actually be smaller than this ** (see btreeParseCellPtr(), 4 bytes is the minimum size of ** any cell). But it is important to pass the correct size to ** insertCell(), so reparse the cell now. ** * This can only happen for b-trees used to evaluate "IN (SELECT ...)" ** and WITHOUT ROWID tables with exactly one column which is the ** primary key.

67801. FTS Table pointer

67802. The last prior entry

67803. ** Load the sqlite3.iSysErrno field if that is an appropriate thing ** to do based on the SQLite error code in rc.

67804. 294

67805. EVIDENCE-OF: R-08404-60887 There are three arguments to ** SQLITE_CONFIG_SCRATCH: A pointer an 8-byte aligned memory buffer from ** which the scratch allocations will be drawn, the size of each scratch ** allocation (sz), and the maximum number of scratch allocations (N).

67806. **comment:** The following macro is used to suppress compiler warnings.

label: code-design

67807. Size of the database file

67808. Information about this WHERE

67809. ** Given that zWal points to a buffer containing a wal file name passed to ** either the xOpen() or xAccess() VFS method, return a pointer to the ** file-handle opened by the same database connection on the corresponding ** database file.

67810. 39==sqlite3LogEst(15)

67811. ** Head and tail of a linked list of all outstanding allocations

67812. Number of open viewpoints (0 -> none)

67813. Raw hash value of the key

67814. ** Write the supplied master journal name into the journal file for pager ** pPager at the current location. The master journal name must be the last ** thing written to a journal file. If the pager is in full-sync mode, the ** journal file descriptor is advanced to the next sector boundary before ** anything is written. The format is: ** * + 4 bytes: PAGER_MJ_PGNO. ** + N bytes: Master journal filename in utf-8. ** + 4 bytes: N (length of master journal name in bytes, no null-terminator). ** + 4 bytes: Master journal name checksum. ** + 8 bytes: aJournalMagic[]. ** * The master journal page checksum is the sum of the bytes in the master ** journal name, where each byte is interpreted as a signed 8-bit integer. ** * If zMaster is a NULL pointer (occurs for a single database transaction), ** this call is a no-op.

67815. Second sorted list to be merged

67816. ESCAPE

67817. ** Historically, SQLite has used both the LockFile and LockFileEx functions. ** When the LockFile function was used, it was always expected to fail ** immediately if the lock could not be obtained. Also, it always expected to ** obtain an exclusive lock. These flags are used with the LockFileEx function ** and reflect those expectations; therefore, they should not be changed.

67818. IMPLEMENTATION-OF: R-26801-64137 If the xInit() method is NULL, then the ** built-in default page cache is used instead of the application defined ** page cache.

67819. Less than 19 digits, so we know that it fits in 64 bits

67820. The principle used to locate the table name in the CREATE TABLE ** statement is that the table name is the first non-space token that ** is immediately followed by a TK_LP or TK_USING token.

67821. **comment:** #define sqliteHashKeysize(E) ((E)->nKey) // NOT USED

label: code-design

67822. The result set must be the special operator **

67823. ** The following structure describes the FROM clause of a SELECT statement. ** Each table or subquery in the FROM clause is a separate element of ** the SrcList.a[] array. ** * With the addition of multiple database support, the following structure ** can also be used to describe a particular table such as the table that ** is modified by an INSERT, DELETE, or UPDATE statement. In standard SQL, ** such a table must be a simple name: ID. But in SQLite, the table can ** now be identified by a database name, a dot, then the table name: ID.ID. ** * The jointype starts out showing the join type between the current table ** and the next table on the list. The parser builds the list this way. ** But sqlite3SrcListShiftJoinType() later shifts the jointypes so that each ** jointype expresses the join between the table and the previous table. ** * In the colUsed field, the high-order bit (bit 63) is set if the table ** contains more than 63 columns and the 64-th or later column is used.

67824. Term.pExpr contains a correlated sub-query

67825. If a prior error occurred, report that error again.

67826. **comment:** ** OMIT_TEMPDB is set to 1 if SQLITE OMIT_TEMPDB is defined, or 0 ** afterward. Having this macro allows us to cause the C compiler ** to omit code used by TEMP tables without messy #ifndef statements.

label: code-design

67827. Total number of tokens in cluster

67828. Update statements used while promoting segments

67829. ** Delete an entire SrcList including all its substructure.
67830. 1300
67831. Maximum value
67832. If there is a vector == or IS term - e.g. "(a, b) == (?, ?)" - create ** new terms for each component comparison - "a = ?" and "b = ?". The ** new terms completely replace the original vector comparison, which is ** no longer used. *** This is only required if at least one side of the comparison operation ** is not a sub-select.
67833. **comment:** ** Routine needed to support the testcase() macro.
label: test
67834. Incremental vacuum
67835. ** Implementation of the sqlite3_pcache.xPagecount method.
67836. Size of suffix (nTerm - nPrefix)
67837. Find a node to store this cell in. pNode->iNode currently contains ** the height of the sub-tree headed by the cell.
67838. Number of entrances
67839. (286) nm ::= ID|INDEXED
67840. **comment:** ** The following macros are used to suppress compiler warnings and to ** make it clear to human readers when a function parameter is deliberately ** left unused within the body of a function. This usually happens when ** a function is called via a function pointer. For example the ** implementation of an SQL aggregate step callback may not use the ** parameter indicating the number of arguments passed to the aggregate, ** if it knows that this is enforced elsewhere. ** ** When a function parameter is not used at all within the body of a function, ** it is generally named "NotUsed" or "NotUsed2" to make things even clearer. ** However, these macros may also be used to suppress warnings related to ** parameters that may or may not be used depending on compilation options. ** For example those parameters only used in assert() statements. In these ** cases the parameters are named as per the usual conventions.
label: code-design
67841. Different columns indexed
67842. ** Routines used to compute the sum, average, and total. *** The SUM() function follows the (broken) SQL standard which means ** that it returns NULL if it sums over no inputs. TOTAL returns ** 0.0 in that case. In addition, TOTAL always returns a float where ** SUM might return an integer if it never encounters a floating point ** value. TOTAL never fails, but SUM might through an exception if ** it overflows an integer.
67843. ** Allocate and return an RBU handle with all fields zeroed except for the ** error code, which is set to SQLITE_MISUSE.
67844. The uptr type is an unsigned integer large enough to hold a pointer
67845. ** This function is called for each Fts3Phrase in a full-text query ** expression to initialize the mechanism for returning rows. Once this ** function has been called successfully on an Fts3Phrase, it may be ** used with fts3EvalPhraseNext() to iterate through the matching docids. *** If parameter bOptOk is true, then the phrase may (or may not) use the ** incremental loading strategy. Otherwise, the entire doclist is loaded into ** memory within this call. *** SQLITE_OK is returned if no error occurs, otherwise an SQLite error code.
67846. File handle we are reading from
67847. Size of this allocation
67848. The new AtoF never returns NaN
67849. The PRIMARY KEY index on the table
67850. Set properties of a table column based on the (magical) ** name of the column.
67851. If it has not already been created, create the rbu_state table
67852. The default collating sequence (BINARY)
67853. SQLITE_OMIT_INCRBLOB
67854. ** Allocate a new MergeEngine object to merge the contents of nPMA level-0 ** PMAs from pTask->file. If no error occurs, set *ppOut to point to ** the new object and return SQLITE_OK. Or, if an error does occur, set *ppOut ** to NULL and return an SQLite error code. *** When this function is called, *piOffset is set to the offset of the ** first PMA to read from pTask->file. Assuming no error occurs, it is ** set to the offset immediately following the last byte of the last ** PMA before returning. If an error does occur, then the final value of ** *piOffset is undefined.
67855. The key to seek for
67856. Array of segments. aSeg[0] is oldest.
67857. ***** Begin destructor definitions *****
67858. minus_num ::= MINUS INTEGER|FLOAT
67859. ** Parameter zName is the name of a table that is about to be altered ** (either with ALTER TABLE ... RENAME TO or ALTER TABLE ... ADD COLUMN). ** If the table is a system table, this function leaves an error message ** in pParse->zErr (system tables may not be altered) and returns non-zero. *** Or, if zName is not a system table, zero is returned.
67860. ** CAPI3REF: Enable Or Disable Shared Pager Cache ** ** ^/(This routine enables or disables the sharing of the database cache ** and schema data structures between [database connection | connections] ** to the same database. Sharing is enabled if the argument is true ** and disabled if the argument is false.)^** ** ^Cache sharing is enabled and disabled for an entire process. ** This is a change as of SQLite [version 3.5.0] ([dateof:3.5.0]). ** In prior versions of SQLite, ** sharing was enabled or disabled for each thread separately. *** ^/(The cache sharing mode set by this interface effects all subsequent ** calls to [sqlite3_open], [sqlite3_open_v2], and [sqlite3_open16]. ** Existing database connections continue use the sharing mode ** that was in effect at the time they were opened.)^** ** ^/(This routine returns [SQLITE_OK] if shared cache was enabled or disabled ** successfully. An [error code] is returned otherwise.)^** ** ^Shared cache is disabled by default. But this might change in ** future releases of SQLite. Applications that care about shared ** cache setting should set it explicitly. *** ** Note: This method is disabled on MacOS X 10.7 and iOS version 5.0 ** and will always return SQLITE_MISUSE. On those systems, ** shared cache mode should be enabled per-database connection via ** [sqlite3_open_v2] with [SQLITE_OPEN_SHAREDCACHE]. *** This interface is threadsafe on processors where writing a ** 32-bit integer is atomic. *** See Also: [SQLite Shared-Cache Mode]
67861. 296
67862. For looping through pKeyInfo->aColl[]
67863. 314
67864. 242
67865. Make the new connection a child of the winShmNode
67866. Update anDLt[], anLt[] and anEq[] to reflect the values that apply ** to the current row of the index.
67867. Number of function arguments
67868. ** Process an UPDATE statement. *** UPDATE OR IGNORE table_wxyz SET a=b, c=d WHERE e<5 AND f NOT NULL; ** _____/_____/_____/
 _____/* onError pTabList pChanges pWhere
67869. Opcode: Real * P2 * P4 ** Synopsis: r[P2]=P4 *** P4 is a pointer to a 64-bit floating point value. ** Write that value into register P2.
67870. The boolean index is empty
67871. Number of users of this page
67872. Do not allow IS constraints from the WHERE clause to be used by the ** right table of a LEFT JOIN. Only constraints in the ON clause are ** allowed
67873. ** The following global variable is incremented whenever the library ** attempts to open a temporary file. This information is used for ** testing and analysis only.
67874. ** 2008 August 16 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil.
 ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
 ***** This file contains routines used for walking the parser tree for ** an SQL statement.
67875. **comment:** ** CAPI3REF: Invert A Changeset ** ** This function is used to "invert" a changeset object. Applying an inverted ** changeset to a database reverses the effects of applying the uninverted ** changeset. Specifically: ** ** ** Each DELETE change is changed to an INSERT, and ** Each INSERT change is changed to a DELETE, and ** For each UPDATE change, the old.* and new.* values are exchanged. ** *** This function does not change the order in which changes appear within ** the changeset. It merely reverses the sense of each individual change. ** ** If successful, a pointer to a buffer containing the inverted changeset ** is stored in *ppOut, the size of the same buffer is stored in *pnOut, and ** SQLITE_OK is returned. If an error occurs, both *pnOut and *ppOut are ** zeroed and an SQLite error code returned. *** It is the responsibility of the caller to eventually call sqlite3_free() ** on the *ppOut pointer to free the buffer allocation following a successful ** call to this function. *** WARNING/TODO: This function currently assumes that the input is a valid ** changeset. If it is not, the results are undefined.
label: code-design

67876. Append the new shared library handle to the db->aExtension array.
67877. Key information
67878. 69
67879. If no error has occurred so far, check if the "number of entries" ** field on the node is too large. If so, set the return code to ** SQLITE_CORRUPT_VTAB.
67880. ** Open a new iterator to iterate though all rowid that match the ** specified token or token prefix.
67881. Extra bits for pNew->eOperator
67882. Loop through the elements that will make up the record to figure ** out how much space is required for the new record.
67883. EVIDENCE-OF: R-47608-56469 Any other value for the b-tree page type is ** an error.
67884. SQLITE_CORRUPT
67885. 1480
67886. 012345678901234567
67887. Indicate VList may no longer be enlarged
67888. Memory cell used for the row counter
67889. Argument to the deferred sqlite3BtreeMoveto()
67890. Is an alias for a result set column
67891. Unique index on parent key columns in pTab
67892. 0
67893. **comment:** ** Try to obtain a page from the cache. *** This routine returns a pointer to an sqlite3_pcache_page object if ** such an object is already in cache, or if a new one is created. ** This routine returns a NULL pointer if the object was not in cache ** and could not be created. *** The createFlags should be 0 to check for existing pages and should ** be 3 (not 1, but 3) to try to create a new page. *** If the createFlag is 0, then NULL is always returned if the page ** is not already in the cache. If createFlag is 1, then a new page ** is created only if that can be done without spilling dirty pages ** and without exceeding the cache size limit. *** The caller needs to invoke sqlite3PcacheFetchFinish() to properly ** initialize the sqlite3_pcache_page object and convert it into a ** PgHdr object. The sqlite3PcacheFetch() and sqlite3PcacheFetchFinish() ** routines are split this way for performance reasons. When separated ** they can both (usually) operate without having to push values to ** the stack on entry and pop them back off on exit, which saves a ** lot of pushing and popping.
label: code-design
67894. ***** Begin file fts3_hash.c *****
67895. ** Return the affinity for a single column of an index.
67896. (282) savepoint_opt ::=
67897. Any kind of inner or cross join
67898. A minimum of one cursor is required if autoincrement is used * See ticket [a696379c1f08866]
67899. 350
67900. Tokenizer implementation
67901. PRIMARY KEY changed in a WITHOUT ROWID table
67902. Values extracted from pExpr
67903. 227
67904. Bitmask of phrases seen by BestSnippet()
67905. True for a >= search
67906. **comment:** ** Run the parser on the given SQL string. The parser structure is ** passed in. An SQLITE_status code is returned. If an error occurs ** then an attempt is made to write an error message into ** memory obtained from sqlite3_malloc() and to make *pzErrMsg point to that ** error message.
label: code-design
67907. Only a BTREE_SINGLE database can be BTREE_UNORDERED
67908. expr ::= expr in_op LP exprlist RP
67909. ** The implementation of an SQLite user-defined-function that accepts a ** single integer as an argument. If the integer is an alpha-numeric ** unicode code point, 1 is returned. Otherwise 0.
67910. System page size
67911. **comment:** Assigned value is never used
label: code-design
67912. ** Takes an open conch file, copies the contents to a new path and then moves ** it back. The newly created file's file descriptor is assigned to the ** conch file structure and finally the original conch file descriptor is ** closed. Returns zero if successful.
67913. Wait a little before trying again
67914. Check the content overflow list
67915. Try to open on pSnapshot when the next read-transaction starts
67916. ** Implementation of the sqlite_log() function. This is a wrapper around ** sqlite3_log(). The return value is NULL. The function exists purely for ** its side-effects.
67917. idx = biasRight ? upr : (lwr+upr)/2;
67918. ** Generate deferred-doclists for all tokens in the pCsr->pDeferred list ** based on the row that pCsr currently points to. *** A deferred-doclist is like any other doclist with position information ** included, except that it only contains entries for a single row of the ** table, not for all rows.
67919. ** Number of freelist hash slots
67920. pLoop->nOut should not exceed nRow-iReduce
67921. SQL_SHIFT_SEGDIR_ENTRY ** Modify the idx value for the segment with idx=:3 on absolute level :2 ** to :1.
67922. EVIDENCE-OF: R-34926-03360 SQLITE_CONFIG_WIN32_HEAPSIZE takes a 32-bit ** unsigned integer value that specifies the maximum size of the created ** heap.
67923. Label resolved to end of generated code
67924. Loop counter: Foreign key number for pTab
67925. ** If possible, return a pointer to a mapping of file fd starting at offset ** iOff. The mapping must be valid for at least nAmt bytes. *** If such a pointer can be obtained, store it in *pp and return SQLITE_OK. ** Or, if one cannot but no error occurs, set *pp to 0 and return SQLITE_OK. ** Finally, if an error does occur, return an SQLite error code. The final ** value of *pp is undefined in this case. *** If this function does return a pointer, the caller must eventually ** release the reference by calling winUnfetch().
67926. ***** End Win32 Threads *****
67927. Constraint operator
67928. cmd ::= ALTER TABLE fullname RENAME TO nm
67929. Load the lowest cost path into pWInfo
67930. ***** Begin file stmt.c *****
67931. Release the array of temp registers
67932. **comment:** It is possible that there is a checkpointer thread running ** concurrent with this code. If this is the case, it may be that the ** checkpointer has already determined that it will checkpoint ** snapshot X, where X is later in the wal file than pSnapshot, but ** has not yet set the pInfo->nBackfillAttempted variable to indicate ** its intent. To avoid the race condition this leads to, ensure that ** there is no checkpointer process by taking a shared CKPT lock ** before checking pInfo->nBackfillAttempted. *** TODO: Does the aReadMark[] lock prevent a checkpointer from doing ** this already?
label: code-design
67933. The VDBE cursor that implements the sort
67934. Jump to this label on an OE_Ignore resolution
67935. Index of the term to be analyzed
67936. Top of the rowid change constraint check
67937. Arg passed to comparison function
67938. STRACCUM_NOMEM or STRACCUM_TOOBIG
67939. defined(_MSC_VER)
67940. #include <string.h>
67941. Number of CTEs in the WITH clause

67942. pageno
67943. The user should invoke this in one of two forms: *** CREATE VIRTUAL TABLE xxx USING fts4aux(fts4-table); ** CREATE VIRTUAL TABLE xxx USING fts4aux(fts4-table-db, fts4-table);
67944. Bytes allocated at pCsr->apSegment[]
67945. Writer object
67946. 10
67947. Opcode: Lt P1 P2 P3 P4 P5 ** Synopsis: IF r[P3]<r[P1] ** Compare the values in register P1 and P3. If reg(P3)<reg(P1) then ** jump to address P2. Or if the SQLITE_STOREP2 flag is set in P5 store ** the result of comparison (0 or 1 or NULL) into register P2. ** ** If the SQLITE_JUMPIFNULL bit of P5 is set and either reg(P1) or ** reg(P3) is NULL then take the jump. If the SQLITE_JUMPIFNULL ** bit is clear then fall through if either operand is NULL. ** ** The SQLITE_AFF_MASK portion of P5 must be an affinity character - ** SQLITE_AFF_TEXT, SQLITE_AFF_INTEGER, and so forth. An attempt is made ** to coerce both inputs according to this affinity before the ** comparison is made. If the SQLITE_AFF_MASK is 0x00, then numeric ** affinity is used. Note that the affinity conversions are stored ** back into the input registers P1 and P3. So this opcode can cause ** persistent changes to registers P1 and P3. ** ** Once any conversions have taken place, and neither value is NULL, ** the values are compared. If both values are blobs then memcmp() is ** used to determine the results of the comparison. If both values ** are text, then the appropriate collating function specified in ** P4 is used to do the comparison. If P4 is not specified then ** memcmp() is used to compare text string. If both values are ** numeric, then a numeric comparison is used. If the two values ** are of different types, then numbers are considered less than ** strings and strings are considered less than blobs.
67948. Distance from left to right (1=adjacent)
67949. same as TK_NOT, synopsis: r[P2]=!r[P1]
67950. 620
67951. Size of buffer zRoot
67952. Number of bytes in header
67953. ** Triggers may access values stored in the old.* or new.* pseudo-table. ** This function returns a 32-bit bitmask indicating which columns of the ** old.* or new.* tables actually are used by triggers. This information ** may be used by the caller, for example, to avoid having to load the entire ** old.* record into memory when executing an UPDATE or DELETE command. ** ** Bit 0 of the returned mask is set if the left-most column of the ** table may be accessed using an [old|new].<col> reference. Bit 1 is set if ** the second leftmost column value is required, and so on. If there ** are more than 32 columns in the table, and at least one of the columns ** with an index greater than 32 may be accessed, 0xffffffff is returned. ** ** It is not possible to determine if the old.rowid or new.rowid column is ** accessed by triggers. The caller must always assume that it is. ** ** Parameter isNew must be either 1 or 0. If it is 0, then the mask returned ** applies to the old.* table. If 1, the new.* table. ** ** Parameter tr_tm must be a mask with one or both of the TRIGGER_BEFORE ** and TRIGGER_AFTER bits set. Values accessed by BEFORE triggers are only ** included in the returned mask if the TRIGGER_BEFORE bit is set in the ** tr_tm parameter. Similarly, values accessed by AFTER triggers are only ** included in the returned mask if the TRIGGER_AFTER bit is set in tr_tm.
67954. Original number of pages in file
67955. If pList==0, it means this routine was called to make a primary ** key out of the last column added to the table under construction. ** So create a fake list to simulate this.
67956. Register holding NULL status. See notes
67957. **comment:** ** 2007 August 14 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

** SQLite will use the standard C-library malloc/realloc/free interface ** to obtain the memory it needs. ** ** This file contains implementations of the low-level memory allocation ** routines specified in the sqlite3_mem_methods object. The content of ** this file is only used if SQLITE_SYSTEM_MALLOC is defined. The ** SQLITE_SYSTEM_MALLOC macro is defined automatically if neither the ** SQLITE_MEMDEBUG nor the SQLITE_WIN32_MALLOC macros are defined. The ** default configuration is to use memory allocation routines in this ** file. ** ** C-preprocessor macro summary: ** ** HAVE_MALLOC_USABLE_SIZE The configure script sets this symbol if ** the malloc_usable_size() interface exists ** on the target platform. Or, this symbol ** can be set manually, if desired. ** If an equivalent interface exists by ** a different name, using a separate -D ** option to rename it. ** ** SQLITE WITHOUT_ZONEMALLOC Some older macs lack support for the zone ** memory allocator. Set this symbol to enable ** building on older macs. ** ** SQLITE WITHOUT_MSIZE Set this symbol to disable the use of ** _msize() on windows systems. This might ** be necessary when compiling for Delphi, ** for example.
label: code-design
67958. ** Check that argument nHeight is less than or equal to the maximum ** expression depth allowed. If it is not, leave an error message in ** pParse.
67959. ** Interpret the given string as a temp db location. Return 1 for file ** backed temporary databases, 2 for the Red-Black tree in memory database ** and 0 to use the compile-time default.
67960. ** This routine sets the state to CURSORFAULT and the error ** code to errCode for every cursor on any BtShared that pBtree ** references. Or if the writeOnly flag is set to 1, then only ** trip write cursors and leave read cursors unchanged. ** ** Every cursor is a candidate to be tripped, including cursors ** that belong to other database connections that happen to be ** sharing the cache with pBtree. ** ** This routine gets called when a rollback occurs. If the writeOnly ** flag is true, then only write-cursors need be tripped - read-only ** cursors save their current positions so that they may continue ** following the rollback. Or, if writeOnly is false, all cursors are ** tripped. In general, writeOnly is false if the transaction being ** rolled back modified the database schema. In this case b-tree root ** pages may be moved or deleted from the database altogether, making ** it unsafe for read cursors to continue. ** ** If the writeOnly flag is true and an error is encountered while ** saving the current position of a read-only cursor, all cursors, ** including all read-cursors are tripped. ** ** SQLITE_OK is returned if successful, or if an error occurs while ** saving a cursor position, an SQLite error code.
67961. Set rCostIdx to the cost of visiting selected rows in index. Add ** it to pNew->rRun, which is currently set to the cost of the index ** seek only. Then, if this is a non-covering index, add the cost of ** visiting the rows in the main table.
67962. nothing to do if the path is NULL, :auto: or matches the existing path
67963. Number of key columns in pIndex
67964. min(maxLocal,127)
67965. We should never write to the journal file the page that ** contains the database locks. The following assert verifies ** that we do not.
67966. Free memory immediately, rather than waiting on sqlite3_finalize()
67967. ** Move the write position of the WAL back to the point identified by ** the values in the aWalData[] array. aWalData must point to an array ** of WAL_SAVEPOINT_NDATA u32 values that has been previously populated ** by a call to WalSavepoint().
67968. The trigger that this step is a part of
67969. Cannot have C without B
67970. True if nTotalRow/aTotalSize[] are valid
67971. Current index in aLeft
67972. ** Table p is a virtual table. This function moves all elements in the ** p->pVTable list to the sqlite3.pDisconnect lists of their associated ** database connections to be disconnected at the next opportunity. ** Except, if argument db is not NULL, then the entry associated with ** connection db is left in the p->pVTable list.
67973. Insert into rbu_tmp_.\$zDataTbl
67974. Normally sqlite3SelectExpand() will be called first and will have ** already expanded this SELECT. However, if this is a subquery within ** an expression, sqlite3ResolveExprNames() will be called without a ** prior call to sqlite3SelectExpand(). When that happens, let ** sqlite3SelectPrep() do all of the processing for this SELECT. ** sqlite3SelectPrep() will invoke both sqlite3SelectExpand() and ** this routine in the correct order.
67975. WAL savepoint context
67976. Token:
67977. Use SQLITE_ENABLE_COMMENTS to enable generation of extra comments on ** each VDBE opcode. ** ** Use the SQLITE_ENABLE_MODULE_COMMENTS macro to see some extra no-op ** comments in VDBE programs that show key decision points in the code ** generator.
67978. 52
67979. Upper bound on the range. ex: "x<455" Might be NULL
67980. The virtual database engine
67981. Index of langid=x constraint, if present
67982. ** Set the bit number pgno in the PagerSavepoint.pInSavepoint ** bitvecs of all open savepoints. Return SQLITE_OK if successful ** or SQLITE_NOMEM if a malloc failure occurs.

67983. 0x7FFFFFFF is the hard limit for the number of pages in a database ** file. By passing this as the number of pages to copy to ** sqlite3_backup_step(), we can guarantee that the copy finishes ** within a single call (unless an error occurs). The assert() statement ** checks this assumption - (p->rc) should be set to either SQLITE_DONE ** or an error code.

67984. Get the database meta information. ** ** Meta values are as follows: ** meta[0] Schema cookie. Changes with each schema change. ** meta[1] File format of schema layer. ** meta[2] Size of the page cache. ** meta[3] Largest rootpage (auto/incr_vacuum mode) ** meta[4] Db text encoding. 1=UTF-8 2=UTF-16LE 3=UTF-16BE ** meta[5] User version ** meta[6] Incremental vacuum mode ** meta[7] unused ** meta[8] unused ** meta[9] unused ** ** Note: The #defined SQLITE_UTF* symbols in sqliteInt.h correspond to ** the possible values of meta[4].

67985. 1st argument to callback

67986. Make a copy of an sqlite3_value object

67987. ** This routine deletes the RtreeGeomCallback object that was attached ** one of the SQL functions create by sqlite3_rtree_geometry_callback() ** or sqlite3_rtree_query_callback(). In other words, this routine is the ** destructor for an RtreeGeomCallback objecct. This routine is called when ** the corresponding SQL function is deleted.

67988. Mutex configuration options are only available in a threadsafe ** compile.

67989. Right hand side of comparison

67990. Type of the new cursor

67991. SQLITE_OMIT_VIRTUALTABLE

67992. First byte of token

67993. ***** Begin file backup.c *****

67994. ** The rbu_state table is used to save the state of a partially applied ** update so that it can be resumed later. The table consists of integer ** keys mapped to values as follows: ** ** RBU_STATE_STAGE: ** May be set to integer values 1, 2, 4 or 5. As follows: ** 1: the *-rbu file is currently under construction. ** 2: the *-rbu file has been constructed, but not yet moved ** to the *-wal path. ** 4: the checkpoint is underway. ** 5: the rbu update has been checkpointed. ** ** RBU_STATE_TBL: ** Only valid if STAGE==1. The target database name of the table ** currently being written. ** ** RBU_STATE_IDX: ** Only valid if STAGE==1. The target database name of the index ** currently being written, or NULL if the main table is currently being ** updated. ** ** RBU_STATE_ROW: ** Only valid if STAGE==1. Number of rows already processed for the current ** table/index. ** ** RBU_STATE_PROGRESS: ** Trbul number of sqlite3rbu_step() calls made so far as part of this ** rbu update. ** ** RBU_STATE_CKPT: ** Valid if STAGE==4. The 64-bit checksum associated with the wal-index ** header created by recovering the *-wal file. This is used to detect ** cases when another client appends frames to the *-wal file in the ** middle of an incremental checkpoint (an incremental checkpoint cannot ** be continued if this happens). ** ** RBU_STATE_COOKIE: ** Valid if STAGE==1. The current change-counter cookie value in the ** target db file. ** ** RBU_STATE_OALSZ: ** Valid if STAGE==1. The size in bytes of the *-oal file.

67995. The subquery

67996. True for "patchset", 0 for "changeset"

67997. Current number of cells in pNode

67998. from vdbeSortSubtaskMain()

67999. Language id to query

68000. 185

68001. ** CAPI3REF: Find the next prepared statement ** METHOD: sqlite3 ** ** ^This interface returns a pointer to the next [prepared statement] after ** pStmt associated with the [database connection] pDb. ^If pStmt is NULL ** then this interface returns a pointer to the first prepared statement ** associated with the database connection pDb. ^If no prepared statement ** satisfies the conditions of this routine, it returns NULL. ** ** The [database connection] pointer D in a call to ** [sqlite3_next_stmt(D,S)] must refer to an open database ** connection and in particular must not be a NULL pointer.

68002. Length of the next token token

68003. Zero the output variables in case an error occurs. If this session ** object is already in the error state (sqlite3_session.rc != SQLITE_OK), ** this call will be a no-op.

68004. Which columns are used by this index. 1st is 0

68005. Array of flags, set on NOT NULL columns

68006. 1500

68007. ** End of MatchinfoBuffer code. *****

68008. Log the error, but continue normal operation using xRead/xWrite

68009. ** Return a static string containing the name corresponding to the error code ** specified in the argument.

68010. 480

68011. 0x20 .. 0x2F

68012. De-reference and close our copy of the shared memory handle

68013. **comment:** Sync the journal file and write all dirty pages to the database. ** If the atomic-update optimization is being used, this sync will not ** create the journal file or perform any real IO. ** ** Because the change-counter page was just modified, unless the ** atomic-update optimization is used it is almost certain that the ** journal requires a sync here. However, in locking_mode=exclusive ** on a system under memory pressure it is just possible that this is ** not the case. In this case it is likely enough that the redundant ** xSync() call will be changed to a no-op by the OS anyhow.
label: code-design

68014. The cursor to seek

68015. ***** End %parse_failure code *****

68016. Cached OP_Column parse information is only valid if cacheStatus matches ** Vdbe.cacheCtr. Vdbe.cacheCtr will never take on the value of ** CACHE_STALE (0) and so setting cacheStatus=CACHE_STALE guarantees that ** the cache is out of date.

68017. The P3 operand

68018. Old value for iStartBlock

68019. Argument set

68020. For looping over result columns

68021. Write all frames into the log file exactly once

68022. **comment:** ** The "printf" code that follows dates from the 1980's. It is in ** the public domain. **
***** This file contains code for a set of "printf"-like routines. These ** routines format strings much like the printf() from the standard C ** library, though the implementation here has enhancements to support ** SQLite.
label: code-design

68023. Remove all pages with pgno>x. Reset the cache if x==0

68024. **comment:** Return TRUE if Mem X contains dynamically allocated content - anything ** that needs to be deallocated to avoid a leak.
label: code-design

68025. ** An instance of this function is used to merge together the (potentially ** large number of) doclists for each term that matches a prefix query. ** See function fts3TermSelectMerge() for details.

68026. 200

68027. Allocated entries

68028. Retrieve matchinfo() data.

68029. Number of rows for a single term

68030. Canonical data cursor (main table or PK index)

68031. Opcode: OffsetLimit P1 P2 P3 * * ** Synopsis: if r[P1]>0 then r[P2]=r[P1]+max(0,r[P3]) else r[P2]=(-1) ** ** This opcode performs a commonly used computation associated with ** LIMIT and OFFSET process. r[P1] holds the limit counter. r[P3] ** holds the offset counter. The opcode computes the combined value ** of the LIMIT and OFFSET and stores that value in r[P2]. The r[P2] ** value computed is the total number of rows that will need to be ** visited in order to complete the query. ** ** If r[P3] is zero or negative, that means there is no OFFSET ** and r[P2] is set to be the value of the LIMIT, r[P1]. ** ** If r[P1] is zero or negative, that means there is no LIMIT ** and r[P2] is set to -1. ** ** Otherwise, r[P2] is set to the sum of r[P1] and r[P3].

68032. The database being vacuemed

68033. same as TK_CONCAT, in1, in2, out3

68034. In case an error occurs

68035. iCol must be less than p->pEList->nExpr. Otherwise an error would ** have been thrown during name resolution and we would not have gotten ** this far

68036. SQLITE_OMIT_LOAD_EXTENSION

68037. 75
68038. ** Increase the memory allocation for pLoop->aLTerm[] to be at least n.
68039. ** Attempt to read the database schema and initialize internal ** data structures for a single database file. The index of the ** database file is given by iDb. iDb==0 is used for the main ** database. iDb==1 should never be used. iDb>=2 is used for ** auxiliary databases. Return one of the SQLITE_error codes to ** indicate success or failure.
68040. Database name or NULL
68041. expr ::= LP expr RP
68042. Fts3 table cursor
68043. NO => ID
68044. seltablist
68045. ***** The fts5_expr.c API above this point is used by the other hand-written ** C code in this module. The interfaces below this point are called by ** the parser code in fts5parse.y.
68046. Local reference to page 1
68047. IN/OUT: Pointer into position-list buffer
68048. A frame is only valid if the salt values in the frame-header ** match the salt values in the wal-header.
68049. Check that there isn't an ORDER BY without a LIMIT clause.
68050. Affinity string to return
68051. ** Save the current cursor position in the variables BtCursor.nKey ** and BtCursor.pKey. The cursor's state is set to CURSOR_REQUIRESEEK. *** The caller must ensure that the cursor is valid (has eState==CURSOR_VALID) ** prior to calling this routine.
68052. Max seg id number 65535
68053. ** Parse a single JSON value which begins at pParse->zJson[i]. Return the ** index of the first character past the end of the value parsed. *** Return negative for a syntax error. Special cases: return -2 if the ** first non-whitespace character is ')' and return -3 if the first ** non-whitespace character is ']'.
68054. 37
68055. Enforce foreign key constraints
68056. First leaf to traverse
68057. EVIDENCE-OF: R-58063-38258 SQLITE_CONFIG_MMAP_SIZE takes two 64-bit ** integer (sqlite3_int64) values that are the default mmap size limit ** (the default setting for PRAGMA mmap_size) and the maximum allowed ** mmap size limit.
68058. To iterate through builtin functions
68059. Complete information about the WHERE clause
68060. NULL-terminated version of pName
68061. Return code from sqlite3_reset()
68062. Do full auto-vacuum
68063. ***** Below this point is the implementation of the integrity-check ** functionality.
68064. Program implementing pTrigger/orconf
68065. INSERT INTO %_content VALUES(...)
68066. Return the value to store in the "stat" column of the sqlite_stat1 ** table for this index. *** The value is a string composed of a list of integers describing ** the index. The first integer in the list is the total number of ** entries in the index. There is one additional integer in the list ** for each indexed column. This additional integer is an estimate of ** the number of rows matched by a stabbing query on the index using ** a key with the corresponding number of fields. In other words, ** if the index is on columns (a,b) and the sqlite_stat1 value is ** "100 10 2", then SQLite estimates that: *** * the index contains 100 rows, *** "WHERE a=? matches 10 rows, and *** "WHERE a=? AND b=?" matches 2 rows. *** If D is the count of distinct values and K is the total number of ** rows, then each estimate is computed as: *** I = (K+D-1)/D
68067. SQLITE_ENABLE_LOCKING_STYLE && !OS_VXWORK
68068. Btree with currently open write transaction
68069. ** Return the file handle for the database file associated ** with the pager. This might return NULL if the file has ** not yet been opened.
68070. Initialize the counter of the number of rows deleted, if ** we are counting rows.
68071. ** 2015 January 12 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code that is specific to MSVC.
68072. Iterate through tables
68073. ** Fill a buffer with pseudo-random bytes. This is used to preset ** the content of a new memory allocation to unpredictable values and ** to clear the content of a freed allocation to unpredictable values.
68074. ** CAPI3REF: Zero Scan-Status Counters ** METHOD: sqlite3_stmt ** ** ^Zero all [sqlite3_stmt_scanstatus()] related event counters. *** This API is only available if the library is built with pre-processor ** symbol [SQLITE_ENABLE_STMT_SCANSTATUS] defined.
68075. Routine to run in a separate thread
68076. 6 SEMI:
68077. # include "sqliteInt.h"
68078. ** Load the doclists for each phrase in the query associated with FTS3 cursor ** pCsr. ** ** If pnPhrase is not NULL, then *pnPhrase is set to the number of matchable ** phrases in the expression (all phrases except those directly or ** indirectly descended from the right-hand-side of a NOT operator). If ** pnToken is not NULL, then it is set to the number of tokens in all ** matchable phrases of the expression.
68079. Allocate space for the previous row
68080. Address of the i-th cell
68081. Byte offset of blob in cursor data
68082. The database of pSrc
68083. ** 2004 May 26 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code use to implement APIs that are part of the ** VDBE.
68084. Check for interrupts on this handle
68085. Ensure that v is greater than zero
68086. Look to see if there is an existing winShmNode that can be used. ** If no matching winShmNode currently exists, create a new one.
68087. If true, pretend pTab contains all NULL values
68088. The BETWEEN expression
68089. User data parameter
68090. For iterating through doclist index
68091. ALL
68092. ** CAPI3REF: Obtain The Primary Key Definition Of A Table ** ** For each modified table, a changeset includes the following: ** ** ** The number of columns in the table, and ** Which of those columns make up the tables PRIMARY KEY. ** ** ** This function is used to find which columns comprise the PRIMARY KEY of ** the table modified by the change that iterator pIter currently points to. ** If successful, *pabPK is set to point to an array of nCol entries, where ** nCol is the number of columns in the table. Elements of *pabPK are set to ** 0x01 if the corresponding column is part of the tables primary key, or ** 0x00 if it is not. ** ** If argument pnCol is not NULL, then *pnCol is set to the number of columns ** in the table. ** ** If this function is called when the iterator does not point to a valid ** entry, SQLITE_MISUSE is returned and the output variables zeroed. Otherwise, ** SQLITE_OK is returned and the output variables populated as described ** above.
68093. Full size
68094. ** Find the current time (in Universal Coordinated Time). Write into *piNow ** the current time and date as a Julian Day number times 86_400_000. In ** other words, write into *piNow the number of milliseconds since the Julian ** epoch of noon in Greenwich on November 24, 4714 B.C according to the ** proleptic Gregorian calendar. *** On success, return SQLITE_OK. Return SQLITE_ERROR if the time and date ** cannot be found.

68095. Compute the comparison permutation and keyinfo that is used with ** the permutation used to determine if the next ** row of results comes from selectA or selectB. Also add explicit ** collations to the ORDER BY clause terms so that when the subqueries ** to the right and the left are evaluated, they use the correct ** collation.
 68096. ** Restore the state of the PRNG to the last state saved using ** PRNG_SAVE. If PRNG_SAVE has never before been called, then ** this verb acts like PRNG_RESET.
 68097. Left-most of SELECT of a compound
 68098. ***** Begin file pragma.h *****
 68099. ** Free a single node of an expression tree.
 68100. Last chance verification. If the sector size isn't a multiple of 512 ** then it isn't valid.
 68101. **comment:** ** Disable a term in the WHERE clause. Except, do not disable the term ** if it controls a LEFT OUTER JOIN and it did not originate in the ON ** or USING clause of that join. ** ** Consider the term t2.z='ok' in the following queries: ** ** (1) SELECT * FROM t1 LEFT JOIN t2 ON t1.a=t2.x WHERE t2.z='ok' ** (2) SELECT * FROM t1 LEFT JOIN t2 ON t1.a=t2.x AND t2.z='ok' ** (3) SELECT * FROM t1, t2 WHERE t1.a=t2.x AND t2.z='ok' ** ** The t2.z='ok' is disabled in the (2) because it originates ** in the ON clause. The term is disabled in (3) because it is not part ** of a LEFT OUTER JOIN. In (1), the term is not disabled. ** ** Disabling a term causes that term to not be tested in the inner loop ** of the join. Disabling is an optimization. When terms are satisfied ** by indices, we disable them to prevent redundant tests in the inner ** loop. We would get the correct results if nothing were ever disabled, ** but joins might run a little slower. The trick is to disable as much ** as we can without disabling too much. If we disabled in (1), we'd get ** the wrong answer. See ticket #813. ** ** If all the children of a term are disabled, then that term is also ** automatically disabled. In this way, terms get disabled if derived ** virtual terms are tested first. For example: ** ** x GLOB 'abc*' AND x>='abc' AND x<'acd' ** _____ / ____ / ____ / ** parent child1 child2 ** ** Only the parent term was in the original WHERE clause. The child1 ** and child2 terms were added by the LIKE optimization. If both of ** the virtual child terms are valid, then testing of the parent can be ** skipped. ** ** Usually the parent term is marked as TERM_CODED. But if the parent ** term was originally TERM_LIKE, then the parent gets TERM_LIKECOND instead. ** The TERM_LIKECOND marking indicates that the term should be coded inside ** a conditional such that is only evaluated on the second pass of a ** LIKE-optimization loop, when scanning BLOBS instead of strings.
label: code-design
 68102. cmd ::= RELEASE savepoint_opt nm
 68103. Root of the input tree
 68104. Next value for iLoop
 68105. A memory allocation error has occurred
 68106. Root node of the tree.
 68107. The following fields are only available in 3.8.11 and later
 68108. The controlling WhereLoop object
 68109. Subquery id 1
 68110. ** The TESTONLY macro is used to enclose variable declarations or ** other bits of code that are needed to support the arguments ** within testcase() and assert() macros.
 68111. value returned by declare_vtab()
 68112. ** Resolve names in the SELECT statement p and all of its descendants.
 68113. Test the storage function
 68114. Original, innermost WhereClause
 68115. List of SQL indexes on this table.
 68116. ** Write a nice string representation of the contents of cell pMem ** into buffer zBuf, length nBuf.
 68117. Record the instruction used to terminate the loop.
 68118. Copy of initial value of pIn1->flags
 68119. Restriction (13)
 68120. ***** Include btreeInt.h in the middle of btmutex.c *****
 68121. ** sqlite3WalkExpr() callback used by havingToWhere(). ** ** If the node passed to the callback is a TK_AND node, return ** WRC_Continue to tell sqlite3WalkExpr() to iterate through child nodes. ** ** Otherwise, return WRC_Prune. In this case, also check if the ** sub-expression matches the criteria for being moved to the WHERE ** clause. If so, add it to the WHERE clause and replace the sub-expression ** within the HAVING expression with a constant "1".
 68122. Jump to the end of the loop if the LIMIT is reached.
 68123. SQLITE_MISMATCH
 68124. Populate the StatCursor fields with the values to be returned ** by the xColumn() and xRowid() methods.
 68125. Original lockingcontext to restore on close
 68126. ** Free all structures in the Pager.aSavepoint[] array and set both ** Pager.aSavepoint and Pager.nSavepoint to zero. Close the sub-journal ** if it is open and the pager is not in exclusive mode.
 68127. IMP: R-48699-48617 This function is an SQL wrapper around the ** sqlite3_libversion() C-interface.
 68128. P4
 68129. Loop termination point for the schema loop
 68130. ** typedef for the authorization callback function.
 68131. Where to write the frame
 68132. ** NOTES ON TRANSACTIONS: ** ** SQLite invokes the following virtual table methods as transactions are ** opened and closed by the user: ** **
 xBegin(): Start of a new transaction. ** xSync(): Initial part of two-phase commit. ** xCommit(): Final part of two-phase commit. ** xRollback(): Rollback the transaction. ** ** Anything that is required as part of a commit that may fail is performed ** in the xSync() callback. Current versions of SQLite ignore any errors ** returned by xCommit(). ** ** And as sub-transactions are opened/closed: ** ** xSavepoint(int S): Open savepoint S. ** xRelease(int S): Commit and close savepoint S. ** xRollbackTo(int S): Rollback to start of savepoint S. ** ** During a write-transaction the fts5_index.c module may cache some data ** in-memory. It is flushed to disk whenever xSync(), xRelease() or ** xSavepoint() is called. And discarded whenever xRollback() or xRollbackTo() ** is called. ** ** Additionally, if SQLITE_DEBUG is defined, an instance of the following ** structure is used to record the current transaction state. This information ** is not required, but it is used in the assert() statements executed by ** function fts5CheckTransactionState() (see below).
 68133. 78
 68134. First Mem address for storing current GROUP BY
 68135. ***** Begin file insert.c *****
 68136. If we are trying to delete from a view, realize that view into ** an ephemeral table.
 68137. Check if this call is removing or replacing an existing collation ** sequence. If so, and there are active VMs, return busy. If there ** are no active VMs, invalidate any pre-compiled statements.
 68138. It is not possible for a REDUCE to be followed by an error
 68139. Pager object associated with page pPg
 68140. Otherwise, the argument is a column name.
 68141. Buffer to deserialize from
 68142. **comment:** ***** Beginning of RBU VFS shim methods. The VFS shim modifies the behaviour ** of a standard VFS in the following ways: ** ** 1. Whenever the first page of a main database file is read or ** written, the value of the change-counter cookie is stored in ** rbu_file.iCookie. Similarly, the value of the "write-version" ** database header field is stored in rbu_file.iWriteVer. This ensures ** that the values are always trustworthy within an open transaction. ** ** 2. Whenever an SQLITE_OPEN_WAL file is opened, the (rbu_file.pWalFd) ** member variable of the associated database file descriptor is set ** to point to the new file. A mutex protected linked list of all main ** db fds opened using a particular RBU VFS is maintained at ** rbu_vfs.pMain to facilitate this. ** ** 3. Using a new file-control "SQLITE_FCNTL_RBU", a main db rbu_file ** object can be marked as the target database of an RBU update. This ** turns on the following extra special behaviour: ** ** 3a. If xAccess() is called to check if there exists a *-wal file ** associated with an RBU target database currently in RBU_STAGE_OAL ** stage (preparing the *-oal file), the following special handling ** applies: ** ** * if the *-wal file does exist, return SQLITE_CANTOPEN. An RBU ** target database may not be in wal mode already. ** ** * if the *-wal file does not exist, set the output parameter to ** non-zero (to tell SQLite that it does exist) anyway. ** ** Then, when xOpen() is called to open the *-wal file associated with ** the RBU target in RBU_STAGE_OAL stage, instead of opening the *-wal ** file, the rbu vfs opens the corresponding *-oal file instead. ** ** 3b. The *-shm pages returned by xShmMap() for a target db file in ** RBU_STAGE_OAL mode are actually stored in heap memory. This is to ** avoid creating a *-shm file on disk. Additionally, xShmLock() calls ** are no-ops on target database files in RBU_STAGE_OAL mode. This is ** because assert() statements in

some VFS implementations fail if ** xShmLock() is called before xShmMap(). *** 3c. If an EXCLUSIVE lock is attempted on a target database file in any ** mode except RBU_STAGE_DONE (all work completed and checkpointed), it ** fails with an SQLITE_BUSY error. This is to stop RBU connections ** from automatically checkpointing a *-wal (or *-oal) file from within ** sqlite3_close(). *** 3d. In RBU_STAGE_CAPTURE mode, all xRead() calls on the wal file, and ** all xWrite() calls on the target database file perform no IO. ** Instead the frame and page numbers that would be read and written ** are recorded. Additionally, successful attempts to obtain exclusive ** xShmLock() WRITER, CHECKPOINTER and READ0 locks on the target ** database file are recorded. xShmLock() calls to unlock the same ** locks are no-ops (so that once obtained, these locks are never ** relinquished). Finally, calls to xSync() on the target database ** file fail with SQLITE_INTERNAL errors.

label: code-design

68143. Size of the record in bytes

68144. ** Clean up the VM after a single run.

68145. SQL statement 2 (see above)

68146. EVIDENCE-OF: R-55548-33817 The compile-time setting for URI filenames ** can be changed at start-time using the ** sqlite3_config(SQLITE_CONFIG_URI,1) or ** sqlite3_config(SQLITE_CONFIG_URI,0) configuration calls.

68147. ** Look through the list of open database files in db->aDb[] and if ** any have been closed, remove them from the list. Reallocate the ** db->aDb[] structure to a smaller size, if possible. ** ** Entry 0 (the "main" database) and entry 1 (the "temp" database) ** are never candidates for being collapsed.

68148. Either X and Y both have COLLATE operator or neither do

68149. The table whose content is at r[regBase]...

68150. 430

68151. ** The following structure keeps track of state information for the ** count() aggregate function.

68152. Add pVtab to the end of sqlite3.aVTrans

68153. If we reach this point, it means that execution is finished with ** an error of some kind.

68154. ** This "finder" function for VxWorks checks to see if posix advisory ** locking works. If it does, then that is what is used. If it does not ** work, then fallback to named semaphore locking.

68155. Columns of the index used

68156. ** Extract all tokens from hash table iHash and link them into a list ** in sorted order. The hash table is cleared before returning. It is ** the responsibility of the caller to free the elements of the returned ** list.

68157. ** The PmaReader passed as the first argument is guaranteed to be an ** incremental-reader (pReadr->pIncr!=0). This function serves to open ** and/or initialize the temp file related fields of the IncrMerge ** object at (pReadr->pIncr). ** ** If argument eMode is set to INCRINIT_NORMAL, then all PmaReaders ** in the sub-tree headed by pReadr are also initialized. Data is then ** loaded into the buffers belonging to pReadr and it is set to point to ** the first key in its range. ** ** If argument eMode is set to INCRINIT_TASK, then pReadr is guaranteed ** to be a multi-threaded PmaReader and this function is being called in a ** background thread. In this case all PmaReaders in the sub-tree are ** initialized as for INCRINIT_NORMAL and the aFile[1] buffer belonging to ** pReadr is populated. However, pReadr itself is not set up to point ** to its first key. A call to vdbePmaReaderNext() is still required to do ** that. ** ** The reason this function does not call vdbePmaReaderNext() immediately ** in the INCRINIT_TASK case is that vdbePmaReaderNext() assumes that it has ** to block on thread (pTask->thread) before accessing aFile[1]. But, since ** this entire function is being run by thread (pTask->thread), that will ** lead to the current background thread attempting to join itself. ** ** Finally, if argument eMode is set to INCRINIT_ROOT, it may be assumed ** that pReadr->pIncr is a multi-threaded IncrMerge objects, and that all ** child-trees have already been initialized using IncrInit(INCRINIT_TASK). ** In this case vdbePmaReaderNext() is called on all child PmaReaders and ** the current PmaReader set to point to the first key in its range. ** ** SQLITE_OK is returned if successful, or an SQLite error code otherwise.

68158. True if uppercase is equivalent to lowercase

68159. Size of azNotindexed[] array

68160. Figure out how large an allocation is required

68161. Full table scan

68162. Bounds of global page cache memory

68163. **comment:** ** Copy the error code and error message belonging to the Vdbe passed ** as the first argument to its database handle (so that they will be ** returned by calls to sqlite3_errcode() and sqlite3_errmsg()). ** ** This function does not clear the Vdbe error code or message, just ** copies them to the database handle.

label: code-design

68164. Cursor number of X in "X <op> <expr>"

68165. EVIDENCE-OF: R-18761-36601 There are three arguments to ** SQLITE_CONFIG_PAGECACHE: A pointer to 8-byte aligned memory (pMem), ** the size of each page cache line (sz), and the number of cache lines ** (N).

68166. **comment:** For following code (kept for historical record only) shows an ** unrolling for the 3- and 4-byte varint cases. This code is ** slightly faster, but it is also larger and much harder to test.

label: code-design

68167. 152

68168. The affinity of the column or 0 if not a column

68169. Instruction to jump to for RAISE(IGNORE)

68170. Opens a file, only if it exists.

68171. ** The code in this file is only used if we are compiling threadsafe ** under unix with pthreads. ** ** Note that this implementation requires a version of pthreads that ** supports recursive mutexes.

68172. register holding first column to insert

68173. Put a copy of the Table struct in Parse.pNewTable for the ** sqlite3AddColumn() function and friends to modify. But modify ** the name by adding an "sqlite_altertab_" prefix. By adding this ** prefix, we insure that the name will not collide with an existing ** table because user table are not allowed to have the "sqlite_" ** prefix on their name.

68174. 136

68175. Figure out the root-page that the lock should be held on. For table ** b-trees, this is just the root page of the b-tree being read or ** written. For index b-trees, it is the root page of the associated ** table.

68176. **comment:** FTS5_STMT_XXX constant

label: code-design

68177. SQLITE_CHECK_PAGES

68178. ** Implementation of the json_QUOTE(VALUE) function. Return a JSON value ** corresponding to the SQL value input. Mostly this means putting ** double-quotes around strings and returning the unquoted string "null" ** when given a NULL input.

68179. Score of best match

68180. ** TRUE if p is a lookaside memory allocation from db

68181. Instance of shared memory on this file

68182. Update contents of aTree[]

68183. Second disjunct

68184. VFS to use with this b-tree

68185. Total number of allocations

68186. Append the number of bytes of new data, then the term data itself ** to the page.

68187. **comment:** ** This function returns true if main-memory should be used instead of ** a temporary file for transient pager files and statement journals. ** The value returned depends on the value of db->temp_store (runtime ** parameter) and the compile time value of SQLITE_TEMP_STORE. The ** following table describes the relationship between these two values ** and this functions return value. ** ** SQLITE_TEMP_STORE db->temp_store Location of temporary database ** ----- ** 0 any file (return 0) ** 1 1 file (return 0) ** 1 2 memory (return 1) ** 1 0 file (return 0) ** 2 1 file (return 0) ** 2 2 memory (return 1) ** 2 0 memory (return 1) ** 3 any memory (return 1)

label: code-design

68188. ** Implementation of the sqlite_compileoption_used() function. ** The result is an integer that identifies if the compiler option ** was used to build SQLite.

68189. ** Obtain the mutex p. If successful, return SQLITE_OK. Otherwise, if another ** thread holds the mutex and it cannot be obtained, return SQLITE_BUSY.

68190. Replace Isalnum() return value with this

68191. If this is a commit, update the wal-index header too.

68192. Register pRt is used to store the memory required to save the state ** of the current program, and the memory required at runtime to execute ** the trigger program. If this trigger has been fired before, then pRt ** is already allocated. Otherwise, it must be initialized.

68193. ** The parser calls this routine for each token after the first token ** in an argument to the module name in a CREATE VIRTUAL TABLE statement.

68194. ** 2005 November 29 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains OS interface code that is common to all ** architectures.

68195. If no entry point was specified and the default legacy ** entry point name "sqlite3_extension_init" was not found, then ** construct an entry point name "sqlite3_X_init" where the X is ** replaced by the lowercase value of every ASCII alphabetic ** character in the filename after the last "/" upto the first ".", ** and eliding the first three characters if they are "lib". ** Examples: *** /usr/local/lib/libExample5.4.3.so ==> sqlite3_example_init ** C:/lib/mathfuncs.dll ==> sqlite3_mathfuncs_init

68196. Rtree handle

68197. OUT: Data for root node

68198. Index of the cell within the node

68199. 166

68200. ** Legal values for BtCursor.curFlags

68201. Current 'token' value

68202. For an SQLITE_ACCESS_EXISTS query, treat a zero-length file ** as if it does not exist.

68203. "view" or "table"

68204. # include "vdbeInt.h"

68205. **comment:** ** Clean up and delete a VDBE after execution. Return an integer which is ** the result code. Write any error message text into *pzErrMsg.
label: code-design

68206. ** Clear the PGHDR_NEED_SYNC flag from all dirty pages.

68207. Number of pending entries in the queue

68208. synopsis: r[P2]=P4' (len=P1)

68209. cmd ::= with insert_cmd INTO fullname idlist_opt select

68210. Some combination of MEM_Null, MEM_Str, MEM_Dyn, etc.

68211. 1450

68212. If pTab is really a view, make sure it has been initialized. ** ViewGetColumnNames() is a no-op if pTab is not a view.

68213. Buffer for matchinfo data

68214. ** Return the name of the table from which a result column derives. ** NULL is returned if the result column is an expression or constant or ** anything else which is not an unambiguous reference to a database column.

68215. Begin by generating some termination code at the end of the ** vdbe program

68216. ** PRAGMA [schema.] mmap_size(N) ** ** Used to set mapping size limit. The mapping size limit is ** used to limit the aggregate size of all memory mapped regions of the ** database file. If this parameter is set to zero, then memory mapping ** is not used at all. If N is negative, then the default memory map ** limit determined by sqlite3_config(SQLITE_CONFIG_MMAP_SIZE) is set. ** The parameter N is measured in bytes. ** ** This value is advisory. The underlying VFS is free to memory map ** as little or as much as it wants. Except, if N is set to 0 then the ** upper layers will never invoke the xFetch interfaces to the VFS.

68217. ccons ::= NOT NULL onconf

68218. The following assert statements verify that if this is a sharable ** b-tree database, the connection is holding the required table locks, ** and that no other connection has any open cursor that conflicts with ** this lock.

68219. INSERT

68220. Invoke the pre-update hook, if any

68221. Score for the new search point

68222. Pointer to shared header

68223. Parser context

68224. The affinity string for the record

68225. exprlist ::= cnearset

68226. Search for the required lock. Either a write-lock on root-page iTab, a ** write-lock on the schema table, or (if the client is reading) a ** read-lock on iTab will suffice. Return 1 if any of these are found.

68227. ** This function is called to allocate an array of Fts3Index structures ** representing the indexes maintained by the current FTS table. FTS tables ** always maintain the main "terms" index, but may also maintain one or ** more "prefix" indexes, depending on the value of the "prefix=" parameter ** (if any) specified as part of the CREATE VIRTUAL TABLE statement. ** ** Argument zParam is passed the value of the "prefix=" option if one was ** specified, or NULL otherwise. ** ** If no error occurs, SQLITE_OK is returned and *apIndex set to point to ** the allocated array. *pnIndex is set to the number of elements in the ** array. If an error does occur, an SQLite error code is returned. ** ** Regardless of whether or not an error is returned, it is the responsibility ** of the caller to call sqlite3_free() on the output array to free it.

68228. Local name of table

68229. Space for other PMAs

68230. Reinitialize the %_data table. This call creates the initial structure ** and averages records.

68231. **comment:** If the expression is not constant then we will need to ** disable the test that was generated above that makes sure ** this code only executes once. Because for a non-constant ** expression we need to rerun this code each time.
label: code-design

68232. **comment:** If all readers are using WAL_READ_LOCK(0) (in other words if no ** readers are currently using the WAL), then the transactions ** frames will overwrite the start of the existing log. Update the ** wal-index header to reflect this. ** ** In theory it would be Ok to update the cache of the header only ** at this point. But updating the actual wal-index header is also ** safe and means there is no special case for sqlite3WalUndo() ** to handle if this transaction is rolled back.
label: code-design

68233. if !OS_VXWORKS

68234. See the tool/mkopcodec.tcl script for details.

68235. Index to cksum (0..p->nIndex-1)

68236. **comment:** ** The select statement passed as the second parameter is a compound SELECT ** with an ORDER BY clause. This function allocates and returns a KeyInfo ** structure suitable for implementing the ORDER BY. ** ** Space to hold the KeyInfo structure is obtained from malloc. The calling ** function is responsible for ensuring that this structure is eventually ** freed.
label: code-design

68237. Hash key

68238. indexed_opt ::= INDEXED BY nm

68239. Zero the first nEqZero entries in the anEq[] array.

68240. ***** This function may be used as part of assert() statements only. ***** ** ** ** Return true if it would be illegal for pBtree to write into the ** table or index rooted at iRoot because other shared connections are ** simultaneously reading that same table or index. ** ** It is illegal for pBtree to write if some other Btree object that ** shares the same BtShared object is currently reading or writing ** the iRoot table. Except, if the other Btree object has the ** read-uncommitted flag set, then it is OK for the other object to ** have a read cursor. ** ** For example, before writing to any part of the table or index ** rooted at page iRoot, one should call: *** assert(!hasReadConflicts(pBtree, iRoot));

68241. ** Remove the memory data structures associated with the given ** Table. No changes are made to disk by this routine. ** ** This routine just deletes the data structure. It does not unlink ** the table data structure from the hash table. But it does destroy ** memory structures of the indices and foreign keys associated with ** the table. ** ** The db parameter is optional. It is needed if the Table object ** contains lookaside memory. (Table objects in the schema do not use ** lookaside memory, but some ephemeral Table objects do.) Or the ** db parameter can be used with db->pnBytesFreed to measure the memory ** used by the Table object.

68242. key_opt

68243. Set the flag to indicate that at least one PMA has been written. ** Or will be, anyhow.

68244. ** Walk the expression tree passed as the first argument. Return non-zero ** if the expression consists entirely of constants or copies of terms ** in pGroupBy that sort with the BINARY collation sequence. *** This routine is used to determine if a term of the HAVING clause can ** be promoted into the WHERE clause. In order for such a promotion to work, ** the value of the HAVING clause term must be the same for all members of ** a "group". The requirement that the GROUP BY term must be BINARY ** assumes that no other collating sequence will have a finer-grained ** grouping than binary. In other words (A=B COLLATE binary) implies ** A=B in every other collating sequence. The requirement that the ** GROUP BY be BINARY is stricter than necessary. It would also work ** to promote HAVING clauses that use the same alternative collating ** sequence as the GROUP BY term, but that is much harder to check, ** alternative collating sequences are uncommon, and this is only an ** optimization, so we take the easy way out and simply require the ** GROUP BY to use the BINARY collating sequence.

68245. Input string

68246. 80

68247. synopsis: r[P2]=rowid

68248. Jump to here if the sqlite3_interrupt() API sets the interrupt ** flag.

68249. ***** End of update.c *****

68250. BLOB

68251. Number of leaves on the trunk of the freelist

68252. If a non-unique index is used, or if a prefix of the key for ** unique index is used (making the index functionally non-unique) ** then the sqlite_stat1 data becomes important for scoring the ** plan

68253. Comparison of term and split-key

68254. lru page list

68255. STEP 2: ** Try to satisfy the allocation by carving a piece off of the end ** of the master chunk. This step usually works if step 1 fails.

68256. Header in shared memory

68257. End of processing for this SELECT

68258. Mutex for accessing the following:

68259. ** Reclaim all memory allocated by a JsonParse object. But do not ** delete the JsonParse object itself.

68260. The thread that will be using the new IncrMerger

68261. Opcode: Ne P1 P2 P3 P4 P5 ** Synopsis: IF r[P3]!=r[P1] *** This works just like the Eq opcode except that the jump is taken if ** the operands in registers P1 and P3 are not equal. See the Eq opcode for ** additional information. *** If both SQLITE_STOREP2 and SQLITE_KEEPNUL flags are set then the ** content of r[P2] is only changed if the new value is NULL or 1 (true). ** In other words, a prior r[P2] value will not be overwritten by 0 (false).

68262. ** Check invariants on a PgHdr entry. Return true if everything is OK. ** Return false if any invariant is violated. *** This routine is for use inside of assert() statements only. For ** example: *** assert(sqlite3PcachePageSanity(pPg));

68263. ** Grow the pWriter->aDlidx[] array to at least nLvl elements in size. ** Any new array elements are zeroed before returning.

68264. ***** Include sqlite3rbu.h in the middle of sqlite3rbu.c *****

68265. Has no INDEXED BY clause

68266. **comment:** Text of document to be inserted
label: documentation

68267. Estimated size of each table row in bytes

68268. * This is the maximum possible initial size of the Win32-specific heap, in * bytes.

68269. Array of trailing arguments

68270. The index entry must begin with a header size

68271. In-memory journal file

68272. The major token code number

68273. ** CAPI3REF: Maximum xShmLock index *** The xShmLock method on [sqlite3_io_methods] may use values ** between 0 and this upper bound as its "offset" argument. ** The SQLite core will never attempt to acquire or release a ** lock outside of this range

68274. Name of table or index being scanned

68275. ** This function does the work for the xUpdate method of FTS3 virtual ** tables. The schema of the virtual table being: *** CREATE TABLE <table name>(** <user columns>, ** <table name> HIDDEN, ** docid HIDDEN, ** <langid> HIDDEN **);***

68276. This call is to merge all segments at level iLevel. find the next ** available segment index at level iLevel+1. The call to ** fts3AllocateSegdirIdx() will merge the segments at level iLevel+1 to ** a single iLevel+2 segment if necessary.

68277. The BtShared this cursor points to

68278. Initialize the state of the random number generator once, ** the first time this routine is called. The seed value does ** not need to contain a lot of randomness since we are not ** trying to do secure encryption or anything like that... *** Nothing in this file or anywhere else in SQLite does any kind of ** encryption. The RC4 algorithm is being used as a PRNG (pseudo-random ** number generator) not as an encryption device.

68279. Used to loop thru the element list

68280. ***** End of expr.c *****

68281. ** Do both phases of a commit.

68282. **comment:** ** Allocate and initialize a new Pager object and put a pointer to it ** in *ppPager. The pager should eventually be freed by passing it ** to sqlite3PagerClose(). *** The zFilename argument is the path to the database file to open. ** If zFilename is NULL then a randomly-named temporary file is created ** and used as the file to be cached. Temporary files are be deleted ** automatically when they are closed. If zFilename is ":memory;" then ** all information is held in cache. It is never written to disk. ** This can be used to implement an in-memory database. *** The nExtra parameter specifies the number of bytes of space allocated ** along with each page reference. This space is available to the user ** via the sqlite3PagerGetExtra() API. When a new page is allocated, the ** first 8 bytes of this space are zeroed but the remainder is uninitialized. ** (The extra space is used by btree as the MemPage object.) *** The flags argument is used to specify properties that affect the ** operation of the pager. It should be passed some bitwise combination ** of the PAGER_* flags. ** The vfsFlags parameter is a bitmask to pass to the flags parameter ** of the xOpen() method of the supplied VFS when opening files. ** If the pager object is allocated and the specified file opened ** successfully, SQLITE_OK is returned and *ppPager set to point to ** the new pager object. If an error occurs, *ppPager is set to NULL ** and error code returned. This function may return SQLITE_NOMEM ** (sqlite3Malloc() is used to allocate memory), SQLITE_CANTOPEN or ** various SQLITE_IO_XXX errors.
label: code-design

68283. Deferred token object for this token

68284. ** 2003 September 6 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used for creating, destroying, and populating ** a VDBE (or an "sqlite3_stmt" as it is known to the outside world.)

68285. Pointer to output buffer

68286. ** Collation sequence comparison function. The pCtx argument points to ** a UCollator structure previously allocated using ucol_open().

68287. xWrite

68288. Shared locks never span more than one byte

68289. Report errors here

68290. Number of entries on one ptrmap page

68291. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains C code routines that are called by the parser ** to handle UPDATE statements.

68292. If no row was found and no error has occurred, then the %_content ** table is missing a row that is present in the full-text index. ** The data structures are corrupt.

68293. having_opt ::=

68294. Verify CHECK constraints

68295. Normal read/write connection

68296. ** Walk an expression tree. Return non-zero if the expression is constant ** and 0 if it involves variables or function calls. ** For the purposes of this function, a double-quoted string (ex: "abc") ** is considered a variable but a single-quoted string (ex: 'abc') is ** a constant.

68297. ** Return a pointer to the collation sequence that should be used by ** a binary comparison operator comparing pLeft and pRight. *** If the left hand expression has a collating sequence type, then it is ** used. Otherwise the collation sequence for the right hand expression ** is used, or the default (BINARY) if neither expression has a collating ** type. *** Argument pRight (but not pLeft) may be a null pointer. In this case, ** it is not considered.

68298. same as TK_ESCAPE, jump

68299. Index of right-most cell in left sibling

68300. True if the leaf of an intKey table

68301. **comment:** ** Adjust settings of the pager to those specified in the pgFlags parameter. *** The "level" in pgFlags & PAGER_SYNCHRONOUS_MASK sets the robustness ** of the database to damage due to OS crashes or power failures by ** changing the number of syncs() when writing the journals. ** There are four levels: *** OFF sqlite3OsSync() is never called. This is the default ** for temporary and transient files. *** NORMAL The journal is synced once before writes begin on the ** database. This is normally adequate protection, but ** it is theoretically possible, though very unlikely, ** that an inopportune power failure could leave the journal ** in a state which would cause damage to the database ** when it is rolled back. *** FULL The journal is synced twice before writes begin on the ** database (with some additional information - the nRec field ** of the journal header - being written in between the two ** syncs). If we assume that writing a ** single disk sector is atomic, then this mode provides ** assurance that the journal will not be corrupted to the ** point of causing damage to the database during rollback. *** EXTRA This is like FULL except that is also syncs the directory ** that contains the rollback journal after the rollback ** journal is unlinked. *** The above is for a rollback-journal mode. For WAL mode, OFF continues ** to mean that no syncs ever occur. NORMAL means that the WAL is synced ** prior to the start of checkpoint and that the database file is synced ** at the conclusion of the checkpoint if the entire content of the WAL ** was written back into the database. But no sync operations occur for ** an ordinary commit in NORMAL mode with WAL. FULL means that the WAL ** file is synced following each commit operation, in addition to the ** syncs associated with NORMAL. There is no difference between FULL ** and EXTRA for WAL mode. *** Do not confuse synchronous=FULL with SQLITE_SYNC_FULL. The ** SQLITE_SYNC_FULL macro means to use the Mac OSX-style full-fsync ** using fcnl(F_FULLFSYNC). SQLITE_SYNC_NORMAL means to do an ** ordinary fsync() call. There is no difference between SQLITE_SYNC_FULL ** and SQLITE_SYNC_NORMAL on platforms other than Mac OSX. But the ** synchronous=FULL versus synchronous=NORMAL setting determines when ** the xSync primitive is called and is relevant to all platforms. *** Numeric values associated with these states are OFF==1, NORMAL==2, ** and FULL==3.

label: code-design

68302. Legacy behavior (sqlite3_close()) behavior is to return ** SQLITE_BUSY if the connection can not be closed immediately.

68303. xSectorSize

68304. EVIDENCE-OF: R-55269-29598 The localtime_r() C function normally only ** works for years between 1970 and 2037. For dates outside this range, ** SQLite attempts to map the year into an equivalent year within this ** range, do the calculation, then map the year back.

68305. Jump to this label to skip a row

68306. Set nLoad4 to the value of (4^nOther) for the next iteration of the ** for-loop. Except, limit the value to 2^24 to prevent it from ** overflowing the 32-bit integer it is stored in.

68307. The simple case - no more than one database file (not counting the ** TEMP database) has a transaction active. There is no need for the ** master-journal. *** If the return value of sqlite3BtreeGetFilename() is a zero length ** string, it means the main database is :memory: or a temp file. In ** that case we do not support atomic multi-file commits, so use the ** simple case then too.

68308. 1220

68309. Affinity must be the same on all columns

68310. OUT. True if change is indirect

68311. At this point, okToChngToIN is true if original pTerm satisfies ** case 1. In that case, construct a new virtual term that is ** pTerm converted into an IN operator.

68312. Write the new expression node here

68313. Maximum of 4 fragments per snippet

68314. The first to increment to 1 does actual initialization

68315. #include "fts3Int.h"

68316. Only sort the current set of entries if they need it

68317. 171

68318. default size is approx 4 bytes

68319. Size of each scratch buffer

68320. ** Return true (non-zero) if we are running under WinNT, Win2K, WinXP, ** or WinCE. Return false (zero) for Win95, Win98, or WinME. *** Here is an interesting observation: Win95, Win98, and WinME lack ** the LockFileEx() API. But we can still statically link against that ** API as long as we don't call it when running Win95/98/ME. A call to ** this routine is used to determine if the host is Win95/98/ME or ** WinNT/2K/XP so that we will know whether or not we can safely call ** the LockFileEx() API.

68321. Expression list to resolve. May be NUL.

68322. Token:

68323. ** Deserialize and return the structure record currently stored in serialized ** form within buffer pData/nData. *** The Fts5Structure.aLevel[] and each Fts5StructureLevel.aSeg[] array ** are over-allocated by one slot. This allows the structure contents ** to be more easily edited. *** If an error occurs, *ppOut is set to NULL and an SQLite error code ** returned. Otherwise, *ppOut is set to point to the new object and ** SQLITE_OK returned.

68324. ** Helper function for fts3ExprIterate() (see below).

68325. ** Convert a schema pointer into the iDb index that indicates ** which database file in db->aDb[] the schema refers to. *** If the same database is attached more than once, the first ** attached database is returned.

68326. FTS index to query

68327. ** This function is a helper function for sqlite3Fts3EvalTestDeferred(). ** Assuming no error occurs or has occurred, It returns non-zero if the ** expression passed as the second argument matches the row that pCsr ** currently points to, or zero if it does not. *** If *pRc is not SQLITE_OK when this function is called, it is a no-op. ** If an error occurs during execution of this function, *pRc is set to ** the appropriate SQLite error code. In this case the returned value is ** undefined.

68328. ** Expression pExpr is one operand of a comparison operator that might ** be useful for indexing. This routine checks to see if pExpr appears ** in any index. Return TRUE (1) if pExpr is an indexed term and return ** FALSE (0) if not. If TRUE is returned, also set aiCurCol[0] to the cursor ** number of the table that is indexed and aiCurCol[1] to the column number ** of the column that is indexed, or XN_EXPR (-2) if an expression is being ** indexed. *** If pExpr is a TK_COLUMN column reference, then this routine always returns ** true even if that particular column is not indexed, because the column ** might be added to an automatic index later.

68329. IMPLEMENTATION-OF: R-62028-47212 All calls obtain an exclusive ** "checkpoint" lock on the database file.

68330. ** Set the "type" of an allocation.

68331. ** Print a single opcode. This routine is used for debugging only.

68332. Update or delete the input segments

68333. WinCE has no concept of a relative pathname, or so I am told.

68334. **comment:** As the winMutexInit() and winMutexEnd() functions are called as part ** of the sqlite3_initialize() and sqlite3_shutdown() processing, the ** "interlocked" magic used here is probably not strictly necessary.

label: code-design

68335. Current entry in aPage[]

68336. No affinity ever needs to be (or should be) applied to a value ** from the RHS of an "? IN (SELECT ...)" expression. The ** sqlite3FindInIndex() routine has already ensured that the ** affinity of the comparison has been applied to the value.

68337. Depth of the SAVEPOINT stack

68338. True if the transaction counter changed

68339. An AS clause always takes first priority

68340. **comment:** Set the text that the regular expression operates on to a NULL ** pointer. This is not really necessary, but it is tidier than ** leaving the regular expression object configured with an invalid ** pointer after this function returns.

label: code-design

68341. Number of columns in pIdx. "N"

68342. Key content for indexes. NULL for tables

68343. The key we are searching for

68344. _FTSINT_H

68345. ** Allocate an instance of the default tokenizer ("simple") at ** Fts5Config.pTokenizer. Return SQLITE_OK if successful, or an SQLite error ** code if an error occurs.
68346. result set is empty
68347. ** Return the pager associated with a BTree. This routine is used for ** testing and debugging only.
68348. **comment:** If this is a CREATE TABLE xx AS SELECT ..., execute the SELECT ** statement to populate the new table. The root-page number for the ** new table is in register pParse->regRoot. ** ** Once the SELECT has been coded by sqlite3Select(), it is in a ** suitable state to query for the column names and types to be used ** by the new table. ** ** A shared-cache write-lock is not required to write to the new table, ** as a schema-lock must have already been obtained to create it. Since ** a schema-lock excludes all other database users, the write-lock would ** be redundant.
label: code-design
68349. Open the master journal.
68350. If true, write ldidx to disk
68351. Opcode: Multiply P1 P2 P3 * * * Synopsis: r[P3]=r[P1]*r[P2] ** * * * Multiply the value in register P1 by the value in register P2 ** and store the result in register P3. ** If either input is NULL, the result is NULL.
68352. Schema containing the trigger
68353. Rebalance the expression. And check that its depth does not exceed ** SQLITE_FTS3_MAX_EXPR_DEPTH.
68354. ** Macros to enter and leave the PCache LRU mutex.
68355. attempt to get the lock
68356. Opcode: PrevIfOpen P1 P2 P3 P4 P5 * * * This opcode works just like Prev except that if cursor P1 is not ** open it behaves a no-op.
68357. IN/OUT: Output pointer
68358. The database
68359. Callback function
68360. ** Delete any previous value and set the value stored in *pMem to NULL. ** ** This routine calls the Mem.xDel destructor to dispose of values that ** require the destructor. But it preserves the Mem.zMalloc memory allocation. ** To free all resources, use sqlite3VdbeMemRelease(), which both calls this ** routine to invoke the destructor and deallocates Mem.zMalloc. ** ** Use this routine to reset the Mem prior to insert a new value. ** ** Use sqlite3VdbeMemRelease() to complete erase the Mem prior to abandoning it.
68361. cnearset ::= nearest
68362. CHAR
68363. ***** Continuing where we left off in wherecode.c *****
68364. ** Convert pMem so that it has types MEM_Real or MEM_Int or both. ** Invalidate any prior representations. ** ** Every effort is made to force the conversion, even if the input ** is a string that does not look completely like a number. Convert ** as much of the string as we can and ignore the rest.
68365. Grave-accent quoted symbols used by MySQL
68366. If a column is marked as 'hidden', omit it from the expanded ** result-set list unless the SELECT has the SF_IncludeHidden ** bit set.
68367. Prepared statement
68368. ** Add an opcode that includes the p4 value as an integer.
68369. xRename - rename the table
68370. Advance pChild until it points to iLast or laster
68371. Fall thru into the next case
68372. For a Real or Integer, use sqlite3_snprintf() to produce the UTF-8 ** string representation of the value. Then, if the required encoding ** is UTF-16le or UTF-16be do a translation. ** ** FIX ME: It would be better if sqlite3_snprintf() could do UTF-16.
68373. Add a single new term to an ExprList that is used to store a ** list of identifiers. Report an error if the ID list contains ** a COLLATE clause or an ASC or DESC keyword, except ignore the ** error while parsing a legacy schema.
68374. 1440
68375. ** 2006 Oct 10 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This header file is used by programs that want to link against the ** FTS3 library. All it does is declare the sqlite3Fts3Init() interface.
68376. When a new VdbeCursor is allocated, only the fields above are zeroed. ** The fields that follow are uninitialized, and must be individually ** initialized prior to first use.
68377. Number of locks in aTableLock
68378. Built-in unlikely() function
68379. OUT: uid to set on the file
68380. The fixer to be initialized
68381. size of db file in pages
68382. File handle for WAL file
68383. ** Argument pIn points to a character that is part of a nul-terminated ** string. Return a pointer to the first character following *pIn in ** the string that is not a white-space character.
68384. Disconnect these in next sqlite3_prepare()
68385. SELECT statement to query table pTab
68386. If we are creating a set for an "expr IN (SELECT ...)" construct, ** then there should be a single item on the stack. Write this ** item into the set table with bogus data.
68387. Used when p4type is P4_TABLE
68388. Number of children that must disable us
68389. Object fully contained within query region
68390. Initialize the upper limit on the number of worker threads
68391. WHERE analysis context
68392. ** This function may only be called from within a pre-update callback. ** It calculates a hash based on the primary key values of the old.* or ** new.* row currently available and, assuming no error occurs, writes it to ** *piHash before returning. If the primary key contains one or more NULL ** values, *pbNullPK is set to true before returning. ** ** If an error occurs, an SQLite error code is returned and the final values ** of *piHash and *pbNullPK are undefined. Otherwise, SQLITE_OK is returned ** and the output variables are set as described above.
68393. 273
68394. idx = (lwr+upr)/2;
68395. OUT: Expression element
68396. Special processing to add years
68397. At this point local variable rc holds the value that should be ** returned if this statement was compiled using the legacy ** sqlite3_prepare() interface. According to the docs, this can only ** be one of the values in the first assert() below. Variable p->rc ** contains the value that would be returned if sqlite3_finalize() ** were called on statement p.
68398. Set the limiting size of a WAL file.
68399. ** Assert that column iCol of statement pStmt is named zName.
68400. True for highlighted terms
68401. !defined(SQLITE_THREADS_IMPLEMENTED)
68402. ** CAPI3REF: Flags for the xShmLock VFS method ** ** These integer constants define the various locking operations ** allowed by the xShmLock method of [sqlite3_io_methods]. The ** following are the only legal combinations of flags to the ** xShmLock method: ** ** ** SQLITE_SHM_LOCK | SQLITE_SHM_SHARED ** SQLITE_SHM_LOCK | SQLITE_SHM_EXCLUSIVE ** SQLITE_SHM_UNLOCK | SQLITE_SHM_SHARED ** SQLITE_SHM_UNLOCK | SQLITE_SHM_EXCLUSIVE ** ** ** When unlocking, the same SHARED or EXCLUSIVE flag must be supplied as ** was given on the corresponding lock. ** ** The xShmLock method can transition between unlocked and SHARED or ** between unlocked and EXCLUSIVE. It cannot transition between SHARED ** and EXCLUSIVE.
68403. Needed for the definition of va_list
68404. Extract snippet from this column

68405. ** Read, deserialize and return the structure record. *** The Fts5Structure.aLevel[] and each Fts5StructureLevel.aSeg[] array ** are over-allocated as described for function fts5StructureDecode() ** above. *** If an error occurs, NULL is returned and an error code left in the ** Fts5Index handle. If an error has already occurred when this function ** is called, it is a no-op.

68406. Phrase object to advance to next docid

68407. ***** Begin file fts3_write.c *****

68408. Generate code to handle collisions

68409. Write the unqualified object name here

68410. Enable to print out how the built-in functions are hashed

68411. (275) ecmd ::= SEMI

68412. If another connection has written to the database file since the ** time the read transaction on this connection was started, then ** the write is disallowed.

68413. Index of a host parameter

68414. ** Allocate a single new register for use to hold some intermediate result.

68415. True if %_stat table exists (2==unknown)

68416. ***** End of fts3_tokenizer.c *****

68417. **comment:** not used 0x0010 // Was: SQLITE_IdxRealAsInt
label: code-design

68418. Invoke any destructors registered for collation sequence user data.

68419. ** Invoke the xFileControl method on a particular database.

68420. Declarations for functions in fkey.c. All of these are replaced by ** no-op macros if OMIT_FOREIGN_KEY is defined. In this case no foreign ** key functionality is available. If OMIT_TRIGGER is defined but ** OMIT_FOREIGN_KEY is not, only some of the functions are no-oped. In ** this case foreign keys are parsed, but no other functionality is ** provided (enforcement of FK constraints requires the triggers sub-system).

68421. The cost of visiting the index rows is N*K, where K is ** between 1.1 and 3.0, depending on the relative sizes of the ** index and table rows.

68422. 286

68423. Name of database (e.g. "main", "temp")

68424. SQLITE_OMIT_VIEW

68425. The triggered program

68426. Maximum instantaneous currentOut

68427. (295) ccons ::= NULL onconf

68428. the WHERE clause to be analyzed

68429. bOpenUri

68430. groupby_opt ::=

68431. Original list of tables

68432. ** Acquire a reader lock. ** Different API routines are called depending on whether or not this ** is Win9x or WinNT.

68433. Add the output column list to the name-context before parsing the ** other expressions in the SELECT statement. This is so that ** expressions in the WHERE clause (etc.) can refer to expressions by ** aliases in the result set. *** Minor point: If this is the case, then the expression will be ** re-evaluated for each reference to it.

68434. Is the PRIMARY KEY for the table

68435. ***** End of os_setup.h *****

68436. **comment:** The SELECT has been coded. If there is an error in the Parse structure, ** set the return code to 1. Otherwise 0.
label: code-design

68437. Draw vertical in column i if bLine[i] is true

68438. ** Restart the backup process. This is called when the pager layer ** detects that the database has been modified by an external database ** connection. In this case there is no way of knowing which of the ** pages that have been copied into the destination database are still ** valid and which are not, so the entire process needs to be restarted. *** It is assumed that the mutex associated with the BtShared object ** corresponding to the source database is held when this function is ** called.

68439. Allocated size of buffer zOut

68440. ** This is the xColumn method, called by SQLite to request a value from ** the row that the supplied cursor currently points to.

68441. Leaf cursor in SKIPNEXT state

68442. ** All token types in the generated fts5parse.h file are greater than 0.

68443. ** The afpLockingContext structure contains all afp lock specific state

68444. Slots used in azResult[], (nRow+1)*nColumn

68445. Value to compare to current sorter key

68446. Address of the initialization bypass jump

68447. ** The schema for each SQL table and view is represented in memory ** by an instance of the following structure.

68448. ** Flush any data stored in the in-memory hash tables to the database.

68449. **comment:** ** CAPI3REF: Open A BLOB For Incremental I/O ** METHOD: sqlite3 ** CONSTRUCTOR: sqlite3_blob *** ^^(This interfaces opens a [BLOB handle | handle] to the BLOB located ** in row iRow, column zColumn, table zTable in database zDb; ** in other words, the same BLOB that would be selected by: ** ** <pre> ** SELECT zColumn FROM zDb.zTable WHERE [rowid] = iRow; ** </pre>)^ ** ^^(Parameter zDb is not the filename that contains the database, but ** rather the symbolic name of the database. For attached databases, this is ** the name that appears after the AS keyword in the [ATTACH] statement. ** For the main database file, the database name is "main". For TEMP ** tables, the database name is "temp").^ ** ^If the flags parameter is non-zero, then the BLOB is opened for read ** and write access. ^If the flags parameter is zero, the BLOB is opened for ** read-only access. *** ^On success, [SQLITE_OK] is returned and the new [BLOB handle] is stored ** in *ppBlob. Otherwise an [error code] is returned and, unless the error ** code is SQLITE_MISUSE, *ppBlob is set to NULL.)^ ** This means that, provided ** the API is not misused, it is always safe to call [sqlite3_blob_close()] ** on *ppBlob after this function it returns. *** This function fails with SQLITE_ERROR if any of the following are true: ** ** ^^(Database zDb does not exist)^, ** ^^(Table zTable does not exist within database zDb)^, ** ^^(Table zTable is a WITHOUT ROWID table)^, ** ^^(Column zColumn does not exist)^, ** ^^(Row iRow is not present in the table)^, ** ^^(The specified column of row iRow contains a value that is not ** a TEXT or BLOB value)^, ** ^^(Column zColumn is part of an index, PRIMARY KEY or UNIQUE ** constraint and the blob is being opened for read/write access)^, ** ^^(foreign key constraints | Foreign key constraints) are enabled, ** column zColumn is part of a [child key] definition and the blob is ** being opened for read/write access)^. ** ** ^Unless it returns SQLITE_MISUSE, this function sets the ** [database connection] error code and message accessible via ** [sqlite3_errcode()] and [sqlite3errmsg()] and related functions. *** A BLOB referenced by sqlite3_blob_open() may be read using the ** [sqlite3_blob_read()] interface and modified by using ** [sqlite3_blob_write()]. The [BLOB handle] can be moved to a ** different row of the same table using the [sqlite3_blob_reopen()] ** interface. However, the column, table, or database of a [BLOB handle] ** cannot be changed after the [BLOB handle] is opened. *** ^If the row that a BLOB handle points to is modified by an ** [UPDATE], [DELETE], or by [ON CONFLICT] side-effects ** then the BLOB handle is marked as "expired". ** This is true if any column of the row is changed, even a column ** other than the one the BLOB handle is open on.)^ ** ^Calls to [sqlite3_blob_read()] and [sqlite3_blob_write()] for ** an expired BLOB handle fail with a return code of [SQLITE_ABORT]. ** ^Changes written into a BLOB prior to the BLOB expiring are not ** rolled back by the expiration of the BLOB. Such changes will eventually ** commit if the transaction continues to completion.)^ ** ^Use the [sqlite3_blob_bytes()] interface to determine the size of ** the opened blob. ^The size of a blob may not be changed by this ** interface. Use the [UPDATE] SQL command to change the size of a ** blob. *** ^The [sqlite3_bind_zeroblob()] and [sqlite3_result_zeroblob()] interfaces ** and the built-in [zeroblob] SQL function may be used to create a ** zero-filled blob to read or write using the incremental-blob interface. *** To avoid a resource leak, every open [BLOB handle] should eventually ** be released by a call to [sqlite3_blob_close()]. *** See also: [sqlite3_blob_close()], ** [sqlite3_blob_reopen()], [sqlite3_blob_read()], ** [sqlite3_blob_bytes()], [sqlite3_blob_write()].
label: code-design

68450. Illegal character

68451. as ::= AS nm

68452. Size of index key in bytes

68453. ***** Begin file keywordhash.h *****

68454. Figure out what action to take in case of a rowid collision

68455. Generate a subroutine to run when the results from select B ** are exhausted and only data in select A remains.

68456. Make sure the column name is unique. If the name is not unique, ** append an integer to the name so that it becomes unique.
68457. 250
68458. VDBE_DISPLAY_P4
68459. Result code from Lock/UnlockFileEx()
68460. The table from which we should delete things
68461. Delete the contents of the %_data and %_docsizes tables.
68462. Control characters are not allowed in strings
68463. ** Check to see if the GetVersionEx[AW] functions are deprecated on the ** target system. GetVersionEx was first deprecated in Win8.1.
68464. Structure version number (currently 3)
68465. ** Change the size to which the WAL file is truncated on each reset.
68466. ** Reset the schema for the database at index iDb. Also reset the ** TEMP schema.
68467. If we reach this point it means that either p[] should be overwritten ** with pTemplate[] if p[] exists, or if p==NULL then allocate a new ** WhereLoop and insert it.
68468. Range of registers allocated for aCol and aFunc
68469. If there is no statement handle, then the blob-handle has ** already been invalidated. Return SQLITE_ABORT in this case.
68470. No longer required OpenEphemeral instr.
68471. ** For the index called zIdxName which is found in the database iDb, ** unlike that index from its Table then remove the index from ** the index hash table and free all memory structures associated ** with the index.
68472. Associated database connection
68473. 102
68474. Check that plter->iSwitchRowid is set correctly.
68475. (iCol<<32) + iPos
68476. If SUBSTR_COMPATIBILITY is defined then substr(X,0,N) work the same as ** as substr(X,1,N) - it returns the first N characters of X. This ** is essentially a back-out of the bug-fix in check-in [5fc125d362df4b8] ** from 2009-02-02 for compatibility of applications that exploited the ** old buggy behavior.
68477. Absolute level to repack
68478. First byte of gap between cell pointers and cell content
68479. **comment:** ** A copy of the following object occurs in the wal-index immediately ** following the second copy of the WalIndexHdr. This object stores ** information used by checkpoint. ** ** nBackfill is the number of frames in the WAL that have been written ** back into the database. (We call the act of moving content from WAL to ** database "backfilling".) The nBackfill number is never greater than ** WalIndexHdr.mxFrame. nBackfill can only be increased by threads ** holding the WAL_CKPT_LOCK lock (which includes a recovery thread). ** However, a WAL_WRITE_LOCK thread can move the value of nBackfill from ** mxFrame back to zero when the WAL is reset. ** ** nBackfillAttempted is the largest value of nBackfill that a checkpoint ** has attempted to achieve. Normally nBackfill==nBackfillAttempted, however ** the nBackfillAttempted is set before any backfilling is done and the ** nBackfill is only set after all backfilling completes. So if a checkpoint ** crashes, nBackfillAttempted might be larger than nBackfill. The ** WalIndexHdr.mxFrame must never be less than nBackfillAttempted. ** ** The aLock[] field is a set of bytes used for locking. These bytes should ** never be read or written. ** ** There is one entry in aReadMark[] for each reader lock. If a reader ** holds read-lock K, then the value in aReadMark[K] is no greater than ** the mxFrame for that reader. The value READMARK_NOT_USED (0xffffffff) ** for any aReadMark[] means that entry is unused. aReadMark[0] is ** a special case; its value is never used and it exists as a place-holder ** to avoid having to offset aReadMark[] index by one. Readers holding ** WAL_READ_LOCK(0) always ignore the entire WAL and read all content ** directly from the database. ** ** The value of aReadMark[K] may only be changed by a thread that ** is holding an exclusive lock on WAL_READ_LOCK(K). Thus, the value of ** aReadMark[K] cannot change while there is a reader is using that mark ** since the reader will be holding a shared lock on WAL_READ_LOCK(K). ** ** The checkpointer may only transfer frames from WAL to database where ** the frame numbers are less than or equal to every aReadMark[] that is ** in use (that is, every aReadMark[j] for which there is a corresponding ** WAL_READ_LOCK(j)). New readers (usually) pick the aReadMark[] with the ** largest value and will increase an unused aReadMark[] to mxFrame if there ** is not already an aReadMark[] equal to mxFrame. The exception to the ** previous sentence is when nBackfill equals mxFrame (meaning that everything ** in the WAL has been backfilled into the database) then new readers ** will choose aReadMark[0] which has value 0 and hence such reader will ** get all their all content directly from the database file and ignore ** the WAL. ** ** Writers normally append new frames to the end of the WAL. However, ** if nBackfill equals mxFrame (meaning that all WAL content has been ** written back into the database) and if no readers are using the WAL ** (in other words, if there are no WAL_READ_LOCK(i) where i>0) then ** the writer will first "reset" the WAL back to the beginning and start ** writing new content beginning at frame 1. ** ** We assume that 32-bit loads are atomic and so no locks are needed in ** order to read from any aReadMark[] entries.
label: code-design
68480. ** Functions to deserialize a 16 bit integer, 32 bit real number and ** 64 bit integer. The serialized value is returned.
68481. Discard the scheme and authority segments of the URI.
68482. 700
68483. SQLITE_UNTESTABLE
68484. Minimum legal index key size
68485. ** PRAGMA busy_timeout = N ** ** Call sqlite3_busy_timeout(db, N). Return the current timeout value ** if one is set. If no busy handler or a different busy handler is set ** then 0 is returned. Setting the busy_timeout to 0 or negative ** disables the timeout.
68486. The cursor of the corresponding table
68487. **comment:** A place to hold %extra_argument
label: code-design
68488. 340
68489. ** This function is called from both BtreeCommitPhaseTwo() and BtreeRollback() ** at the conclusion of a transaction.
68490. ** Run the integrity-check. If no error occurs and the current contents of ** the FTS index are correct, return SQLITE_OK. Or, if the contents of the ** FTS index are incorrect, return SQLITE_CORRUPT_VTAB. ** ** Or, if an error (e.g. an OOM or IO error) occurs, return an SQLite ** error code. ** ** The integrity-check works as follows. For each token and indexed token ** prefix in the document set, a 64-bit checksum is calculated (by code ** in fts3ChecksumEntry()) based on the following: ** ** + The index number (0 for the main index, 1 for the first prefix ** index etc.), ** + The token (or token prefix) text itself, ** + The language-id of the row it appears in, ** + The docid of the row it appears in, ** + The column it appears in, and ** + The tokens position within that column. ** ** The checksums for all entries in the index are XORed together to create ** a single checksum for the entire index. ** ** The integrity-check code calculates the same checksum in two ways: ** ** 1. By scanning the contents of the FTS index, and ** 2. By scanning and tokenizing the content table. ** ** If the two checksums are identical, the integrity-check is deemed to have ** passed.
68491. Sizes of deleted documents
68492. end of the 'extern "C"' block
68493. Array of registers where record is assembled
68494. ePragFlg:
68495. The cell index. First cell is 0
68496. Opcode: ParseSchema P1 ** P4 * ** * Read and parse all entries from the SQLITE_MASTER table of database P1 ** that match the WHERE clause P4. ** ** This opcode invokes the parser to create a new virtual machine, ** then runs the new virtual machine. It is thus a re-entrant opcode.
68497. Cursor open on index being analyzed
68498. ** This macro evaluates to true if either the update hook or the preupdate ** hook are enabled for database connect DB.
68499. ***** Begin file opcodes.h *****
68500. RENAME => ID
68501. Check to see if the column is in one of the tables in the FROM ** clause of the aggregate query
68502. **comment:** Number of pages per doc loaded
label: documentation
68503. ***** End of fts3_aux.c *****
68504. ** The input pointer currently points to the second byte of a table-header. ** Specifically, to the following: ** ** + number of columns in table (varint) ** + array of PK flags (1 byte per column), ** + table name (nul terminated). ** ** This function ensures that all of the above is present in the input ** buffer (i.e. that it can be accessed without any calls to xInput()). ** If successful, SQLITE_OK is returned. Otherwise, an SQLite error code. ** The input pointer is not moved.
68505. An ORDER BY (or GROUP BY) clause, or NULL

68506. A malformed database page might cause us to read past the end ** of page when parsing a cell. *** The following block of code checks early to see if a cell extends ** past the end of a page boundary and causes SQLITE_CORRUPT to be ** returned if it does.

68507. OP_DeferredSeek always uses a single rowid

68508. The reference count on pShmNode has already been incremented under ** the cover of the unixEnterMutex() mutex and the pointer from the ** new (struct unixShm) object to the pShmNode has been set. All that is ** left to do is to link the new object into the linked list starting ** at pShmNode->pFirst. This must be done while holding the pShmNode->mutex ** mutex.

68509. True if pSub is the right side of a LEFT JOIN

68510. ** Table *p is a virtual table. This function removes the VTable object ** for table *p associated with database connection db from the linked ** list in p->pVTab. It also decrements the VTable ref count. This is ** used when closing database connection db to free all of its VTable ** objects without disturbing the rest of the Schema object (which may ** being used by other shared-cache connections).

68511. ONEPASS_OFF, _SINGLE, or _MULTI. See above

68512. 179

68513. SQL statement used to access %_content

68514. ** Call sqlite3WalOpen() to open the WAL handle. If the pager is in ** exclusive-locking mode when this function is called, take an EXCLUSIVE ** lock on the database file and use heap-memory to store the wal-index ** in. Otherwise, use the normal shared-memory.

68515. If the cursor cache is stale (meaning it is not currently point at ** the correct row) then bring it up-to-date by doing the necessary ** B-Tree seek.

68516. True if a write cursor

68517. Store the result in pMerger->aTree[iOut]

68518. ** Get/set the locking-mode for this pager. Parameter eMode must be one ** of PAGER_LOCKINGMODE_QUERY, PAGER_LOCKINGMODE_NORMAL or ** PAGER_LOCKINGMODE_EXCLUSIVE. If the parameter is not _QUERY, then ** the locking-mode is set to the value specified. *** The returned value is either PAGER_LOCKINGMODE_NORMAL or ** PAGER_LOCKINGMODE_EXCLUSIVE, indicating the current (possibly updated) ** locking-mode.

68519. **comment:** SQLITE_VDBE_COVERAGE
label: test

68520. Lock the destination database, if it is not locked already.

68521. Make sure this is not an attempt to ALTER a view.

68522. SQLITE OMIT_EXPLAIN

68523. If there is insufficient space allocated at StrBuffer.z, use realloc() ** to grow the buffer until so that it is big enough to accomodate the ** appended data.

68524. A table in the database

68525. Parent tokenizer instance

68526. Set the pointer-map entry for the new sibling page.

68527. 70

68528. ** End of the routinely-changing class members *****

68529. ** Record the fact that the schema cookie will need to be verified ** for database iDb. The code to actually verify the schema cookie ** will occur at the end of the top-level Vdbe and will be generated ** later, by sqlite3FinishCoding().

68530. Nothing to do at end-of-loop for a single-pass

68531. Acquire a RESERVED lock

68532. True if a shared lock has ever been held

68533. ** Convert pMem so that it is of type MEM_Real. ** Invalidate any prior representations.

68534. Number of leaf pages left to write

68535. Complete SELECT tree

68536. SQLITE OMIT_UTF16

68537. If the parent table is the same as the child table, and we are about ** to increment the constraint-counter (i.e. this is an INSERT operation), ** then check if the row being inserted matches itself. If so, do not ** increment the constraint-counter. *** If any of the parent-key values are NULL, then the row cannot match ** itself. So set JUMPIFNULL to make sure we do the OP_Found if any ** of the parent-key values are NULL (at this point it is known that ** none of the child key values are).

68538. ** The following is executed when the parser accepts

68539. Number of buckets in the hash table

68540. ** Generate code for an UPDATE of a virtual table. *** There are two possible strategies - the default and the special ** "onepass" strategy. Onepass is only used if the virtual table ** implementation indicates that pWhere may match at most one row. *** The default strategy is to create an ephemeral table that contains ** for each row to be changed: *** (A) The original rowid of that row. ** (B) The revised rowid for the row. ** (C) The content of every column in the row. *** Then loop through the contents of this ephemeral table executing a ** VUpdate for each row. When finished, drop the ephemeral table. *** The "onepass" strategy does not use an ephemeral table. Instead, it ** stores the same values (A, B and C above) in a register array and ** makes a single invocation of VUpdate.

68541. Set idxFlags flags for the ORDER BY clause

68542. Resolve names in the result set.

68543. ** Add seg-reader objects to the Fts3MultiSegReader object passed as the ** 8th argument. *** This function returns SQLITE_OK if successful, or an SQLite error code ** otherwise.

68544. raisetype := ROLLBACK

68545. ** CAPI3REF: Run-Time Limit Categories ** KEYWORDS: {limit category} {*limit categories} *** These constants define various performance limits ** that can be lowered at run-time using [sqlite3_limit|]. ** The synopsis of the meanings of the various limits is shown below. ** Additional information is available at [limits | Limits in SQLite]. *** <dl> ** [[SQLITE_LIMIT_LENGTH]] ^<dt>SQLITE_LIMIT_LENGTH</dt> ** <dd>The maximum size of any string or BLOB or table row, in bytes.<dd>^ ** [[SQLITE_LIMIT_SQL_LENGTH]] ^<dt>SQLITE_LIMIT_SQL_LENGTH</dt> ** <dd>The maximum length of an SQL statement, in bytes.</dd>^ ** [[SQLITE_LIMIT_COLUMN]] ^<dt>SQLITE_LIMIT_COLUMN</dt> ** <dd>The maximum number of columns in a table definition or in the ** result set of a [SELECT] or the maximum number of columns in an index ** or in an ORDER BY or GROUP BY clause.</dd>^ ** [[SQLITE_LIMIT_EXPR_DEPTH]] ^<dt>SQLITE_LIMIT_EXPR_DEPTH</dt> ** <dd>The maximum depth of the parse tree on any expression.</dd>^ ** [[SQLITE_LIMIT_COMPOUND_SELECT]] ^<dt>SQLITE_LIMIT_COMPOUND_SELECT</dt> ** <dd>The maximum number of terms in a compound SELECT statement.</dd>^ ** [[SQLITE_LIMIT_VDBE_OP]] ^<dt>SQLITE_LIMIT_VDBE_OP</dt> ** <dd>The maximum number of instructions in a virtual machine program ** used to implement an SQL statement. If [sqlite3_prepare_v2|] or ** the equivalent tries to allocate space for more than this many opcodes ** in a single prepared statement, an SQLITE_NOMEM error is returned.</dd>^ ** [[SQLITE_LIMIT_FUNCTION_ARG]] ^<dt>SQLITE_LIMIT_FUNCTION_ARG</dt> ** <dd>The maximum number of arguments on a function.</dd>^ ** [[SQLITE_LIMIT_ATTACHED]] ^<dt>SQLITE_LIMIT_ATTACHED</dt> ** <dd>The maximum number of [ATTACH | attached databases].</dd> ** <dd>*** [[SQLITE_LIMIT_PATTERN_LENGTH]] ** ^<dt>SQLITE_LIMIT_PATTERN_LENGTH</dt> ** <dd>The maximum length of the pattern argument to the [LIKE] or ** [GLOB] operators.</dd>^ ** [[SQLITE_LIMIT_VARIABLE_NUMBER]] ** ^<dt>SQLITE_LIMIT_VARIABLE_NUMBER</dt> ** <dd>The maximum index number of any [parameter] in an SQL statement.)^ ** [[SQLITE_LIMIT_TRIGGER_DEPTH]] ^<dt>SQLITE_LIMIT_TRIGGER_DEPTH</dt> ** <dd>The maximum depth of recursion for triggers.</dd>^ ** [[SQLITE_LIMIT_WORKER_THREADS]] ^<dt>SQLITE_LIMIT_WORKER_THREADS</dt> ** <dd>The maximum number of auxiliary worker threads that a single ** [prepared statement] may start.</dd>^ ** </dl>

68546. the FROM clause

68547. True if this is an OUTER join

68548. OUT: Size of WAL log in frames

68549. ** Close an existing SQLite database

68550. Loop through all candidate snippets. Store the best snippet in ** *pFragment. Store its associated 'score' in iBestScore.

68551. If sqlite3FindInIndex() did not find or create an index that is ** suitable for evaluating the IN operator, then evaluate using a ** sequence of comparisons. *** This is step (1) in the in-operator.md optimized algorithm.

68552. **comment:** Comparison against ArraySize-1 since we hold back one extra slot ** as a contingency. In other words, never need more than 3 overflow ** slots but 4 are allocated, just to be safe.

68553. **label:** code-design

68553. If this is a non-composite (single column) foreign key, check if it ** maps to the INTEGER PRIMARY KEY of table pParent. If so, leave *ppIdx ** and *paiCol set to zero and return early. *** Otherwise, for a composite foreign key (more than one column), allocate ** space for the aiCol array (returned via output parameter *paiCol). ** Non-composite foreign keys do not require the aiCol array.

68554. 20

68555. FTS5_TOKENIZE_* flags

68556. Analyze a term that is composed of two or more subterms connected by ** an OR operator.

68557. Check that the CTE name is unique within this WITH clause. If ** not, store an error in the Parse structure.

68558. Create new entry if true and does not otherwise exist

68559. ***** End of vdbeInt.h *****

68560. Index of required stats in anEq[] etc.

68561. Name passed to create_module()

68562. SQL used to determine nLeafEst

68563. ** If a Win32 native heap has been configured, this function will attempt to ** destroy and recreate it. If the Win32 native heap is not isolated and/or ** the sqlite3_memory_used() function does not return zero, SQLITE_BUSY will ** be returned and no changes will be made to the Win32 native heap.

68564. The WAL file is readonly

68565. ** 2007 June 22 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This is part of an SQLite module implementing full-text search. ** This particular file implements the generic tokenizer interface.

68566. 30

68567. Use full fsync on the backend

68568. **comment:** Prevent endless loops on corrupt database files
label: code-design

68569. with ::= WITH RECURSIVE wqlist

68570. Initial key value

68571. Run through the tokenchars. Fold them into the output buffer along ** the way.

68572. The two journal files

68573. Iterator to skip through changeset

68574. An non-ascii-range character. Fold it into the output buffer if ** it is a token character, or break out of the loop if it is not.

68575. Write index of pSrc->a[] here

68576. Loop counter

68577. One for each phrase

68578. 25

68579. Initialize the local lockdata

68580. True if low on PAGECACHE memory

68581. Net deferred constraints this transaction.

68582. Name of database file to open

68583. 162

68584. ** Mark a data page as writeable. This routine must be called before ** making changes to a page. The caller must check the return value ** of this function and be careful not to change any page data unless ** this routine returns SQLITE_OK. *** The difference between this function and pager_write() is that this ** function also deals with the special case where 2 or more pages ** fit on a single disk sector. In this case all co-resident pages ** must have been written to the journal file before returning. *** If an error occurs, SQLITE_NOMEM or an IO error code is returned ** as appropriate. Otherwise, SQLITE_OK.

68585. Second change record

68586. Remove pParent from the original tree.

68587. 0x000008 // available for use

68588. Iterator to run through pList with.

68589. The page and offset from which the current term was read. The offset ** is the offset of the first rowid in the current doclist.

68590. Allocated cursor

68591. Pragma Name 1st arg or NULL

68592. OUT: Set to true if PK row is found

68593. If the original WHERE clause is z of the form: (x1 OR x2 OR ...) AND y ** Then for every term xN, evaluate as the subexpression: xN AND z ** That way, terms in y that are factored into the disjunction will ** be picked up by the recursive calls to sqlite3WhereBegin() below. *** Actually, each subexpression is converted to "xN AND w" where w is ** the "interesting" terms of z - terms that did not originate in the ** ON or USING clause of a LEFT JOIN, and terms that are usable as ** indices. *** This optimization also only applies if the (x1 OR x2 OR ...) term ** is not contained in the ON clause of a LEFT JOIN. *** See ticket <http://www.sqlite.org/src/info/f2369304e4>

68594. Get a NULL terminated version of the new table name.

68595. Create a new pager cache. ** Under memory stress, invoke xStress to try to make pages clean. ** Only clean and unpinned pages can be reclaimed.

68596. If true, jump if either operand is NULL

68597. Set up the local name-context to pass to sqlite3ResolveExprNames() to ** resolve the result-set expression list.

68598. True to dequote

68599. Case 1.

68600. EVIDENCE-OF: R-28375-38319 SQLite will never request a scratch buffer ** that is more than 6 times the database page size.

68601. ** This routine generates VDBE code that causes a single row of a ** single table to be deleted. Both the original table entry and ** all indices are removed. *** Preconditions: *** 1. iDataCur is an open cursor on the btree that is the canonical data ** store for the table. (This will be either the table itself, ** in the case of a rowid table, or the PRIMARY KEY index in the case ** of a WITHOUT ROWID table.) *** 2. Read/write cursors for all indices of pTab must be open as ** cursor number iIdxCur+i for the i-th index. *** 3. The primary key for the row to be deleted must be stored in a ** sequence of nPk memory cells starting at iPk. If nPk==0 that means ** that a search record formed from OP_MakeRecord is contained in the ** single memory location iPk. *** eMode: ** Parameter eMode may be passed either ONEPASS_OFF (0), ONEPASS_SINGLE, or ** ONEPASS_MULTI. If eMode is not ONEPASS_OFF, then the cursor ** iDataCur already points to the row to delete. If eMode is ONEPASS_OFF ** then this function must seek iDataCur to the entry identified by iPk ** and nPk before reading from it. *** If eMode is ONEPASS_MULTI, then this call is being made as part ** of a ONEPASS delete that affects multiple rows. In this case, if ** iIdxNoSeek is a valid cursor number (>=0) and is not the same as ** iDataCur, then its position should be preserved following the delete ** operation. Or, if iIdxNoSeek is not a valid cursor number, the ** position of iDataCur should be preserved instead. *** iIdxNoSeek: ** If iIdxNoSeek is a valid cursor number (>=0) not equal to iDataCur, ** then it identifies an index cursor (from within array of cursors ** starting at iIdxCur) that already points to the index entry to be deleted. ** Except, this optimization is disabled if there are BEFORE triggers since ** the trigger body might have moved the cursor.

68602. ** Turn parser tracing on by giving a stream to which to write the trace ** and a prompt to preface each trace message. Tracing is turned off ** by making either argument NULL ** ** Inputs: ** ** A FILE* to which trace output should be written. ** If NULL, then tracing is turned off. ** A prefix string written at the beginning of every ** line of trace output. If NULL, then tracing is ** turned off. ** ** ** Outputs: ** None.

68603. Unable to open the database file

68604. Schema containing the table

68605. 1150

68606. This statement is used to determine which level to read the input from ** when performing an incremental merge. It returns the absolute level number ** of the oldest level in the db that contains at least ? segments. Or, ** if no level in the FTS index contains more than ? segments, the statement ** returns zero rows.

68607. ** The TableLock structure is only used by the sqlite3TableLock() and ** codeTableLocks() functions.

68608. OUT: Error message, if any

68609. ***** Begin file sqliteicu.h *****

68610. for auto-named local lock file, just check the host ID and we'll ** use the local lock file path that's already in there

68611. Context used by sqlite3WhereXXX()

68612. SQLITE_STATUS_MALLOC_COUNT

68613. Name of ICU function that failed
68614. Prevents a race condition. Ticket #3537
68615. ** Return the number of rows that were changed. If this routine is ** generating code because of a call to sqlite3NestedParse(), do not ** invoke the callback function.
68616. Number of scratch buffers
68617. Use a rollback journal on this file
68618. Mask of WO_xx values describing operator
68619. ** Extract a value from the supplied expression in the manner described ** above sqlite3ValueFromExpr(). Allocate the sqlite3_value object ** using valueNew().
*** If pCtx is NULL and an error occurs after the sqlite3_value object ** has been allocated, it is freed before returning. Or, if pCtx is not ** NULL, it is assumed that the caller will free any allocated object ** in all cases.
68620. Status verb
68621. ** Allocate nByte bytes of space from within the B-Tree page passed ** as the first argument. Write into *pIdx the index into pPage->aData[] ** of the first byte of allocated space. Return either SQLITE_OK or ** an error code (usually SQLITE_CORRUPT). ** ** The caller guarantees that there is sufficient space to make the ** allocation. This routine might need to defragment in order to bring ** all the space together, however. This routine will avoid using ** the first two bytes past the cell pointer area since presumably this ** allocation is being made in order to insert a new cell, so we will ** also end up needing a new cell pointer.
68622. ***** Continuing where we left off in fts3.c *****
68623. Error message from sqlite3ParseUri()
68624. If memory status is enabled, then the malloc.c wrapper will already ** hold the STATIC_MEM mutex when the routines here are invoked.
68625. Comparison function
68626. Last possible cell or freeblock offset
68627. IN/OUT: Right input list
68628. Expression opcode
68629. ** Return the number of pages still to be backed up as of the most recent ** call to sqlite3_backup_step().
68630. ** Invoke sqlite3_declare_vtab() to declare the schema for the FTS3 table ** passed as the first argument. This is done as part of the xConnect() ** and xCreate()
methods. *** If *pRc is non-zero when this function is called, it is a no-op. ** Otherwise, if an error occurs, an SQLite error code is stored in *pRc ** before
returning.
68631. Arguments to table-valued-function
68632. Set zIn to point at the start of the input buffer and zTerm to point 1 ** byte past the end. *** Variable zOut is set to point at the output buffer, space obtained **
from sqlite3_malloc().
68633. ** Destroy the private VFS created for the rru handle passed as the only ** argument by an earlier call to rruCreateVfs().
68634. Name of a table or index
68635. Both or neither are NULL
68636. Destructor function
68637. When to reset OP_Once counters
68638. How to handle constraint errors
68639. Size of all column names (incl. 0x00)
68640. Index (main or aPrefix[] entry)
68641. If the statement is of the form *** INSERT INTO <table1> SELECT * FROM <table2>; ** ** Then special optimizations can be applied that make the transfer
** very fast and which reduce fragmentation of indices. *** This is the 2nd template.
68642. **comment:** If pSchema is NULL, then return -1000000. This happens when code in ** expr.c is trying to resolve a reference to a transient table (i.e. one ** created
by a sub-select). In this case the return value of this ** function should never be used. *** We return -1000000 instead of the more usual -1 simply because using
** -1000000 as the incorrect index into db->aDb[] is much ** more likely to cause a segfault than -1 (of course there are assert() ** statements too, but it never
hurts to play the odds).
label: code-design
68643. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not
evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains C code routines that are called by the parser ** in
order to generate code for DELETE FROM statements.
68644. Flags to pass to open()
68645. xCreate - create a table
68646. Pointer to array of page numbers
68647. **comment:** ** Create a new PCache object. Storage space to hold the object ** has already been allocated and is passed in as the p pointer. ** The caller discovers
how much space needs to be allocated by ** calling sqlite3PcacheSize(). *** szExtra is some extra space allocated for each page. The first ** 8 bytes of the extra
space will be zeroed as the page is allocated, ** but remaining content will be uninitialized. Though it is opaque ** to this module, the extra space really ends up
being the MemPage ** structure in the pager.
label: code-design
68648. The index of the database the expression refers to
68649. Fsync at this offset
68650. If this is an INSERT operation, or an UPDATE that modifies the rowid ** value, then this operation requires constraint handling. ** ** If the on-conflict mode is
REPLACE, this means that the existing row ** should be deleted from the database before inserting the new row. Or, ** if the on-conflict mode is other than
REPLACE, then this method must ** detect the conflict and return SQLITE_CONSTRAINT before beginning to ** modify the database file.
68651. True if this change is "indirect"
68652. We are trying for an exclusive lock but another thread in this ** same process is still holding a shared lock.
68653. 560
68654. ** Insert cell pCell into node pNode. Node pNode is the head of a ** subtree iHeight high (leaf nodes have iHeight==0).
68655. This happens if the segment is being used as an input to an incremental ** merge and all data has already been "trimmed". See function ** fts5TrimSegments() for
details. In this case leave the iterator empty. ** The caller will see the (pIter->pLeaf==0) and assume the iterator is ** at EOF already.
68656. (283) cmd ::= create_table create_table_args
68657. ** Implementation of the last_insert_rowid() SQL function. The return ** value is the same as the sqlite3_last_insert_rowid() API function.
68658. Source table
68659. Covering index scans
68660. Percent symbol. %
68661. Opening a db handle. Fourth parameter is passed 0.
68662. If the caller is an UPDATE or DELETE statement that is requesting ** to use a one-pass algorithm, determine if this is appropriate.
68663. ** Compute the Year, Month, and Day from the julian day number.
68664. ** The first argument passed to this function is a serial-type that ** corresponds to an integer - all values between 1 and 9 inclusive ** except 7. The second points
to a buffer containing an integer value ** serialized according to serial_type. This function deserializes ** and returns the value.
68665. ** Allowed values for et_info.flags
68666. VDBE has completed execution
68667. Allocate a new SessionChange object. Ensure that the aRecord[] ** buffer of the new object is large enough to hold any record that ** may be generated by
combining the input records.
68668. ** Set the bit in the IntegrityCk.aPgRef[] array that corresponds to page iPg.
68669. Next info block in a list of them all
68670. ** Apply the changeset passed via xInput/pIn to the main database ** attached to handle "db". Invoke the supplied conflict handler callback ** to resolve any
conflicts encountered while applying the change.
68671. DELETE_CONTENT
68672. Acquire an EXCLUSIVE lock
68673. 209

68674. Are LHS fields reordered?
68675. ** Return a pointer to the Pager.nExtra bytes of "extra" space ** allocated along with the specified page.
68676. ** Mark a single data page as writeable. The page is written into the ** main journal or sub-journal as required. If the page is written into ** one of the journals, the corresponding bit is set in the ** Pager.pInJournal bitvec and the PagerSavepoint.pInSavepoint bitvecs ** of any open savepoints as appropriate.
68677. ** Forward declaration to account for the circular dependency between ** functions fts3SegmentMerge() and fts3AllocateSegdirIdx().
68678. INITIALLY
68679. DEBUG and TESTING functions
68680. User is requesting the value of the special column with the same name ** as the table. Return the cursor integer id number. This value is only ** useful in that it may be passed as the first argument to an FTS5 ** auxiliary function.
68681. ** Determine if any of the arguments to the pExpr Function reference ** pSrcList. Return true if they do. Also return true if the function ** has no arguments or has only constant arguments. Return false if pExpr ** references columns but not columns of tables found in pSrcList.
68682. Wal file descriptor for this main db
68683. sqlite3_test_control(SQLITE_TESTCTRL_SCRATCHMALLOC, sz, &pNew, pFree); ** ** Pass pFree into sqlite3ScratchFree(). ** If sz>0 then allocate a scratch buffer into pNew.
68684. OUT: Pointer to page number array
68685. ***** 3rd ***** 4th *****
68686. Discard the contents of the pending-terms hash table.
68687. ** Deallocate all chunks from a RowSet. This frees all memory that ** the RowSet has allocated over its lifetime. This routine is ** the destructor for the RowSet.
68688. ** If there is a "-oal" file in the file-system corresponding to the ** target database in the file-system, delete it. If an error occurs, ** leave an error code and error message in the rbu handle.
68689. Insert is likely an append
68690. ** CAPI3REF: Virtual Table Indexing Information ** KEYWORDS: sqlite3_index_info ** ** The sqlite3_index_info structure and its substructures is used as part ** of the [virtual table] interface to ** pass information into and receive the reply from the [xBestIndex] ** method of a [virtual table module]. The fields under **Inputs** are the ** inputs to xBestIndex and are read-only. xBestIndex inserts its ** results into the **Output** fields. ** ** ^The aConstraint[] array records WHERE clause constraints of the form: ** ** <blockquote>column OP expr</blockquote> ** ** where OP is =, <, <=, >, or >=.)^^(The particular operator is ** stored in aConstraint[].op using one of the ** [SQLITE_INDEX_CONSTRAINT_EQ | SQLITE_INDEX_CONSTRAINT_VALUES].)^^(The index of the column is stored in ** aConstraint[].iColumn.^^(aConstraint[],usable is TRUE if the ** expr on the right-hand side can be evaluated (and thus the constraint ** is usable) and false if it cannot.)^** ** ^The optimizer automatically inverts terms of the form "expr OP column" ** and makes other simplifications to the WHERE clause in an attempt to ** get as many WHERE clause terms into the form shown above as possible. ** ^The aConstraint[] array only reports WHERE clause terms that are ** relevant to the particular virtual table being queried. ** ** ^Information about the ORDER BY clause is stored in aOrderBy[], ** ^Each term of aOrderBy records a column of the ORDER BY clause. ** ** The colUsed field indicates which columns of the virtual table may be ** required by the current scan. Virtual table columns are numbered from ** zero in the order in which they appear within the CREATE TABLE statement ** passed to sqlite3_declare_vtab(). For the first 63 columns (columns 0-62), ** the corresponding bit is set within the colUsed mask if the column may be ** required by SQLite. If the table has at least 64 columns and any column ** to the right of the first 63 is required, then bit 63 of colUsed is also ** set. In other words, column iCol may be required if the expression ** (colUsed & ((sqlite3_uint64)1 << (iCol>=63 ? 63 : iCol))) evaluates to ** non-zero. ** ** The [xBestIndex] method must fill aConstraintUsage[] with information ** about what parameters to pass to xFilter. ^If argvIndex>0 then ** the right-hand side of the corresponding aConstraint[] is evaluated ** and becomes the argvIndex-th entry in argv. ^If aConstraintUsage[],omit ** is true, then the constraint is assumed to be fully handled by the ** virtual table and is not checked again by SQLite.)^** ** ^The idxNum and idxPtr values are recorded and passed into the ** [xFilter] method. ** ^[sqlite3_free()] is used to free idxPtr if and only if ** needToFreeIdxPtr is true. ** ** ^The orderByConsumed means that output from [xFilter]/[xNext] will occur in ** the correct order to satisfy the ORDER BY clause so that no separate ** sorting step is required. ** ** ^The estimatedCost value is an estimate of the cost of a particular ** strategy. A cost of N indicates that the cost of the strategy is similar ** to a linear scan of an SQLite table with N rows. A cost of log(N) ** indicates that the expense of the operation is similar to that of a ** binary search on a unique indexed field of an SQLite table with N rows. ** ** ^The estimatedRows value is an estimate of the number of rows that ** will be returned by the strategy. ** ** The xBestIndex method may optionally populate the idxFlags field with a ** mask of SQLITE_INDEX_SCAN_* flags. Currently there is only one such flag - ** SQLITE_INDEX_SCAN_UNIQUE. If the xBestIndex method sets this flag, SQLite ** assumes that the strategy may visit at most one row. ** ** Additionally, if xBestIndex sets the SQLITE_INDEX_SCAN_UNIQUE flag, then ** SQLite also assumes that if a call to the xUpdate() method is made as ** part of the same statement to delete or update a virtual table row and the ** implementation returns SQLITE_CONSTRAINT, then there is no need to rollback ** any database changes. In other words, if the xUpdate() returns ** SQLITE_CONSTRAINT, the database contents must be exactly as they were ** before xUpdate was called. By contrast, if SQLITE_INDEX_SCAN_UNIQUE is not ** set and xUpdate returns SQLITE_CONSTRAINT, any database changes made by ** the xUpdate method are automatically rolled back by SQLite. ** ** IMPORTANT: The estimatedRows field was added to the sqlite3_index_info ** structure for SQLite [version 3.8.2] ([dateof:3.8.2]). ** If a virtual table extension is ** used with an SQLite version earlier than 3.8.2, the results of attempting ** to read or write the estimatedRows field are undefined (but are likely ** to included crashing the application). The estimatedRows field should ** therefore only be used if [sqlite3_libversion_number()] returns a ** value greater than or equal to 3008002. Similarly, the idxFlags field ** was added for [version 3.9.0] ([dateof:3.9.0]). ** It may therefore only be used if ** sqlite3_libversion_number() returns a value greater than or equal to ** 3009000.
68691. OK to spill pager cache
68692. Expression tree height of current sub-select
68693. xShmMap method
68694. Blob handle open on %_segments table
68695. ** Step the cursor to the back to the previous entry in the database. ** Return values: ** ** SQLITE_OK success ** SQLITE_DONE the cursor is already on the first element of the table ** otherwise some kind of error occurred ** ** The main entry point is sqlite3BtreePrevious(). That routine is optimized ** for the common case of merely decrementing the cell counter BtCursor.aiIdx ** to the previous cell on the current page. The (slower) btreePrevious() ** helper routine is called when it is necessary to move to a different page ** or to restore the cursor. ** ** If bit 0x01 of the F argument to sqlite3BtreePrevious(C,F) is 1, then ** the cursor corresponds to an SQL index and this routine could have been ** skipped if the SQL index had been a unique index. The F argument is a ** hint to the implement. The native SQLite btree implementation does not ** use this hint, but COMDB2 does.
68696. Level for the new search point
68697. 98
68698. PRAGMA synchronous=FULL
68699. Buffer to write page data to
68700. Register any built-in FTS5 module before loading the automatic ** extensions. This allows automatic extensions to register FTS5 ** tokenizers and auxiliary functions.
68701. name
68702. Create a label to jump to when we want to abort the query
68703. The Index
68704. Parent row data starts here
68705. OUT: Columns for imposter table
68706. Id of cursor to find
68707. **comment:** ** Return TRUE if the innermost loop of the WHERE clause implementation ** returns rows in ORDER BY order for complete run of the inner loop. ** ** Across multiple iterations of outer loops, the output rows need not be ** sorted. As long as rows are sorted for just the innermost loop, this ** routine can return TRUE.
 label: code-design
68708. Function context
68709. TK_LT
68710. Opcode: NotNull P1 P2 * * * ** Synopsis: if r[P1]!=NULL goto P2 ** ** Jump to P2 if the value in register P1 is not NULL.
68711. ***** End of rowset.c *****
68712. Preallocated, blank file handle
68713. ** Create a new virtual database engine.
68714. ** An instance of the following structure is used to hold information ** about a cell. The parseCellPtr() function fills in this structure ** based on information extract from the raw disk page.

68715. Get the local shared locks
68716. Height of the tree headed by this node
68717. Array of page numbers
68718. Result returned by the thread
68719. Delete the index entries associated with the current record.
68720. Must be element [3]
68721. HAVE_ISNAN
68722. The column the token must match
68723. Size of buffer z in bytes
68724. OP_Column only used for length()
68725. **comment:** ** Attempt to reduce the value of the WalCkptInfo.nBackfillAttempted ** variable so that older snapshots can be accessed. To do this, loop ** through all wal frames from nBackfillAttempted to (nBackfill+1), ** comparing their content to the corresponding page with the database ** file, if any. Set nBackfillAttempted to the frame number of the ** first frame for which the wal file content matches the db file. *** This is only really safe if the file-system is such that any page ** writes made by earlier checkpointers were atomic operations, which ** is not always true. It is also possible that nBackfillAttempted ** may be left set to a value larger than expected, if a wal frame ** contains content that duplicate of an earlier version of the same ** page. *** SQLITE_OK is returned if successful, or an SQLite error code if an ** error occurs. It is not an error if nBackfillAttempted cannot be ** decreased at all.
label: code-design
68726. Note: this cast is safe, because the origin data point was an int ** that was cast to a (const char *).
68727. First register holding data before packing
68728. Nul-terminated string containing boolean
68729. Handle compound SELECT statements using the separate multiSelect() ** procedure.
68730. LCS for the current iterator positions
68731. 230
68732. Added by 3.4.1
68733. Tokenizer name
68734. OUT: Configuration object
68735. argv[0] passed to function
68736. Size of the current data field
68737. ** Indexes for use with Pager.aStat[]. The Pager.aStat[] array contains ** the values accessed by passing SQLITE_DBSTATUS_CACHE_HIT, CACHE_MISS ** or CACHE_WRITE to sqlite3_db_status().
68738. Array of subprograms
68739. Generate code to report an FK violation to the caller.
68740. ** The minimum amount of free space that we have seen.
68741. Generate a subroutine that outputs the current row of the B ** select as the next output row of the compound select.
68742. Permissions to create file with
68743. ** The main routine for vdbePmaReaderIncrMergeInit() operations run in ** background threads.
68744. Vdbe pre-update hook is invoked by
68745. MATCHINFO
68746. All prior WhereLoops are order-distinct
68747. EVIDENCE-OF: R-37839-54301 Value is a big-endian 24-bit ** two's-complement integer.
68748. Hash of page content
68749. Cursor for a table whose size needs checking
68750. same as TK_NOTNULL, jump, in1
68751. Error code from preupdate_new/old
68752. Fsync the WAL header if true
68753. ** No-op versions of all memory allocation routines
68754. EQP id of right-hand query
68755. Result of prior MovetoUnpacked() call
68756. IN/OUT: Increment this value by PMA size
68757. Opcode of pRight
68758. ** Create a temporary file name and store the resulting pointer into pzBuf. ** The pointer returned in pzBuf must be freed via sqlite3_free().
68759. Bytes of PMA space required
68760. aLevel[] array
68761. ** Allocate and return new r-tree node. Initially, (RtreeNode.iNode==0), ** indicating that node has not yet been assigned a node number. It is ** assigned a node number when nodeWrite() is called to write the ** node contents out to the database.
68762. Name of database containing pTab
68763. Indexes with default row estimates should not have stat1 data
68764. The wrFlag argument to sqlite3BtreeCursor()
68765. PRAGMA secure_delete is enabled
68766. The main parser program. ** The first argument is a pointer to a structure obtained from ** "sqlite3ParserAlloc" which describes the current state of the parser. ** The second argument is the major token number. The third is ** the minor token. The fourth optional argument is whatever the ** user wants (and specified in the grammar) and is available for ** use by the action routines. *** Inputs: ** ** A pointer to the parser (an opaque structure.) ** The major token number. ** The minor token number. ** An option argument of a grammar-specified type. ** *** Outputs: ** None.
68767. BLOB(k), VARCHAR(k), CHAR(k) -> r=(k/4+1)
68768. ** Create a new symbolic label for an instruction that has yet to be ** coded. The symbolic label is really just a negative number. The ** label can be used as the P2 value of an operation. Later, when ** the label is resolved to a specific address, the VDBE will scan ** through its operation list and change all values of P2 which match ** the label into the resolved address. *** The VDBE knows that a P2 value is a label because labels are ** always negative and P2 values are suppose to be non-negative. ** Hence, a negative P2 value is a label that has yet to be resolved. *** Zero is returned if a malloc() fails.
68769. op==TK_REGISTER && p->pTab!=0 happens when pExpr was originally ** a TK_COLUMN but was previously evaluated and cached in a register
68770. Values to bind to statement
68771. Mask of configuration flags
68772. **comment:** Size of temporary buffer
label: code-design
68773. Entry point. Use "sqlite3_extension_init" if 0
68774. Seed the search with a single WherePath containing zero WhereLoops. *** TUNING: Do not let the number of iterations go above 28. If the cost ** of computing an automatic index is not paid back within the first 28 ** rows, then do not use the automatic index.
68775. z[] is now the stemmed word in reverse order. Flip it back ** around into forward order and return.
68776. Storage module to write to
68777. ** CUSTOM AUXILIARY FUNCTIONS *****
68778. FT5S backend
68779. ** Implementation of FT5S xRename method. Rename an fts5 table.
68780. Offset of first rowid on leaf
68781. ** CAPI3REF: Changeset Iterator Handle
68782. Held in memory not obtained from malloc()
68783. Restriction (5)
68784. Authenticated as an administrator
68785. ** Create a "unicode61" tokenizer.
68786. Allocate space for the Fts3MultiSegReader.aCsr[] array

68787. Write the extra pages and truncate the database file as required
68788. ** Return true if the current thread holds the database connection ** mutex and all required BtShared mutexes. *** This routine is used inside assert() statements only.
68789. Special case - the last token of the snippet is also the last token ** of the column. Append any punctuation that occurred between the end ** of the previous token and the end of the document to the output. ** Then break out of the loop.
68790. ** CAPI3REF: Enable Or Disable Extension Loading ** METHOD: sqlite3 *** ^So as not to open security holes in older applications that are ** unprepared to deal with [extension loading], and as a means of disabling ** [extension loading] while evaluating user-entered SQL, the following API ** is provided to turn the [sqlite3_load_extension()] mechanism on and off. *** ^Extension loading is off by default. ** ^Call the sqlite3_enable_load_extension() routine with onoff==1 ** to turn extension loading on and call it with onoff==0 to turn ** it back off again. *** ^This interface enables or disables both the C-API ** [sqlite3_load_extension()] and the SQL function [load_extension()]. ** ^Use [sqlite3_db_config](db, [SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION],...) ** to enable or disable only the C-API.)^ *** Security warning: It is recommended that extension loading ** be disabled using the [SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION] method ** rather than this interface, so the [load_extension()] SQL function ** remains disabled. This will prevent SQL injections from giving attackers ** access to extension loading capabilities.
68791. ** Delete the named file. *** Note that Windows does not allow a file to be deleted if some other ** process has it open. Sometimes a virus scanner or indexing program ** will open a journal file shortly after it is created in order to do ** whatever it does. While this other process is holding the ** file open, we will be unable to delete it. To work around this ** problem, we delay 100 milliseconds and try to delete again. Up ** to MX_DELETION_ATTEMPTS deletion attempts are run before giving ** up and returning an error.
68792. ** Bind a blob value to an SQL statement variable.
68793. ** CAPI3REF: Impose A Limit On Heap Size *** ^The sqlite3_soft_heap_limit64() interface sets and/or queries the ** soft limit on the amount of heap memory that may be allocated by SQLite. ** ^SQLite strives to keep heap memory utilization below the soft heap ** limit by reducing the number of pages held in the page cache ** as heap memory usages approaches the limit. ** ^The soft heap limit is "soft" because even though SQLite strives to stay ** below the limit, it will exceed the limit rather than generate ** an [SQLITE_NOMEM] error. In other words, the soft heap limit ** is advisory only. *** ^The return value from sqlite3_soft_heap_limit64() is the size of ** the soft heap limit prior to the call, or negative in the case of an ** error. ^If the argument N is negative ** then no change is made to the soft heap limit. Hence, the current ** size of the soft heap limit can be determined by invoking ** sqlite3_soft_heap_limit64() with a negative argument. *** ^If the argument N is zero then the soft heap limit is disabled. *** ^The soft heap limit is not enforced in the current implementation ** if one or more of following conditions are true: *** ** The soft heap limit is set to zero. ** Memory accounting is disabled using a combination of the ** [sqlite3_config](SQLITE_CONFIG_MEMSTATUS,...) start-time option and ** the [SQLITE_DEFAULT_MEMSTATUS] compile-time option. ** An alternative page cache implementation is specified using ** [sqlite3_config](SQLITE_CONFIG_PCACHE2,...). ** The page cache allocates from its own memory pool supplied ** by [sqlite3_config](SQLITE_CONFIG_PAGECACHE,...) rather than ** from the heap. ** ^ *** Beginning with SQLite [version 3.7.3] ([dateof:3.7.3]), ** the soft heap limit is enforced ** regardless of whether or not the [SQLITE_ENABLE_MEMORY_MANAGEMENT] ** compile-time option is invoked. With [SQLITE_ENABLE_MEMORY_MANAGEMENT], ** the soft heap limit is enforced on every memory allocation. Without ** [SQLITE_ENABLE_MEMORY_MANAGEMENT], the soft heap limit is only enforced ** when memory is allocated by the page cache. Testing suggests that because ** the page cache is the predominate memory user in SQLite, most ** applications will achieve adequate soft heap limit enforcement without ** the use of [SQLITE_ENABLE_MEMORY_MANAGEMENT]. *** ^The circumstances under which SQLite will enforce the soft heap limit may ** changes in future releases of SQLite.
68794. UNION
68795. Prior to version 3.1.2, when LIMIT and OFFSET had to be simple constants, ** not arbitrary expressions, we allowed some combining of LIMIT and OFFSET ** because they could be computed at compile-time. But when LIMIT and OFFSET ** became arbitrary expressions, we were forced to add restrictions (13) ** and (14).
68796. ** Constant tokens for values 0 and 1.
68797. ***** Begin Unique File ID Utility Used By VxWorks ***** On most versions of unix, we can get a unique ID for a file by concatenating ** the device number and the inode number. But this does not work on VxWorks. ** On VxWorks, a unique file id must be based on the canonical filename. *** ^A pointer to an instance of the following structure can be used as a ** unique file ID in VxWorks. Each instance of this structure contains ** a copy of the canonical filename. There is also a reference count. ** The structure is reclaimed when the number of pointers to it drops to ** zero. *** ^There are never very many files open at one time and lookups are not ** a performance-critical path, so it is sufficient to put these ** structures on a linked list.
68798. Miscellaneous flags. See below
68799. String pointer
68800. SQLITE_OMIT_SUBQUERY
68801. only re-establish the shared lock if necessary
68802. ** Attempt to reallocate p. If the reallocation fails, then free p ** and set the mallocFailed flag in the database connection.
68803. phrase
68804. Size of scratch memory requested
68805. ** Maximum depth of an SQLite B-Tree structure. Any B-Tree deeper than ** this will be declared corrupt. This value is calculated based on a ** maximum database size of 2^31 pages a minimum fanout of 2 for a ** root-node and 3 for all other internal nodes. *** ^If a tree that appears to be taller than this is encountered, it is ** assumed that the database is corrupt.
68806. ** If FTS_LOG_MERGES is defined, call sqlite3_log() to report each automatic ** and incremental merge operation that takes place. This is used for ** debugging FTS only, it should not usually be turned on in production ** systems.
68807. End non-truncate path
68808. Register holding results
68809. OUT: changeset (left <concat> right)
68810. flags controlling this file
68811. Program Counter in parent (calling) frame
68812. Cell content
68813. Pointer map page data
68814. ** Flush any data in the pending-terms hash table to disk. If successful, ** merge all segments in the database (including the new segment, if ** there was any data to flush) into a single segment.
68815. If there is both a GROUP BY and an ORDER BY clause and they are ** identical, then it may be possible to disable the ORDER BY clause ** on the grounds that the GROUP BY will cause elements to come out ** in the correct order. It also may not - the GROUP BY might use a ** database index that causes rows to be grouped together as required ** but not actually sorted. Either way, record the fact that the ** ORDER BY and GROUP BY clauses are the same by setting the orderByGrp ** variable.
68816. ** Get/set the size-limit used for persistent journal files. *** ^Setting the size limit to -1 means no limit is enforced. ** An attempt to set a limit smaller than -1 is a no-op.
68817. These routines are used inside assert() statements only.
68818. ** Return non-zero if a read (or write) transaction is active.
68819. Preference to built-in funcs
68820. Case 4.
68821. Do not create or use a rollback journal
68822. True to iterate in reverse order
68823. ** Walk an expression tree. Return 1 if the expression contains a ** subquery of some kind. Return 0 if there are no subqueries.
68824. Remove the table entry from SQLite's internal schema and modify ** the schema cookie.
68825. Check if the requested node is already in the hash table. If so, ** increase its reference count and return it.
68826. Real value of left operand
68827. Memory cell to set to string value
68828. **comment:** Grow the two buffers to pgosz + padding bytes in size.
 label: code-design
68829. Info on how to code the ORDER BY clause
68830. ***** End of printf.c *****

68831. Figure out the ON CONFLICT policy that will be used for this step ** of the trigger program. If the statement that caused this trigger ** to fire had an explicit ON CONFLICT, then use it. Otherwise, use ** the ON CONFLICT policy that was specified as part of the trigger ** step statement. Example: ** ** CREATE TRIGGER AFTER INSERT ON t1 BEGIN; ** INSERT OR REPLACE INTO t2 VALUES(new.a, new.b); ** END; ** ** INSERT INTO t1 ... ; -- insert into t2 uses REPLACE policy ** INSERT OR IGNORE INTO t1 ... ; -- insert into t2 uses IGNORE policy

68832. "VIEW" or "TABLE"

68833. 20

68834. Plan with no prereqs and no IN(...) seen

68835. Expression text

68836. OUT: Result of comparison

68837. #ifndef SQLITE_OMIT_SUBQUERY

68838. !defined(__SQLITESESSION_H_) && defined(SQLITE_ENABLE_SESSION)

68839. Information on the ORDER BY clause

68840. Load the extension into this database connection

68841. Index of column in pTab

68842. 180

68843. The name context

68844. SelectDest targetting the Queue table

68845. ** Check to make sure the temporary directory ends with an appropriate ** separator. If it does not and there is not enough space left to add ** one, fail.

68846. If the ending matches this... (Reversed)

68847. Read the database schema. If an error occurs, leave an error message ** and code in pParse and return NULL.

68848. ** Remove all records associated with segment iSegid.

68849. Tables have different CHECK constraints. Ticket #2252

68850. synopsis: r[P2]=r[P1]

68851. OUT: Value read from the nRec field

68852. If an existing hash entry cannot be found, create a new one.

68853. Name of the VFS to use

68854. Initialize any AUTOINCREMENT data structures required.

68855. ** Gather the results for matchinfo directives 'y' and 'b'.

68856. Array to populate

68857. All SELECT results must be columns.

68858. ** The expression is the default value for the most recently added column ** of the table currently under construction. ** ** Default value expressions must be constant. Raise an exception if this ** is not the case. ** ** This routine is called by the parser while in the middle of ** parsing a CREATE TABLE statement.

68859. ***** Begin file sqlite3rbu.h *****

68860. If the xMutexAlloc method has not been set, then the user did not ** install a mutex implementation via sqlite3_config() prior to ** sqlite3_initialize() being called. This block copies pointers to ** the default implementation into the sqlite3GlobalConfig structure.

68861. Number of the page to fetch

68862. NOT

68863. Current value

68864. TUNING: unlikely() probability is 0.0625. likely() is 0.9375

68865. Value of the argument and schema

68866. Path of file (possibly) being created

68867. First replace any existing entry. ** ** Actually, the way the column cache is currently used, we are guaranteed ** that the object will never already be in cache. Verify this guarantee.

68868. **comment:** ** Change the P2 operand of instruction addr so that it points to ** the address of the next instruction to be coded.
label: documentation

68869. Number of pending OP_SqlExec opcodes

68870. If zName is the not the name of a table in the schema created using ** CREATE, then check to see if it is the name of an virtual table that ** can be an eponymous virtual table.

68871. Offset to write

68872. Always returns pCsr->sPoint

68873. ** This routine is called once for each row in the result table. Its job ** is to fill in the TabResult structure appropriately, allocating new ** memory as necessary.

68874. size of the input

68875. Schema of database iDb

68876. Passing NULL to HeapFree is undefined.

68877. Statement to query %_docsize

68878. sqlite3_mutex_methods*

68879. ** At least one instance of the following structure is created for each ** trigger that may be fired while parsing an INSERT, UPDATE or DELETE ** statement. All such objects are stored in the linked list headed at ** Parse.pTriggerPrg and deleted once statement compilation has been ** completed. ** ** A Vdbe sub-program that implements the body and WHEN clause of trigger ** TriggerPrg.pTrigger, assuming a default ON CONFLICT clause of ** TriggerPrg.orconf, is stored in the TriggerPrg.pProgram variable. ** The Parse.pTriggerPrg list never contains two entries with the same ** values for both pTrigger and orconf. ** ** The TriggerPrg.aColmask[0] variable is set to a mask of old.* columns ** accessed (or set to 0 for triggers fired as a result of INSERT ** statements). Similarly, the TriggerPrg.aColmask[1] variable is set to ** a mask of new.* columns used by the program.

68880. True for patchsets

68881. Offset of header on pPg

68882. Number of pointers to this structure

68883. Value of P1 operand

68884. Pointer to the cell text.

68885. Instruction pcx is the OP_Program that invoked the sub-program ** currently being halted. If the p2 instruction of this OP_Halt ** instruction is set to OE_Ignore, then the sub-program is throwing ** an IGNORE exception. In this case jump to the address specified ** as the p2 of the calling OP_Program.

68886. ** CAPI3REF: User Data For Functions ** METHOD: sqlite3_context ** ** ^The sqlite3_user_data() interface returns a copy of ** the pointer that was the pUserData parameter (the 5th parameter) ** of the [sqlite3_create_function()] ** and [sqlite3_create_function16()] routines that originally ** registered the application-defined function. ** ** This routine must be called from the same thread in which ** the application-defined function is running.

68887. The number of == or IN constraints to code

68888. Size of term suffix

68889. Allocate a new page. This page will become the right-sibling of ** pPage. Make the parent page writable, so that the new divider cell ** may be inserted. If both these operations are successful, proceed.

68890. rc==0 here means that one or both of the keys ran out of fields and ** all the fields up to that point were equal. Return the default_rc ** value.

68891. Increment the xRowid value

68892. Index of the database to use

68893. ** Interpret the argument as a unicode codepoint. If the codepoint ** is an upper case character that has a lower case equivalent, ** return the codepoint corresponding to the lower case version. ** Otherwise, return a copy of the argument. ** ** The results are undefined if the value passed to this function ** is less than zero.

68894. Also used in P2 (not P5) of OP_Delete

68895. **comment:** ** Parameter zMaster is the name of a master journal file. A single journal ** file that referred to the master journal file has just been rolled back. ** This routine checks if it is possible to delete the master journal file, ** and does so if it is. ** ** Argument zMaster may point to Pager.pTmpSpace. So that buffer is not ** available for use within this function. ** ** When a master journal file is created, it is populated with the names ** of all of its child journals, one after another, formatted as utf-8 ** encoded text. The end of each child journal file is marked with a ** nul-terminator byte (0x00). i.e. the entire contents of a master journal ** file for a transaction involving two databases might be: ** ** "/home/bill/a.db-journal\x00/home/bill/b.db-journal\x00" ** ** A master journal file may

only be deleted once all of its child ** journals have been rolled back. *** This function reads the contents of the master-journal file into ** memory and loops through each of the child journal names. For ** each child journal, it checks if: *** * if the child journal exists, and if so ** * if the child journal contains a reference to master journal ** file zMaster *** If a child journal can be found that matches both of the criteria ** above, this function returns without doing anything. Otherwise, if ** no such child journal can be found, file zMaster is deleted from ** the file-system using sqlite3OsDelete(). *** If an IO error within this function, an error code is returned. This ** function allocates memory by calling sqlite3Malloc(). If an allocation ** fails, SQLITE_NOMEM is returned. Otherwise, if no IO or malloc errors ** occur, SQLITE_OK is returned. *** TODO: This function allocates a single block of memory to load ** the entire contents of the master journal file. This could be ** a couple of kilobytes or so - potentially larger than the page ** size.

label: code-design

68896. Pointer to sub-list content

68897. Database file handle

68898. EVIDENCE-OF: R-28312-64704 However, the usable size is not allowed to ** be less than 480. In other words, if the page size is 512, then the ** reserved space size cannot exceed 32.

68899. 64

68900. 313

68901. store the result

68902. Opcode: Integer P1 P2 * * * * Synopsis: r[P2]=P1 *** The 32-bit integer value P1 is written into register P2.

68903. ** If argument zDb is NULL, then call sqlite3CodeVerifySchema() for each ** attached database. Otherwise, invoke it for the database named zDb only.

68904. If there is no %_docszie table, there should be no requests for ** statements to operate on it.

68905. 0x10

68906. Temp area for cell content

68907. Might happen if EMPTY_RESULT_CALLBACKS are on

68908. ** If the TEMP database is open, close it and mark the database schema ** as needing reloading. This must be done when using the SQLITE_TEMP_STORE ** or DEFAULT_TEMP_STORE pragmas.

68909. This routine is a no-op for virtual tables. Leave the output ** variables *pDataCur and *pIdxCur uninitialized so that valgrind ** can detect if they are used by mistake in the caller.

68910. 160

68911. ** Read bytes from *pz and convert them into a positive integer. When ** finished, leave *pz pointing to the first character past the end of ** the integer. The *pLen parameter holds the length of the string ** in *pz and is decremented once for each character in the integer.

68912. Temp space used by merge-sort

68913. Clear flags from pages of the page cache

68914. Update any backup objects copying the contents of this pager.

68915. If it is not a BLOB literal, then it must be an ID, since no ** SQL keywords start with the letter 'x'. Fall through

68916. Suppress duplicates for UNION, EXCEPT, and INTERSECT

68917. High accuracy coordinate

68918. EVIDENCE-OF: R-43263-13491 The total number of bytes in all fragments ** is stored in the fifth field of the b-tree page header. ** EVIDENCE-OF: R-07161-27322 The one-byte integer at offset 7 gives the ** number of fragmented free bytes within the cell content area.

68919. The current SQL statement

68920. IMP: R-24505-23230

68921. ** Read data from the in-memory journal file. This is the implementation ** of the sqlite3_vfs.xRead method.

68922. DESC

68923. Loop counter over the terms of the join

68924. The table key

68925. Optional column name list for the table

68926. Query column to extract text from

68927. Level of current node or entry

68928. Remove connection p from the blocked connections list.

68929. Loop over the tables in the join, from left to right

68930. If we survive all prior tests, that means this term is worth hinting

68931. Compute and store the full pathname in an allocated buffer pointed ** to by zPathname, length nPathname. Or, if this is a temporary file, ** leave both nPathname and zPathname set to 0.

68932. Append content to the JSON parse

68933. The open cursor on the table

68934. Opcode: IntCopy P1 P2 * * * * Synopsis: r[P2]=r[P1] *** Transfer the integer value held in register P1 into register P2. ** * This is an optimized version of SCopy that works only for integer ** values.

68935. **comment:** The following constant maps TK_xx codes into corresponding ** seek opcodes. It depends on a particular ordering of TK_xx
label: code-design

68936. Constraining operation

68937. Number of references to this one

68938. Test that the "totals" (sometimes called "averages") record looks Ok

68939. Increment the free page count on pPage1

68940. The auto-commit flag.

68941. UPDATE + UPDATE

68942. First position in position-list

68943. Temp space sufficient to hold a single cell

68944. If no match, return 0.

68945. Count of the number of search attempts

68946. ***** End of fkey.c *****

68947. A sub-query in the FROM clause of a SELECT

68948. Cursor to scan b-tree

68949. **comment:** ERROR: bad checksum

label: code-design

68950. After pStmt has returned SQLITE_DONE

68951. Notifications from sqlite3_log()

68952. Number of slots in apCsr[]

68953. ** Open a read-transaction on the snapshot identified by pSnapshot.

68954. The table being indexed

68955. ** Release a reference to page pPg. pPg must have been returned by an ** earlier call to pagerAcquireMapPage().

68956. ** Generate code that pushes the value of every element of the given ** expression list into a sequence of registers beginning at target. ** * Return the number of elements evaluated. *** The SQLITE_ECEL_DUP flag prevents the arguments from being ** filled using OP_SCopy. OP_Copy must be used instead. ***

The SQLITE_ECEL_FACTOR argument allows constant arguments to be ** factored out into initialization code. *** The SQLITE_ECEL_REF flag means that expressions in the list with ** ExprList.a[].u.x.iOrderByCol>0 have already been evaluated and stored ** in registers at srcReg, and so the value can be copied from there.

68957. Opcode: RowData P1 P2 P3 * * * * Synopsis: r[P2]=data *** Write into register P2 the complete row content for the row at ** which cursor P1 is currently pointing. ** There is no interpretation of the data. ** It is just copied onto the P2 register exactly as ** it is found in the database file. ** * If cursor P1 is an index, then the content is the key of the row. ** If cursor P2 is a table, then the content extracted is the data. ** * If the P1 cursor must be pointing to a valid row (not a NULL row) ** of a real table, not a pseudo-table. ** * If P3!=0 then this opcode is allowed to make an ephemeral pointer ** into the database page. That means that the content of the output ** register will be invalidated as soon as the cursor moves - including ** moves caused by other cursors that "save" the the current cursors ** position in order that they can write to the same table. If P3==0 ** then a copy of the data is made into memory. P3!=0 is faster, but ** P3==0 is

safer. *** If P3!=0 then the content of the P2 register is unsuitable for use ** in OP_Result and any OP_Result will invalidate the P2 register content. ** The P2 register content is invalidated by opcodes like OP_Function or ** by any use of another cursor pointing to the same table.

68958. ** Free a doclist-index iterator object allocated by fts5DlidxIterInit().

68959. iColumn matches a term of the ORDER BY clause

68960. -1: before desired location +1: after

68961. Ensure the %_content statement is reset.

68962. 41

68963. Not allowed on leftward elements

68964. ** Round down to the nearest multiple of 8

68965. Root node of rtree structure

68966. Zero the output value in case an error occurs.

68967. Description of content to append

68968. Even though this opcode does not use dynamic strings for ** the result, result columns may become dynamic if the user calls ** sqlite3_column_text16(), causing a translation to UTF-16 encoding.

68969. sqlite_stat4.nEq

68970. This checks any of the "extra" bytes allocated due ** to rounding up to an 8 byte boundary to ensure ** they haven't been overwritten.

68971. Btree this cursor belongs to

68972. ** This routine generates code that opens the sqlite_statN tables. ** The sqlite_stat1 table is always relevant. sqlite_stat2 is now ** obsolete. sqlite_stat3 and sqlite_stat4 are only opened when ** appropriate compile-time options are provided. ** ** If the sqlite_statN tables do not previously exist, it is created. ** ** Argument zWhere may be a pointer to a buffer containing a table name, ** or it may be a NULL pointer. If it is not NULL, then all entries in ** the sqlite_statN tables associated with the named table are deleted. ** If zWhere==0, then code is generated to delete all stat table entries.

68973. Fts3MultiSegReader to modify

68974. Set to true if at least one expr. value extracted

68975. Sort in ascending order

68976. Tokenizer argument strings

68977. Decomposition into subterms

68978. The sqlite_sequence table used by AUTOINCREMENT

68979. Bytes of local payload

68980. Obtain the page at pgnoRoot

68981. *** Implementation of offsets() function.

68982. ** This function may only be called when a read-transaction is open on ** the pager. It returns the total number of pages in the database. ** ** However, if the file is between 1 and <page-size> bytes in size, then ** this is considered a 1 page file.

68983. If the attempt to grab the exclusive lock failed, release the ** pending lock that may have been obtained instead.

68984. ** Return the current state of the RBU vacuum or update operation.

68985. Check if this page has already been written into the wal file by ** the current transaction. If so, overwrite the existing frame and ** set Wal.writeLock to WAL_WRITELOCK_RECKSUM - indicating that ** checksums must be recomputed when the transaction is committed.

68986. User pointer passed to xCreate()

68987. Number of SELECT statements seen

68988. Seg-reader is at EOF. Remove the entire input segment.

68989. ***** End of Unique File ID Utility Used By VxWorks *****

68990. 259

68991. ** CAPI3REF: Database Connection Configuration Options ** ** These constants are the available integer configuration options that ** can be passed as the second argument to the [sqlite3_db_config()] interface. ** ** New configuration options may be added in future releases of SQLite. ** Existing configuration options might be discontinued. Applications ** should check the return code from [sqlite3_db_config()] to make sure that ** the call worked. ^The [sqlite3_db_config()] interface will return a ** non-zero [error code] if a discontinued or unsupported configuration option ** is invoked. ** ** <dd> ** <dt>SQLITE_DBCONFIG_LOOKASIDE</dt> ** <dd> ^This option takes three additional arguments that determine the ** [lookaside memory allocator] configuration for the [database connection]. ** ^The first argument (the third parameter to [sqlite3_db_config()]) is a ** pointer to a memory buffer to use for lookaside memory. ** ^The first argument after the SQLITE_DBCONFIG_LOOKASIDE verb ** may be NULL in which case SQLite will allocate the ** lookaside buffer itself using [sqlite3_malloc()]. ^The second argument is the ** size of each lookaside buffer slot. ^The third argument is the number of ** slots. The size of the buffer in the first argument must be greater than ** or equal to the product of the second and third arguments. The buffer ** must be aligned to an 8-byte boundary. ^If the second argument to ** SQLITE_DBCONFIG_LOOKASIDE is not a multiple of 8, it is internally ** rounded down to the next smaller multiple of 8. ^The lookaside memory ** configuration for a database connection can only be changed when that ** connection is not currently using lookaside memory, or in other words ** when the "current value" returned by ** [sqlite3_db_status](D,[SQLITE_CONFIG_LOOKASIDE],..) is zero. ** Any attempt to change the lookaside memory configuration when lookaside ** memory is in use leaves the configuration unchanged and returns ** [SQLITE_BUSY].^</dd> ** <dt>SQLITE_DBCONFIG_ENABLE_FKEY</dt> ** <dd> ^This option is used to enable or disable the enforcement of ** [foreign key constraints]. There should be two additional arguments. ** The first argument is an integer which is 0 to disable FK enforcement, ** positive to enable FK enforcement or negative to leave FK enforcement ** unchanged. The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether FK enforcement is off or on ** following this call. The second parameter may be a NULL pointer, in ** which case the FK enforcement setting is not reported back. </dd> ** ** <dt>SQLITE_DBCONFIG_ENABLE_TRIGGER</dt> ** <dd> ^This option is used to enable or disable [CREATE TRIGGER | triggers]. ** There should be two additional arguments. ** The first argument is an integer which is 0 to disable triggers, ** positive to enable triggers or negative to leave the setting unchanged. ** The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether triggers are disabled or enabled ** following this call. The second parameter may be a NULL pointer, in ** which case the trigger setting is not reported back. </dd> ** ** <dt>SQLITE_DBCONFIG_ENABLE_FTS3_TOKENIZER</dt> ** <dd> ^This option is used to enable or disable the two-argument ** version of the [fts3_tokenizer()] function which is part of the ** [FTS3] full-text search engine extension. ** There should be two additional arguments. ** The first argument is an integer which is 0 to disable fts3_tokenizer() or ** positive to enable fts3_tokenizer() or negative to leave the setting ** unchanged. ** The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether fts3_tokenizer is disabled or enabled ** following this call. The second parameter may be a NULL pointer, in ** which case the new setting is not reported back. </dd> ** ** <dt>SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION</dt> ** <dd> ^This option is used to enable or disable the [sqlite3_load_extension()] ** interface independently of the [load_extension()] SQL function. ** The [sqlite3_enable_load_extension()] API enables or disables both the ** C-API [sqlite3_load_extension()] and the SQL function [load_extension()]. ** There should be two additional arguments. ** When the first argument to this interface is 1, then only the C-API is ** enabled and the SQL function remains disabled. If the first argument to ** this interface is 0, then both the C-API and the SQL function are disabled. ** If the first argument is -1, then no changes are made to state of either the ** C-API or the SQL function. ** The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether [sqlite3_load_extension()] interface ** is disabled or enabled following this call. The second parameter may ** be a NULL pointer, in which case the new setting is not reported back. ** </dd> ** ** <dt>SQLITE_DBCONFIG_MAINDBNAME</dt> ** <dd> ^This option is used to change the name of the "main" database ** schema. ^The sole argument is a pointer to a constant UTF8 string ** which will become the new schema name in place of "main". ^SQLite ** does not make a copy of the new main schema name string, so the application ** must ensure that the argument passed into this DBCONFIG option is unchanged ** until after the database connection closes. ** </dd> ** ** <dt>SQLITE_DBCONFIG_NO_CKPT_ON_CLOSE</dt> ** <dd> Usually, when a database in wal mode is closed or detached from a ** database handle, SQLite checks if this will mean that there are now no ** connections at all to the database. If so, it performs a checkpoint ** operation before closing the connection. This option may be used to ** override this behaviour. The first parameter passed to this operation ** is an integer - non-zero to disable checkpoints-on-close, or zero (the ** default) to enable them. The second parameter is a pointer to an integer ** into which is written 0 or 1 to indicate whether checkpoints-on-close ** have been disabled - 0 if they are not disabled, 1 if they are. ** </dd> ** ** <dt>SQLITE_DBCONFIG_ENABLE_QPSG</dt> ** <dd> ^The SQLITE_DBCONFIG_ENABLE_QPSG option activates or deactivates ** the [query planner stability guarantee] (QPSG). When the QPSG is active, ** a single SQL query statement will always use the same algorithm regardless ** of values of [bound parameters].^< The QPSG disables some query optimizations ** that look at the values of bound parameters, which can make some queries ** slower. But the QPSG has the advantage of more predictable behavior. With ** the QPSG active, SQLite will always use the same query plan in the field as ** was used during testing in the lab. ** </dd> ** ** </dl>

68992. ***** End of btmutex.c *****

68993. ** This function is used to change the actual size of the database ** file in the file-system. This only happens when committing a transaction, ** or rolling back a transaction (including rolling back a hot-journal). *** If the main database file is not open, or the pager is not in either ** DBMOD or OPEN state, this function is a no-op. Otherwise, the size ** of the file is changed to nPage pages (nPage*pPager->pageSize bytes). ** If the file on disk is currently larger than nPage pages, then use the VFS ** xTruncate() method to truncate it. *** Or, it might be the case that the file on disk is smaller than ** nPage pages. Some operating system implementations can get confused if ** you try to truncate a file to some size that is larger than it ** currently is, so detect this case and write a single zero byte to ** the end of the new file instead. *** If successful, return SQLITE_OK. If an IO error occurs while modifying ** the database file, return the error code to the caller.

68994. Mask of new.* columns referenced

68995. NodeWriter for root node

68996. The DISTINCT marking is pointless. Ignore it.

68997. CREATE VIEW name(arglist) AS ... ** The names of the columns in the table are taken from ** arglist which is stored in pTable->pCheck. The pCheck field ** normally holds CHECK constraints on an ordinary table, but for ** a VIEW it holds the list of column names.

68998. Cache of overflow page locations

68999. OUT: Size of buffer at *ppChangese

69000. ** CAPI3REF: Create A New Changegroup Object ** ** An sqlite3_changegroup object is used to combine two or more changesets ** (or patchsets) into a single changeset (or patchset). A single changegroup ** object may combine changesets or patchsets, but not both. The output is ** always in the same format as the input. ** ** If successful, this function returns SQLITE_OK and populates (*pp) with ** a pointer to a new sqlite3_changegroup object before returning. The caller ** should eventually free the returned object using a call to ** sqlite3changegroup_delete(). If an error occurs, an SQLite error code ** (i.e. SQLITE_NOMEM) is returned and *pp is set to NULL. ** ** The usual usage pattern for an sqlite3_changegroup object is as follows: ** ** ** It is created using a call to sqlite3changegroup_new(). ** ** Zero or more changesets (or patchsets) are added to the object ** by calling sqlite3changegroup_add(). ** ** The result of combining all input changesets together is obtained ** by the application via a call to sqlite3changegroup_output(). ** ** The object is deleted using a call to sqlite3changegroup_delete(). ** ** ** Any number of calls to add() and output() may be made between the calls to ** new() and delete(), and in any order. ** ** As well as the regular sqlite3changegroup_add() and ** sqlite3changegroup_output() functions, also available are the streaming ** versions sqlite3changegroup_add_strm() and sqlite3changegroup_output_strm().

69001. A WHERE clause term

69002. ** Candidate values for Wal.exclusiveMode.

69003. OUT: Array of booleans - true for PK col

69004. Loop through all of the arguments passed by the user to the FTS3/4 ** module (i.e. all the column names and special arguments). This loop ** does the following: ** ** + Figures out the number of columns the FTSX table will have, and ** the number of bytes of space that must be allocated to store copies ** of the column names. ** ** + If there is a tokenizer specification included in the arguments, ** initializes the tokenizer pTokenizer.

69005. EVIDENCE-OF: R-43249-19882 The third through sixth parameters to the ** callback are either NULL pointers or zero-terminated strings that ** contain additional details about the action to be authorized. ** ** The following testcase() macros show that any of the 3rd through 6th ** parameters can be either NULL or a string.

69006. Last synced page in dirty page list

69007. Test the query function

69008. ** These file mapping APIs are common to both Win32 and WinRT.

69009. Total number of tokens in row

69010. The value return

69011. The VFS for which this is the xOpen method

69012. Original KeyInfo on the sorter table

69013. EACH

69014. **comment:** ** Some functions like COALESCE() and IFNULL() and UNLIKELY() are implemented ** as VDBE code so that unused argument values do not have to be computed. ** However, we still need some kind of function implementation for this ** routines in the function table. The noopFunc macro provides this. ** noopFunc will never be called so it doesn't matter what the implementation ** is. We might as well use the "version()" function as a substitute.
label: code-design

69015. Number of u32 values in hash table.

69016. **comment:** ** If required, grow the hash table used to store changes on table pTab ** (part of the session pSession). If a fatal OOM error occurs, set the ** session object to failed and return SQLITE_ERROR. Otherwise, return ** SQLITE_OK. ** ** It is possible that a non-fatal OOM error occurs in this function. In ** that case the hash-table does not grow, but SQLITE_OK is returned anyway. ** Growing the hash table in this case is a performance optimization only, ** it is not required for correct operation.
label: code-design

69017. ** Register the built-in LIKE and GLOB functions. The caseSensitive ** parameter determines whether or not the LIKE operator is case ** sensitive. GLOB is always case sensitive.

69018. s: p0<<14 | p2 (masked)

69019. Make sure the mutex subsystem is initialized. If unable to ** initialize the mutex subsystem, return early with the error. ** If the system is so sick that we are unable to allocate a mutex, ** there is not much SQLite is going to be able to do. ** ** The mutex subsystem must take care of serializing its own ** initialization.

69020. ** eType: ** Expression node type. Always one of: ** ** FTS5_AND (nChild, apChild valid) ** FTS5_OR (nChild, apChild valid) ** FTS5_NOT (nChild, apChild valid) ** FTS5_STRING (pNear valid) ** FTS5_TERM (pNear valid)

69021. ** Initialize the contents of the unixFile structure pointed to by pid.

69022. Make a copy of the second argument (a blob) in aBlob[]. The aBlob[] ** copy is followed by FTS5_DATA_ZERO_PADDING 0x00 bytes, which prevents ** buffer overreads even if the record is corrupt.

69023. **comment:** ** If we are to be thread-safe, include the pthreads header and define ** the SQLITE_UNIX_THREADS macro.
label: requirement

69024. The span to be added

69025. EVIDENCE-OF: R-11498-58022 In a well-formed b-tree page, the total ** number of bytes in fragments may not exceed 60.

69026. Column to search

69027. Number of frames in pFrame list

69028. Statement under construction

69029. synopsis: accum=r[P3] step(r[P2@P5])

69030. Column to read value from

69031. Space for new cursor

69032. sqlite3_reset() return code

69033. **comment:** ** The first argument is a file-handle open on a temporary file. The file ** is guaranteed to be nByte bytes or smaller in size. This function ** attempts to extend the file to nByte bytes in size and to ensure that ** the VFS has memory mapped it. ** ** Whether or not the file does end up memory mapped of course depends on ** the specific VFS implementation.
label: code-design

69034. Opcode: Function0 P1 P2 P3 P4 P5 ** Synopsis: r[P3]=func(r[P2@P5]) ** ** Invoke a user function (P4 is a pointer to a FuncDef object that ** defines the function) with P5 arguments taken from register P2 and ** successors. The result of the function is stored in register P3. ** Register P3 must not be one of the function inputs. ** ** P1 is a 32-bit bitmask indicating whether or not each argument to the ** function was determined to be constant at compile time. If the first ** argument was constant then bit 0 of P1 is set. This is used to determine ** whether meta data associated with a user function argument using the ** sqlite3_set_auxdata() API may be safely retained until the next ** invocation of this opcode. ** ** See also: Function, AggStep, AggFinal

69035. Update the anEq[] fields of any samples already collected.

69036. Content of the NEW.* table in triggers

69037. Top of the input loop

69038. Segid to write to

69039. If non-zero free this temporary register

69040. ** This function is used by changeset_concat() to merge two UPDATE changes ** on the same row.

69041. INDEX

69042. in1, in2
69043. Allocate the new entry in the db->aDb[] array and initialize the schema ** hash tables.
69044. not a system error
69045. **comment:** Even if an IO or diskfull error occurs while journaling the ** page in the block above, set the need-sync flag for the page. ** Otherwise, when the transaction is rolled back, the logic in ** playback_one_page() will think that the page needs to be restored ** in the database file. And if an IO error occurs while doing so, ** then corruption may follow.
label: code-design
69046. Buffer in which to accumulate changeset
69047. True if the terms are equal
69048. ** The table object reference passed as the second argument to this function ** must represent a virtual table. This function invokes the xBestIndex() ** method of the virtual table with the sqlite3_index_info object that ** comes in as the 3rd argument to this function. ** ** If an error occurs, pParse is populated with an error message and a ** non-zero value is returned. Otherwise, 0 is returned and the output ** part of the sqlite3_index_info structure is left populated. ** ** Whether or not an error is returned, it is the responsibility of the ** caller to eventually free p->idxStr if p->needToFreeIdxStr indicates ** that this is required.
69049. Pointer to rru object (rru target only)
69050. ** Helper function used by the implementation of the overloaded snippet(), ** offsets() and optimize() SQL functions. ** ** If the value passed as the third argument is a blob of size ** sizeof(Fts3Cursor*), then the blob contents are copied to the ** output variable *ppCsr and SQLITE_OK is returned. Otherwise, an error ** message is written to context pContext and SQLITE_ERROR returned. The ** string passed via zFunc is used as part of the error message.
69051. Access authorization function
69052. Find a dirty page to write-out and recycle. First try to find a ** page that does not require a journal-sync (one with PGHDR_NEED_SYNC ** cleared), but if that is not possible settle for any other ** unreferenced dirty page. ** ** If the LRU page in the dirty list that has a clear PGHDR_NEED_SYNC ** flag is currently referenced, then the following may leave pSynced ** set incorrectly (pointing to other than the LRU page with NEED_SYNC ** cleared). This is Ok, as pSynced is just an optimization.
69053. BY => ID
69054. ** This is an fts3ExprIterate() callback used while loading the doclists ** for each phrase into Fts3Expr.aDoclist[]/nDoclist. See also ** fts3ExprLoadDoclists().
69055. Collating sequence name
69056. Dummy variables
69057. Step 1C.
69058. Byte available in aRow
69059. IMPLEMENTATION-OF: R-59782-36818 The SQLITE_CHECKPOINT_FULL, RESTART and ** TRUNCATE modes also obtain the exclusive "writer" lock on the database ** file. ** ** EVIDENCE-OF: R-60642-04082 If the writer lock cannot be obtained ** immediately, and a busy-handler is configured, it is invoked and the ** writer lock retried until either the busy-handler returns 0 or the ** lock is successfully obtained.
69060. ** Delete the file located at zPath.
69061. The underlying file iNode
69062. Current offset in pgidx
69063. pIdx is usable
69064. In the IEEE 754 standard, zero is signed.
69065. The following testcase is true for indices with redundant columns. ** Ex: CREATE INDEX i1 ON t1(a,b,a); SELECT * FROM t1 WHERE a=0 AND b=0;
69066. SQLITE_OMIT_TRACE
69067. List of statements inside the trigger body
69068. Rowid from %_data table
69069. Destructor for pUserData
69070. ** This function returns a pointer to a blob of memory associated with ** a single shared-btree. The memory is used by client code for its own ** purposes (for example, to store a high-level schema associated with ** the shared-btree). The btree layer manages reference counting issues. ** ** The first time this is called on a shared-btree, nBytes bytes of memory ** are allocated, zeroed, and returned to the caller. For each subsequent ** call the nBytes parameter is ignored and a pointer to the same blob ** of memory returned. ** ** If the nBytes parameter is 0 and the blob of memory has not yet been ** allocated, a null pointer is returned. If the blob has already been ** allocated, it is returned as normal. ** ** Just before the shared-btree is closed, the function passed as the ** xFree argument when the memory allocation was made is invoked on the ** blob of allocated memory. The xFree function should not call sqlite3_free() ** on the memory, the btree layer does that.
69071. Make sure the results of the current row are \000 terminated ** and have an assigned type. The results are de-ephemeralized as ** a side effect.
69072. If we have any lock, then the lock file already exists. All we have ** to do is adjust our internal record of the lock level.
69073. colset
69074. ** This routine is called when a VDBE tries to halt. If the VDBE ** has made changes and is in autocommit mode, then commit those ** changes. If a rollback is needed, then do the rollback. ** ** This routine is the only way to move the state of a VM from ** SQLITE_MAGIC_RUN to SQLITE_MAGIC_HALT. It is harmless to ** call this on a VM that is in the SQLITE_MAGIC_HALT state. ** ** Return an error code. If the commit could not complete because of ** lock contention, return SQLITE_BUSY. If SQLITE_BUSY is returned, it ** means the close did not happen and needs to be repeated.
69075. True for ORDER BY ASC
69076. Rare case of a really large header
69077. 240
69078. Try to allocate nByte bytes of 8-byte aligned bulk memory for pBuf ** from the ReusableSpace object. Return a pointer to the allocated ** memory on success. If insufficient memory is available in the ** ReusableSpace object, increase the ReusableSpace.nNeeded ** value by the amount needed and return NULL. ** ** If pBuf is not initially NULL, that means that the memory has already ** been allocated by a prior call to this routine, so just return a copy ** of pBuf and leave ReusableSpace unchanged. ** ** This allocator is employed to repurpose unused slots at the end of the ** opcode array of prepared state for other memory needs of the prepared ** statement.
69079. If this index contains every column of its table, then mark ** it as a covering index
69080. ** Used to pass information from the analyzer reader through to the ** callback routine.
69081. Register to copy to
69082. ** Return true if the correct mutexes are held for accessing the ** db->aDb[iDb].pSchema structure. The mutexes required for schema ** access are: ** ** (1) The mutex on db ** (2) if iDb!=1, then the mutex on db->aDb[iDb].pBt. ** ** If pSchema is not NULL, then iDb is computed from pSchema and ** db using sqlite3SchemaToIndex().
69083. Remove result from a UNION index
69084. ** If we are on an architecture with mixed-endian floating ** points (ex: ARM7) then swap the lower 4 bytes with the ** upper 4 bytes. Return the result. ** ** For most architectures, this is a no-op. ** ** (later): It is reported to me that the mixed-endian problem ** on ARM7 is an issue with GCC, not with the ARM7 chip. It seems ** that early versions of GCC stored the two words of a 64-bit ** float in the wrong order. And that error has been propagated ** ever since. The blame is not necessarily with GCC, though. ** GCC might have just copying the problem from a prior compiler. ** I am also told that newer versions of GCC that follow a different ** ABI get the byte order right. ** ** Developers using SQLite on an ARM7 should compile and run their ** application using -DSQLITE_DEBUG=1 at least once. With DEBUG ** enabled, some asserts below will ensure that the byte order of ** floating point values is correct. ** ** (2007-08-30) Frank van Vugt has studied this problem closely ** and has sent his findings to the SQLite developers. Frank ** writes that some Linux kernels offer floating point hardware ** emulation that uses only 32-bit mantissas instead of a full ** 48-bits as required by the IEEE standard. (This is the ** CONFIG_FPE_FASTFP option.) On such systems, floating point ** byte swapping becomes very complicated. To avoid problems, ** the necessary byte swapping is carried out using a 64-bit integer ** rather than a 64-bit float. Frank assures us that the code here ** works for him. We, the developers, have no way to independently ** verify this, but Frank seems to know what he is talking about ** so we trust him.
69085. If an OOM occurs, set to SQLITE_NOMEM
69086. Reserved: 0x00F00000
69087. Never set both isSaveLeft and isExact for the same invocation.
69088. The next sections is a series of control #defines. ** various aspects of the generated parser. ** fts5YYCODETYPE is the data type used to store the integer codes ** that represent terminal and non-terminal symbols. ** "unsigned char" is used if there are fewer than ** 256 symbols. Larger types otherwise. ** fts5YYNOCODE is a number of type fts5YYCODETYPE that is not used for ** any terminal or nonterminal symbol. ** fts5YYFALLBACK If defined, this indicates that one or more tokens ** (also known as: "terminal symbols") have fall-back ** values which should be used if the original symbol ** would not parse.

This permits keywords to sometimes ** be used as identifiers, for example. ** fts5YYACTIONTYPE is the data type used for "action codes" - numbers ** that indicate what to do in response to the next ** token. ** sqlite3Fts5ParserFTS5TOKENTYPE is the data type used for minor type for terminal ** symbols. Background: A "minor type" is a semantic ** value associated with a terminal or non-terminal ** symbols. For example, for an "ID" terminal symbol, ** the minor type might be the name of the identifier. ** Each non-terminal can have a different minor type. ** Terminal symbols all have the same minor type, though. ** This macros defines the minor type for terminal ** symbols. ** fts5YYMINORTYPE is the data type used for all minor types. ** This is typically a union of many types, one of ** which is sqlite3Fts5ParserFTS5TOKENTYPE. The entry in the union ** for terminal symbols is called "fts5yy0". ** fts5YYSTACKDEPTH is the maximum depth of the parser's stack. If ** zero the stack is dynamically sized using realloc() ** sqlite3Fts5ParserARG_SDECL A static variable declaration for the %extra_argument ** sqlite3Fts5ParserARG_PDECL A parameter declaration for the %extra_argument ** sqlite3Fts5ParserARG_STORE Code to store %extra_argument into fts5yypParser ** sqlite3Fts5ParserARG_FETCH Code to extract %extra_argument from fts5yypParser ** fts5YYERRORSYMBOL is the code number of the error symbol. If not ** defined, then do no error processing. ** fts5YYNSTATE the combined number of states. ** fts5YYNRULE the number of rules in the grammar ** fts5YY_MAX_SHIFT Maximum value for shift actions ** fts5YY_MIN_SHIFTREDUCE Minimum value for shift-reduce actions ** fts5YY_MAX_SHIFTREDUCE Maximum value for shift-reduce actions ** fts5YY_MIN_REDUCE Maximum value for reduce actions ** fts5YY_ERROR_ACTION The fts5yy_action[] code for syntax error ** fts5YY_ACCEPT_ACTION The fts5yy_action[] code for accept ** fts5YY_NO_ACTION The fts5yy_action[] code for no-op

69089. Maximum local payload in a LEAFDATA table

69090. Free-list trunk page

69091. ** Attach a table to a session. All subsequent changes made to the table ** while the session object is enabled will be recorded. ** ** Only tables that have a PRIMARY KEY defined may be attached. It does ** not matter if the PRIMARY KEY is an "INTEGER PRIMARY KEY" (rowid alias) ** or not.

69092. Symbolic ID of this loop for debugging use

69093. **comment:** ** The "winIoerrCanRetry1" macro is used to determine if a particular I/O ** error code obtained via GetLastError() is eligible to be retried. It ** must accept the error code DWORD as its only argument and should return ** non-zero if the error code is transient in nature and the operation ** responsible for generating the original error might succeed upon being ** retried. The argument to this macro should be a variable. ** ** Additionally, a macro named "winIoerrCanRetry2" may be defined. If it ** is defined, it will be consulted only when the macro "winIoerrCanRetry1" ** returns zero. The "winIoerrCanRetry2" macro is completely optional and ** may be used to include additional error codes in the set that should ** result in the failing I/O operation being retried by the caller. If ** defined, the "winIoerrCanRetry2" macro must exhibit external semantics ** identical to those of the "winIoerrCanRetry1" macro.

label: code-design

69094. 0x1e

69095. Update the PCache1.pSynced variable if necessary.

69096. Check that the affinity that will be used to perform each ** comparison is the same as the affinity of each column in table ** on the RHS of the IN operator. If it not, it is not possible to ** use any index of the RHS table.

69097. (2c)

69098. Checkpoint sequence counter in the wal-header

69099. ***** End reduce actions *****

69100. Begin reading at this offset

69101. If the sub-query is a compound SELECT statement, then (by restrictions ** 17 and 18 above) it must be a UNION ALL and the parent query must ** be of the form: ** ** SELECT <expr-list> FROM (<sub-query>) <where-clause> *** followed by any ORDER BY, LIMIT and/or OFFSET clauses. This block ** creates N-1 copies of the parent query without any ORDER BY, LIMIT or ** OFFSET clauses and joins them to the left-hand-side of the original ** using UNION ALL operators. In this case N is the number of simple ** select statements in the compound sub-query. ** ** Example: ** ** SELECT a+1 FROM (** SELECT x FROM tab ** UNION ALL ** SELECT y FROM tab ** UNION ALL ** SELECT abs(z*2) FROM tab2 **) WHERE a!=5 ORDER BY 1 ** ** Transformed into: ** ** SELECT x+1 FROM tab WHERE x+1!=5 ** UNION ALL ** SELECT y+1 FROM tab WHERE y+1!=5 ** UNION ALL ** SELECT abs(z*2)+1 FROM tab2 WHERE abs(z*2)+1!=5 ** ORDER BY 1 ** ** We call this the "compound-subquery flattening".

69102. **comment:** ** If SQLITE_CHECK_PAGES is defined then we do some sanity checking ** on the cache using a hash function. This is used for testing ** and debugging only.

label: code-design

69103. Look up every table named in the FROM clause of the select. If ** an entry of the FROM clause is a subquery instead of a table or view, ** then create a transient table structure to describe the subquery.

69104. True if must seek pStmt to %_content row

69105. ** xSetOutputs callback used by detail=none tables.

69106. ** Free the phrase object passed as the only argument.

69107. xDlClose

69108. Omit constant folding optimizations

69109. P4 is a pointer to a CollSeq structure

69110. ** Drop a page from the cache. There must be exactly one reference to the ** page. This function deletes that reference, so after it returns the ** page pointed to by p is invalid.

69111. Transform expressions in this WHERE clause

69112. ** This function returns a linked list of FKey objects (connected by ** FKey.pNextTo) holding all children of table pTab. For example, ** given the following schema: ** ** CREATE TABLE t1(a PRIMARY KEY); ** CREATE TABLE t2(b REFERENCES t1(a); ** ** Calling this function with table "t1" as an argument returns a pointer ** to the FKey structure representing the foreign key constraint on table ** "t2". Calling this function with "t2" as the argument would return a ** NULL pointer (as there are no FK constraints for which t2 is the parent ** table).

69113. String to tokenize

69114. Test for EOF.

69115. OUT: WHERE clause

69116. xNext

69117. Write the node-id here

69118. The handle to our heap.

69119. Number of entries without a matching leave

69120. ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This module contains C code that generates VDBE code used to process ** the WHERE clause of SQL statements. This module is responsible for ** generating the code that loops through a table looking for applicable ** rows. Indices are selected and used to speed the search when doing ** so is applicable. Because this module is responsible for selecting ** indices, you might also think of this module as the "query optimizer".

69121. ** Close all cursors. ** ** Also release any dynamic memory held by the VM in the Vdbe.aMem memory ** cell array. This is necessary as the memory cell array may contain ** pointers to VdbeFrame objects, which may in turn contain pointers to ** open cursors.

69122. ** Initialize all database files - the main database file, the file ** used to store temporary tables, and any additional database files ** created using ATTACH statements. Return a success code. If an ** error occurs, write an error message into *pzErrMsg. ** ** After a database is initialized, the DB_SchemaLoaded bit is set ** bit is set in the flags field of the Db structure. If the database ** file was of zero-length, then the DB_EEmpty flag is also set.

69123. Rowid for the inserted record

69124. ** start of TTTTT ** ** Move the date backwards to the beginning of the current day, ** or month or year.

69125. Reserved for future use

69126. ** Construct a trigger step that implements a DELETE statement and return ** a pointer to that trigger step. The parser calls this routine when it ** sees a DELETE statement inside the body of a CREATE TRIGGER.

69127. If the SQLITE_CountRows flag is set in sqlite3.flags mask, then ** DML statements invoke this opcode to return the number of rows ** modified to the user. This is the only way that a VM that ** opens a statement transaction may invoke this opcode. ** ** In case this is such a statement, close any statement transaction ** opened by this VM before returning control to the user. This is to ** ensure that statement-transactions are always nested, not overlapping. ** If the open statement-transaction is not closed here, then the user ** may step another VM that opens its own statement transaction. This ** may lead to overlapping statement transactions. ** ** The statement transaction is never a top-level transaction. Hence ** the RELEASE call below can never fail.

69128. For json_each() path and root are the same so fall through ** into the root case

69129. Check that the results returned for ASC and DESC queries are ** the same. If not, call this corruption.
69130. Column-list terminator
69131. Allocated size of aIdx[]
69132. trigger_time ::= BEFORE|AFTER
69133. Address of test for empty pSrc
69134. The call to execSql() to attach the temp database has left the file ** locked (as there was more than one active statement when the transaction ** to read the schema was concluded. Unlock it here so that this doesn't ** cause problems for the call to BtreeSetPageSize() below.
69135. expr ::= expr IS NOT expr
69136. Number of constraint terms
69137. List of all chunk allocations
69138. natural
69139. Value of 'payload' column
69140. Size of a cell
69141. Used by: module_list pragma_list
69142. These macros are provided to "stringify" the value of the define ** for those options in which the value is meaningful.
69143. Update the aggregate accumulators based on the content of ** the current row
69144. Advance the current PmaReader
69145. Space allocated for leaf blocks
69146. expr ::= expr between_op expr AND expr
69147. ** This is the xSetSystemCall() method of sqlite3_vfs for all of the ** "unix" VFSes. Return SQLITE_OK upon successfully updating the ** system call pointer, or SQLITE_NOTFOUND if there is no configurable ** system call named zName.
69148. ** Search for an auxiliary function named zName that can be used with table ** pTab. If one is found, return a pointer to the corresponding Fts5Auxiliary ** structure. Otherwise, if no such function exists, return NULL.
69149. ** Helper function for printing out trace information from debugging ** binaries. This returns the string representation of the supplied ** integer lock-type.
69150. Btree database to read cookie from
69151. Serial type of the rowid
69152. 0x000004 // available for use
69153. ** Rtree virtual table module xRowid method.
69154. Byte-length of string zStop
69155. Version 3.9.0 and later
69156. The underlying winShmNode object
69157. Pointer to previous term buffer
69158. ** Deinitialize the memory allocation subsystem.
69159. If the value is being opened for writing, check that the ** column is not indexed, and that it is not part of a foreign key.
69160. Mem address holding flag indicating that at least ** one row of the input to the aggregator has been ** processed
69161. ** Initialize the iterator structure passed as the second argument. ** ** If no error occurs, SQLITE_OK is returned and the iterator is left ** pointing to the first entry. Otherwise, an error code and message is ** left in the RBU handle passed as the first argument. A copy of the ** error code is returned.
69162. Parsing and code generating context
69163. ** Each session object maintains a set of the following structures, one ** for each table the session object is monitoring. The structures are ** stored in a linked list starting at sqlite3_session.pTable. ** ** The keys of the SessionTable.aChange[] hash table are all rows that have ** been modified in any way since the session object was attached to the ** table. ** ** The data associated with each hash-table entry is a structure containing ** a subset of the initial values that the modified row contained at the ** start of the session. Or no initial values if the row was inserted.
69164. >=2
69165. ** Name of the connection operator, used for error messages.
69166. Low-level page-cache interface
69167. All ram filesystem writes are atomic
69168. The GROUP BY clause
69169. ***** Begin file hwtime.h *****
69170. Array of objects. Might be reallocated
69171. Opcode: AggStep * P2 P3 P4 P5 ** Synopsis: accum=r[P3] step(r[P2@P5]) ** ** Execute the step function for an aggregate. The ** function has P5 arguments. P4 is a pointer to an sqlite3_context ** object that is used to run the function. Register P3 is ** as the accumulator. ** ** The P5 arguments are taken from register P2 and its ** successors. ** ** This opcode is initially coded as OP_AggStep0. On first evaluation, ** the FuncDef stored in P4 is converted into an sqlite3_context and ** the opcode is changed. In this way, the initialization of the ** sqlite3_context only happens once, instead of on each call to the ** step function.
69172. 87
69173. Number of purgeable pages allocated
69174. Never spill cache. Set via pragma
69175. SELECT rowid, * FROM ... ORDER BY 1 ASC
69176. Number of bytes of data to be bound
69177. Last position from pp2
69178. Make *pzErr point to any syntax error in zPath
69179. Used to iterate through FKs
69180. fts3 query parse context
69181. Initialize and fill this JsonParse object
69182. Original value of pNew->nLTerm
69183. Initialize the JsonString object
69184. Cursor pointing to a table row
69185. Assert that the upper layer has set one of the "file-type" flags.
69186. Mutex to serialize access
69187. Always update the timestamp on the old file
69188. B
69189. Cursor for this UNIQUE index
69190. ** This function is only ever called on iterators created by calls to ** Fts5IndexQuery() with the FTS5INDEX_QUERY_DESC flag set. ** ** The iterator is in an unusual state when this function is called: the ** Fts5SegIter.iLeafOffset variable is set to the offset of the start of ** the position-list size field for the first relevant rowid on the page. ** Fts5SegIter.rowid is set, but nPos and bDel are not. ** ** This function advances the iterator so that it points to the last ** relevant rowid on the page and, if necessary, initializes the ** aRowidOffset[] and iRowidOffset variables. At this point the iterator ** is in its regular state - Fts5SegIter.iLeafOffset points to the first ** byte of the position list content associated with said rowid.
69191. old.* record for first change
69192. Convert the lower bound to upper-case and the upper bound to ** lower-case (upper-case is less than lower-case in ASCII) so that ** the range constraints also work for BLOBS
69193. IMPLEMENTATION-OF: R-07677-41881 If the largest ROWID is equal to the ** largest possible integer (9223372036854775807) then the database ** engine starts picking positive candidate ROWIDs at random until ** it finds one that is not previously used.
69194. PRAGMA index_list = (pIter->zTbl)
69195. Allocate space for both the pJournal and pMaster file descriptors. ** If successful, open the master journal file for reading.
69196. ** Argument pList points to a position list nList bytes in size. This ** function checks to see if the position list contains any entries for ** a token in position 0 (of any column). If so, it writes argument iDelta ** to the output buffer pOut, followed by a position list consisting only ** of the entries from pList at position 0, and terminated by an 0x00 byte. ** The value returned is the number of bytes written to pOut (if any).

69197. If foreign-key support is enabled, iterate through all FKs that ** refer to table pTab. If there is an action associated with the FK ** for this operation (either update or delete), invoke the associated ** trigger sub-program.

69198. **comment:** The right-most of SELECTs to be coded
label: code-design

69199. **comment:** If this playback is happening automatically as a result of an IO or ** malloc error that occurred after the change-counter was updated but ** before the transaction was committed, then the change-counter ** modification may just have been reverted. If this happens in exclusive ** mode, then subsequent transactions performed by the connection will not ** update the change-counter at all. This may lead to cache inconsistency ** problems for other processes at some point in the future. So, just ** in case this has happened, clear the changeCountDone flag now.
label: code-design

69200. Cursor to iterate through aLeft

69201. If we are rereading the sqlite_master table create the in-memory ** record of the table. The xConnect() method is not called until ** the first time the virtual table is used in an SQL statement. This ** allows a schema that contains virtual tables to be loaded before ** the required virtual table implementations are registered.

69202. Digits prior to the decimal point

69203. For leaf pages, the coverage check will occur in the same loop ** as the other cell checks, so initialize the heap.

69204. The real indices of the table are only considered if the ** NOT INDEXED qualifier is omitted from the FROM clause

69205. Load the value for the inequality constraint at the end of the ** range (if any).

69206. ** Provide the ability to override linkage features of the interface.

69207. Create the whole "CREATE TABLE" statement to pass to SQLite

69208. The CREATE token that begins the statement

69209. Unrecognized argument

69210. ** Return the value of the iSize parameter specified when Bitvec *p ** was created.

69211. True if rev is known

69212. ** 2014 August 11 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

69213. Set the first entry (number of rows in the index) to the estimated ** number of rows in the table, or half the number of rows in the table ** for a partial index. But do not let the estimate drop below 10.

69214. 1: file 2: memory 0: default

69215. ** This function may only be called with a changeset iterator that has been ** passed to an SQLITE_CHANGESET_DATA or SQLITE_CHANGESET_CONFLICT ** conflict-handler function. Otherwise, SQLITE_MISUSE is returned. ** ** If successful, *ppValue is set to point to an sqlite3_value structure ** containing the iVal'th value of the conflicting record. ** ** If value iVal is out-of-range or some other error occurs, an SQLite error ** code is returned. Otherwise, SQLITE_OK.

69216. Generate code into this VM

69217. 292

69218. eCreate defines what to do if the page does not exist. ** 0 Do not allocate a new page. (createFlag==0) ** 1 Allocate a new page if doing so is inexpensive. ** (createFlag==1 AND bPurgeable AND pDirty) ** 2 Allocate a new page even it doing so is difficult. ** (createFlag==1 AND !(bPurgeable AND pDirty)

69219. ** CAPI3REF: Determine if a database is read-only ** METHOD: sqlite3 *** ** ^The sqlite3_db_READONLY(D,N) interface returns 1 if the database N ** of connection D is read-only, 0 if it is read/write, or -1 if N is not ** the name of a database on connection D.

69220. The opcodes to be added

69221. Table being altered

69222. The nTo best paths at the current level

69223. Size of journal file on disk

69224. Value for real types

69225. ** If possible, return a pointer to a mapping of file fd starting at offset ** iOff. The mapping must be valid for at least nAmt bytes. ** ** If such a pointer can be obtained, store it in *pp and return SQLITE_OK. ** Or, if one cannot but no error occurs, set *pp to 0 and return SQLITE_OK. ** Finally, if an error does occur, return an SQLite error code. The final ** value of *pp is undefined in this case. ** ** If this function does return a pointer, the caller must eventually ** release the reference by calling unixUnfetch().

69226. Existing change

69227. ** The datatype used to store estimates of the number of rows in a ** table or index. This is an unsigned integer type. For 99.9% of ** the world, a 32-bit integer is sufficient. But a 64-bit integer ** can be used at compile-time if desired.

69228. ** Set result column names for a pragma.

69229. True if next term will be first in leaf

69230. on_opt ::= ON expr

69231. Number of bytes in aPoslist

69232. **comment:** TUNING: Estimated cost of a full external sort, where N is ** the number of rows to sort is: ** ** cost = (3.0 * N * log(N)). ** ** Or, if the order-by clause has X terms but only the last Y ** terms are out of order, then block-sorting will reduce the ** sorting cost to: ** ** cost = (3.0 * N * log(N)) * (Y/X) ** ** The (Y/X) term is implemented using stack variable rScale ** below.
label: code-design

69233. ** Return SQLITE_ERROR if the maximum depth of the expression tree passed ** as the only argument is more than nMaxDepth.

69234. Serial type

69235. Make sure mem5.aiFreelist[iLogsize] contains at least one free ** block. If not, then split a block of the next larger power of ** two in order to create a new free block of size iLogsize.

69236. ** SELECT ... FROM (SELECT ... LIMIT a OFFSET b) LIMIT x OFFSET y; ** ** One is tempted to try to add a and b to combine the limits. But this ** does not work if either limit is negative.

69237. END

69238. synopsis: if r[P1]>0 then r[P1]=-P3, goto P2

69239. True for DESCENDING sort order

69240. Temporary registers

69241. Size of pData. 0 if none.

69242. 83

69243. ** Allowed values for the eMode parameter to vdbeMergeEngineInit() ** and vdbePmaReaderIncrMergeInit(). ** ** Only INCRINIT_NORMAL is valid in single-threaded builds (when ** SQLITE_MAX_WORKER_THREADS==0). The other values are only used ** when there exists one or more separate worker threads.

69244. ** Register a 2nd-generation geometry callback named zScore that can be ** used as part of an R-Tree geometry query as follows: ** ** SELECT ... FROM <rtree> WHERE <rtree col> MATCH \$zQueryFunc(... params ...)

69245. ** 2006 Oct 10 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This is an SQLite module implementing full-text search.

69246. ** Create and populate a new TriggerPrg object with a sub-program ** implementing trigger pTrigger with ON CONFLICT policy onconf.

69247. Bind values to the UPDATE statement.

69248. ** This routine does the work of opening a database on behalf of ** sqlite3_open() and sqlite3_open16(). The database filename "zFilename" ** is UTF-8 encoded.

69249. **comment:** ** Obtain an indication as to the current stage of an RBU update or vacuum. ** This function always returns one of the SQLITE_RBU_STATE_XXX constants ** defined in this file. Return values should be interpreted as follows: ** ** SQLITE_RBU_STATE_OAL: ** RBU is currently building a *-oal file. The next call to sqlite3rbu_step() ** may either add further data to the *-oal file, or compute data that will ** be added by a subsequent call. ** ** SQLITE_RBU_STATE_MOVE: ** RBU has finished building the *-oal file. The next call to sqlite3rbu_step() ** will move the *-oal file to the equivalent *-wal path. If the current ** operation is an RBU update, then the updated version of the database ** file will become visible to ordinary SQLite clients following the next ** call to sqlite3rbu_step(). ** ** SQLITE_RBU_STATE_CHECKPOINT: ** RBU is currently performing an incremental checkpoint. The next call to **

sqlite3rbu_step() will copy a page of data from the *-wal file into ** the target database file. *** ** SQLITE_RBU_STATE_DONE: ** The RBU operation has finished. Any subsequent calls to sqlite3rbu_step() ** will immediately return SQLITE_DONE. *** ** SQLITE_RBU_STATE_ERROR: ** An error has occurred. Any subsequent calls to sqlite3rbu_step() will ** immediately return the SQLite error code associated with the error.

label: code-design

69250. ** Make sure that the compiler intrinsics we desire are enabled when ** compiling with an appropriate version of MSVC unless prevented by ** the SQLITE_DISABLE_INTRINSIC define.
69251. True if REPLACE conflict resolution might happen
69252. Release the reference to the root node.
69253. ** The first two arguments are a pointer to and the size of a segment b-tree ** node. The node may be a leaf or an internal node. *** ** This function creates a new node image in blob object *pNew by copying ** all terms that are greater than or equal to zTerm/nTerm (for leaf nodes) ** or greater than zTerm/nTerm (for internal nodes) from aNode/nNode.
69254. Try to find the next page in the overflow list using the ** autovacuum pointer-map pages. Guess that the next page in ** the overflow list is page number (ovfl+1). If that guess turns ** out to be wrong, fall back to loading the data of page ** number ovfl to determine the next page number.
69255. Clear the TEMP schema separately and last
69256. ** Bits of the sqlite3.dbOptFlags field that are used by the ** sqlite3_test_control(SQLITE_TESTCTRL_OPTIMIZATIONS,...) interface to ** selectively disable various optimizations.
69257. 490
69258. The new sorter
69259. Set the foreign key value to NULL
69260. New column to add to colset object
69261. Current state, using numbers defined in header comment
69262. True if a '-' sign preceded <value>
69263. ** Release all dynamic resources held by node-reader object *p.
69264. ** This function is called whenever an IOERR or FULL error that requires ** the pager to transition into the ERROR state may have occurred. ** The first argument is a pointer to the pager structure, the second ** the error-code about to be returned by a pager API function. The ** value returned is a copy of the second argument to this function. *** ** If the second argument is SQLITE_FULL, SQLITE_IOERR or one of the ** IOERR sub-codes, the pager enters the ERROR state and the error code ** is stored in Pager.errCode. While the pager remains in the ERROR state, ** all major API calls on the Pager will immediately return Pager.errCode. *** ** The ERROR state indicates that the contents of the pager-cache ** cannot be trusted. This state can be cleared by completely discarding ** the contents of the pager-cache. If a transaction was active when ** the persistent error occurred, then the rollback journal may need ** to be replayed to restore the contents of the database file (as if ** it were a hot-journal).
69265. ***** Include parse.h in the middle of sqliteInt.h *****
69266. **comment:** ** This function is called by the pcache layer when it has reached some ** soft memory limit. The first argument is a pointer to a Pager object ** (cast as a void*). The pager is always 'purgeable' (not an in-memory ** database). The second argument is a reference to a page that is ** currently dirty but has no outstanding references. The page ** is always associated with the Pager object passed as the first ** argument. *** ** The job of this function is to make pPg clean by writing its contents ** out to the database file, if possible. This may involve syncing the ** journal file. *** ** If successful, sqlite3PcacheMakeClean() is called on the page and ** SQLITE_OK returned. If an IO error occurs while trying to make the ** page clean, the IO error code is returned. If the page cannot be ** made clean for some other reason, but no error occurs, then SQLITE_OK ** is returned by sqlite3PcacheMakeClean() is not called.
- label:** code-design
69267. 202
69268. Generate code for all sub-queries in the FROM clause
69269. The text expression being built
69270. Tokenizer cursor
69271. ** Recompute all indices of all tables in all databases where the ** indices use the collating sequence pColl. If pColl==0 then recompute ** all indices everywhere.
69272. The table with the non-unique rowid
69273. The parent or outer SELECT statement
69274. ** Advance PmaReader pReadr to the next key in its PMA. Return SQLITE_OK if ** no error occurs, or an SQLite error code if one does.
69275. First WhereLoop to compare
69276. ** Convert bulk memory into a valid WhereLoop that can be passed ** to whereLoopClear harmlessly.
69277. Fall through into OP_Halt
69278. ** CAPI3REF: Name Of A Host Parameter ** METHOD: sqlite3_stmt *** ** ^The sqlite3_bind_parameter_name(P,N) interface returns ** the name of the N-th [SQL parameter] in the [prepared statement] P. ** ^{SQL parameters of the form "?NNN" or ":AAA" or "@AAA" or "\$AAA" ** have a name which is the string "?NNN" or ":AAA" or "\$AAA" ** respectively. ** In other words, the initial ":" or "\$" or "@" or "?" ** is included as part of the name.}^ ** ^Parameters of the form ":" without a following integer have no name ** and are referred to as "nameless" or "anonymous parameters". *** ** ^The first host parameter has an index of 1, not 0. ** ** ^If the value N is out of range or if the N-th parameter is ** nameless, then NULL is returned. ^The returned string is ** always in UTF-8 encoding even if the named parameter was ** originally specified as UTF-16 in [sqlite3_prepare16()], ** [sqlite3_prepare16_v2()], or [sqlite3_prepare16_v3()]. ** ** See also: [sqlite3_bind_blob|sqlite3_bind()], ** [sqlite3_bind_parameter_count()], and ** [sqlite3_bind_parameter_index()].
69279. Column to return position list for
69280. WHERE_* flags describing the plan
69281. Size of WAL
69282. ** Decrement the reference count on a page. If the page is clean and the ** reference count drops to 0, then it is made eligible for recycling.
69283. fdatsync() on HFS+ doesn't yet flush the file size if it changed correctly ** so currently we default to the macro that redefines fdatsync to fsync
69284. Do nothing if the pager does not have an open write transaction ** or at least a RESERVED lock. This function may be called when there ** is no write-transaction active but a RESERVED or greater lock is ** held under two circumstances: *** ** 1. After a successful hot-journal rollback, it is called with ** eState==PAGER_NONE and eLock==EXCLUSIVE_LOCK. *** ** 2. If a connection with locking_mode=exclusive holding an EXCLUSIVE ** lock switches back to locking_mode=normal and then executes a ** read-transaction, this function is called with eState==PAGER_READER ** and eLock==EXCLUSIVE_LOCK when the read-transaction is closed.
69285. The node containing the cell to be read
69286. ** Check to see if index pSrc is compatible as a source of data ** for index pDest in an insert transfer optimization. The rules ** for a compatible index: *** ** The index is over the same set of columns ** * The same DESC and ASC markings occurs on all columns ** * The same onError processing (OE_Abort, OE_Ignore, etc) ** * The same collating sequence on each column ** * The index has the exact same WHERE clause
69287. **
69288. pPage and up to two siblings
69289. Maximum instantaneous currentCount
69290. ***** Begin file vdbeaux.c *****
69291. selcollist ::= sclp STAR
69292. Search the list for a matching table
69293. A SELECT statement that will become the new view
69294. xCurrentTimeInt64
69295. The SET clause: list of column and new values
69296. Resolve function names
69297. ** PRAGMA temp_store ** PRAGMA temp_store = "default""memory""file" *** ** Return or set the local value of the temp_store flag. Changing ** the local value does not make changes to the disk file and the default ** value will be restored the next time the database is opened. *** ** Note that it is possible for the library compile-time options to ** override this setting
69298. Valid entries in aIdx[]
69299. Number of errors encountered while resolving names
69300. sortorder ::= ASC
69301. Current token offset in zIn[]

69302. This block codes the top of loop only. The complete loop is the ** following pseudocode (template 4): *** *** rewind temp table, if empty goto D ** C: loop over rows of intermediate table ** transfer values from intermediate table into <table> ** end loop ** D: ...
69303. If the cursor already points to the last entry, this is a no-op.
69304. ** One or more phrases that must appear within a certain token distance of ** each other within each matching document.
69305. Verify that all NOT NULL columns really are NOT NULL
69306. ID of current select for EXPLAIN output
69307. synopsis: if r[P1]>0 then r[P2]=r[P1]+max(0,r[P3]) else r[P2]=(-1)
69308. Index of docid<=x constraint, if present
69309. This error is now caught by the caller. ** Search for "xBestIndex malfunction" below
69310. ** Rounding constants for float->double conversion.
69311. where_opt ::=
69312. 115
69313. Numeric type of left operand
69314. the hash of the key modulo hash table size
69315. Opcode: Or P1 P2 P3 * * *** Synopsis: r[P3]=(r[P1] || r[P2]) *** *** Take the logical OR of the values in register P1 and P2 and ** store the answer in register P3. *** *** If either P1 or P2 is nonzero (true) then the result is 1 (true) ** even if the other input is NULL. A NULL and false or two NULLs ** give a NULL output.
69316. Call balance_quick() to create a new sibling of pPage on which ** to store the overflow cell. balance_quick() inserts a new cell ** into pParent, which may cause pParent overflow. If this ** happens, the next iteration of the do-loop will balance pParent ** use either balance_nonroot() or balance_deeper(). Until this ** happens, the overflow cell is stored in the aBalanceQuickSpace[] ** buffer. *** *** The purpose of the following assert() is to check that only a ** single call to balance_quick() is made for each call to this ** function. If this were not verified, a subtle bug involving reuse ** of the aBalanceQuickSpace[] might sneak in.
69317. Write JSON here
69318. ***** Start of MatchinfoBuffer code.
69319. COLLATE function from an ORDER BY clause term
69320. OUT: Docid value
69321. ** Print into memory obtained from sqlite3_malloc(). Omit the internal ** %-conversion extensions.
69322. Open mode flags
69323. OUT: Old value (or NULL pointer)
69324. ** This function sets the data directory or the temporary directory based on ** the provided arguments. The type argument must be 1 in order to set the ** data directory or 2 in order to set the temporary directory. The zValue ** argument is the name of the directory to use. The return value will be ** SQLITE_OK if successful.
69325. A frame is only valid if a checksum of the WAL header, ** all prior frames, the first 16 bytes of this frame-header, ** and the frame-data matches the checksum in the last 8 ** bytes of this frame-header.
69326. fts3_unicode2.c (functions generated by parsing unicode text files)
69327. ** Determine whether or not a cursor has moved from the position where ** it was last placed, or has been invalidated for any other reason. ** Cursors can move when the row they are pointing at is deleted out ** from under them, for example. Cursor might also move if a btree ** is rebalanced. *** *** Calling this routine with a NULL cursor pointer returns false. *** *** Use the separate sqlite3BtreeCursorRestore() routine to restore a cursor ** back to where it ought to be if this routine returns true.
69328. The vm under construction
69329. Write content into this buffer
69330. Required table locks for shared-cache mode
69331. Iterate through all dirty pages currently stored in the cache. This ** interface is only available if SQLITE_CHECK_PAGES is defined when the ** library is built.
69332. Source database page data
69333. ** Test to see whether or not a table is a virtual table. This is ** done as a macro so that it will be optimized out when virtual ** table support is omitted from the build.
69334. Pointer to the appropriate info structure
69335. Condition 5
69336. Size of buffer pBuf in bytes
69337. Pointer to buffer containing root node
69338. Size of aFold[] in bytes
69339. ***** End of util.c *****
69340. Statement for DELETE ops
69341. **comment:** ** Generate a human-readable explanation of an expression tree.
 label: code-design
69342. same as TK_STAR, in1, in2, out3
69343. Find the Fts3SegReader object with Fts3SegReader.iIdx==i. It is hiding ** somewhere in the pCsr->apSegment[] array.
69344. Fold to lower case
69345. position-list size field value
69346. ** Remember the SQL string for a prepared statement.
69347. Argument to xCommitCallback()
69348. If the table is not located in the temp-db (in which case NULL is ** returned, loop through the tables list of triggers. For each trigger ** that is not part of the temp-db schema, add a clause to the WHERE ** expression being built up in zWhere.
69349. Deinitialize the memory allocator
69350. ** Finish off a string by making sure it is zero-terminated. ** Return a pointer to the resulting string. Return a NULL ** pointer if any kind of error was encountered.
69351. Fetch the info entry for the field
69352. IMP: R-42420-56072
69353. Opcode: SorterCompare P1 P2 P3 P4 ** Synopsis: if key(P1)!=trim(r[P3],P4) goto P2 ** ** P1 is a sorter cursor. This instruction compares a prefix of the ** record blob in register P3 against a prefix of the entry that ** the sorter cursor currently points to. Only the first P4 fields ** of r[P3] and the sorter record are compared. *** *** If either P3 or the sorter contains a NULL in one of their significant ** fields (not counting the P4 fields at the end which are ignored) then ** the comparison is assumed to be equal. *** *** Fall through to next instruction if the two records compare equal to ** each other. Jump to P2 if they are different.
69354. Build the Trigger object
69355. Amount to increment deferred counter by
69356. Index in pTabList->a[] of table being read
69357. Iterate through levels, segments
69358. ***** End of func.c *****
69359. Cursor to scan for
69360. stl_prefix
69361. IDF for each phrase
69362. ***** Begin file fts3Int.h *****
69363. ** This function is called to change the WAL subsystem into or out ** of locking_mode=EXCLUSIVE. *** *** If op is zero, then attempt to change from locking_mode=EXCLUSIVE ** into locking_mode=NORMAL. This means that we must acquire a lock ** on the pWal->readLock byte. If the WAL is already in locking_mode=NORMAL ** or if the acquisition of the lock fails, then return 0. If the ** transition out of exclusive-mode is successful, return 1. This ** operation must occur while the pager is still holding the exclusive ** lock on the main database file. *** *** If op is one, then change from locking_mode=NORMAL into ** locking_mode=EXCLUSIVE. This means that the pWal->readLock must ** be released. Return 1 if the transition is made and 0 if the ** WAL is already in exclusive-locking mode - meaning that this ** routine is a no-op. The pager must already hold the exclusive lock ** on the main database file before invoking this operation. *** *** If op is negative, then do a dry-run of the op==1 case but do ** not actually change anything. The pager uses this to see if it ** should acquire the database exclusive lock prior to invoking ** the op==1 case.
69364. Either SQLITE_ATTACH or SQLITE_DETACH

69365. ** Count the number of fields (a.k.a. columns) in the record given by ** pKey,nKey. The verify that this count is less than or equal to the ** limit given by pKeyInfo->nField + pKeyInfo->nXField. *** If this constraint is not satisfied, it means that the high-speed ** vdbeRecordCompareInt() and vdbeRecordCompareString() routines will ** not work correctly. If this assert() ever fires, it probably means ** that the KeyInfo.nField or KeyInfo.nXField values were computed ** incorrectly.

69366. ***** Begin file mem5.c *****

69367. Pointer to new Btree object written here

69368. ** At this point, there should be no outstanding memory allocations on ** the heap. Also, since both the master and memsys locks are currently ** being held by us, no other function (i.e. from another thread) should ** be able to even access the heap. Attempt to destroy and recreate our ** isolated Win32 native heap now.

69369. Length of the zCanonicalName[] string

69370. Values for parameters to the SQL function

69371. Generate code to handle the case of A==B

69372. Do nothing

69373. Check to see if pWLoop should be added to the set of ** mxChoice best-so-far paths. *** First look for an existing path among best-so-far paths ** that covers the same set of loops and has the same isOrdered ** setting as the current path candidate. *** The term "((pTo->isOrdered^isOrdered)&0x80)==0" is equivalent ** to (pTo->isOrdered==(-1))==((isOrdered==(-1))" for the range ** of legal values for isOrdered, -1..64.

69374. ** Return the database handle used by pRbu.

69375. ** This function is responsible for invoking the collation factory callback ** or substituting a collation sequence of a different encoding when the ** requested collation sequence is not available in the desired encoding. *** If it is not NULL, then pColl must point to the database native encoding ** collation sequence with name zName, length nName. *** The return value is either the collation sequence to be used in database ** db for collation type name zName, length nName, or NULL, if no collation ** sequence can be found. If no collation is found, leave an error message. *** See also: sqlite3LocateCollSeq(), sqlite3FindCollSeq()

69376. ** This function is called during initialization if a static buffer is ** supplied to use for the page-cache by passing the SQLITE_CONFIG_PAGECACHE ** verb to sqlite3_config(). Parameter pBuf points to an allocation large ** enough to contain 'n' buffers of 'sz' bytes each. *** This routine is called from sqlite3_initialize() and so it is guaranteed ** to be serialized already. There is no need for further mutexing.

69377. OUT: Acquired node

69378. The data is distributed over two or more pages. Copy it into the ** Fts5Iter.poslist buffer and then set the output pointer to point ** to this buffer.

69379. Level to select (relative level)

69380. True if argument iFrom is valid

69381. NC_IsCheck or NC_PartIdx or NC_IdxExpr

69382. Object reference count

69383. Do not allow any kind of write-lock on a read-only database

69384. DO NOT EDIT! ** This file is automatically generated by the script at ** ..//tool/mkpragmatab.tcl. To update the set of pragmas, edit ** that script and rerun it.

69385. ***** Include msvc.h in the middle of sqliteInt.h *****

69386. No duplicates

69387. ** A wrapper around sqlite3_bind_value() that detects an extra problem. ** See comments in the body of this function for details.

69388. ** Try to increase the size of the parser stack. Return the number ** of errors. Return 0 on success.

69389. WHERE clause of trigger step

69390. Restriction (16)

69391. Payload with which to construct the cell

69392. Space to hold the Lemon-generated Parser object

69393. Pointer to BtShared that this page is part of

69394. True if uppercase==lowercase

69395. Name of the table or view

69396. Index of new output segment

69397. True if current change was indirect

69398. !SQLITE_OMIT_TRACE && SQLITE_ENABLE_IOTRACE

69399. Parameters to CREATE TABLE statement

69400. Maximum allowed page size

69401. SQLITE_OMIT_AUTOINCREMENT

69402. defined(__APPLE__) && SQLITE_ENABLE_LOCKING_STYLE

69403. ** Expression pRight, which is the RHS of a comparison operation, is ** either a vector of n elements or, if n==1, a scalar expression. ** Before the comparison operation, affinity zAff is to be applied ** to the pRight values. This function modifies characters within the ** affinity string to SQLITE_AFF_BLOB if either: ** *** the comparison will be performed with no affinity, or *** the affinity change in zAff is guaranteed not to change the value.

69404. TODO1: Can use (writer.pgidx.n==0) instead of bFirstTermInPage

69405. True after initialization has finished

69406. ** Determine the average document (row) size in pages. If successful, ** write this value to *pnPage and return SQLITE_OK. Otherwise, return ** an SQLite error code. *** The average document size in pages is calculated by first calculating ** determining the average size in bytes, B. If B is less than the amount ** of data that will fit on a single leaf page of an intkey table in ** this database, then the average docsize is 1. Otherwise, it is 1 plus ** the number of overflow pages consumed by a record B bytes in size.

69407. Merge engine thread reads data from

69408. Finally, rollback pages from the sub-journal. Page that were ** previously rolled back out of the main journal (and are hence in pDone) ** will be skipped. Out-of-range pages are also skipped.

69409. Cache for fts3CursorSeekStmt()

69410. term ::= INTEGER

69411. Sanity check: Assert that the IntValue is non-negative if it exists

69412. ** If pExpr is an ASC iterator, this function returns a value with the ** same sign as: *** (iLhs - iRhs) ** ** Otherwise, if this is a DESC iterator, the opposite is returned: *** (iRhs - iLhs)

69413. Mask of cursors that are not available

69414. ** Called when a page of data is written to offset iOff of the database ** file while the rbu handle is in capture mode. Record the page number ** of the page being written in the aFrame[] array.

69415. Connections from one process only

69416. explicit IPK column

69417. Input changeset

69418. !defined(SQLITE_TEST) && !defined(SQLITE_OMIT_RANDOMNESS)

69419. This undoes the effect of the HintPop() above - so that no entry ** is removed from the hint blob.

69420. ON CONFLICT strategy

69421. Pages of work to do

69422. If p==0 (unmap the entire file) then there must be no outstanding ** xFetch references. Or, if p!=0 (meaning it is an xFetch reference), ** then there must be at least one outstanding.

69423. Pointer to virtual tables needing locking

69424. An index to being analyzed

69425. ***** Begin file vdbeblob.c *****

69426. Number of bytes in input

69427. Ignore terms with these prereqs

69428. ** This is the xFilter interface for the virtual table. See ** the virtual table xFilter method documentation for additional ** information. *** If idxNum==FTS3_FULLSCAN_SEARCH then do a full table scan against ** the %_content table. *** If idxNum==FTS3_DOCID_SEARCH then do a docid lookup for a single entry ** in the %_content table. *** If idxNum>=FTS3_FULLTEXT_SEARCH then use the full text index. The ** column on the left-hand side of the MATCH operator is column ** number idxNum-FTS3_FULLTEXT_SEARCH, 0 indexed. argv[0] is the right-hand ** side of the MATCH operator.

69429. no-op

69430. 40..47 @ABCDEFG

69431. **comment:** Verify that no more than two scratch allocation per thread ** is outstanding at one time. (This is only checked in the ** single-threaded case since checking in the multi-threaded case ** would be much more complicated.)

label: code-design

69432. ** Iterator pIter was used to iterate through the input segments of on an ** incremental merge operation. This function is called if the incremental ** merge step has finished but the input has not been completely exhausted.

69433. Request permission to read the parent key columns. If the ** authorization callback returns SQLITE_IGNORE, behave as if any ** values read from the parent table are NULL.

69434. Write the hash value here

69435. ***** End of fts3_unicode2.c *****

69436. ** CAPI3REF: The pre-update hook. ** ** ^These interfaces are only available if SQLite is compiled using the ** [SQLITE_ENABLE_PREUPDATE_HOOK] compile-time option. ** ** ^The [sqlite3_preupdate_hook()] interface registers a callback function ** that is invoked prior to each [INSERT], [UPDATE], and [DELETE] operation ** on a database table. ** ^At most one preupdate hook may be registered at a time on a single ** [database connection]; each call to [sqlite3_preupdate_hook()] overrides ** the previous setting. ** ^The preupdate hook is disabled by invoking [sqlite3_preupdate_hook()] ** with a NULL pointer as the second parameter. ** ^The third parameter to [sqlite3_preupdate_hook()] is passed through as ** the first parameter to callbacks. ** ** ^The preupdate hook only fires for changes to real database tables; the ** preupdate hook is not invoked for changes to [virtual tables] or to ** system tables like sqlite_master or sqlite_stat1. ** ** ^The second parameter to the preupdate callback is a pointer to ** the [database connection] that registered the preupdate hook. ** ^The third parameter to the preupdate callback is one of the constants ** [SQLITE_INSERT], [SQLITE_DELETE], or [SQLITE_UPDATE] to identify the ** kind of update operation that is about to occur. ** ^The fourth parameter to the preupdate callback is the name of the ** database within the database connection that is being modified. This ** will be "main" for the main database or "temp" for TEMP tables or ** the name given after the AS keyword in the [ATTACH] statement for attached ** databases.)^ ** ^The fifth parameter to the preupdate callback is the name of the ** table that is being modified. ** ** For an UPDATE or DELETE operation on a [rowid table], the sixth ** parameter passed to the preupdate callback is the initial [rowid] of the ** row being modified or deleted. For an INSERT operation on a rowid table, ** or any operation on a WITHOUT ROWID table, the value of the sixth ** parameter is undefined. For an INSERT or UPDATE on a rowid table the ** seventh parameter is the final rowid value of the row being inserted ** or updated. The value of the seventh parameter passed to the callback ** function is not defined for operations on WITHOUT ROWID tables, or for ** INSERT operations on rowid tables. ** ** The [sqlite3_preupdate_old()], [sqlite3_preupdate_new()], [sqlite3_preupdate_count()], and [sqlite3_preupdate_depth()] interfaces ** provide additional information about a preupdate event. These routines ** may only be called from within a preupdate callback. Invoking any of ** these routines from outside of a preupdate callback or with a ** [database connection] pointer that is different from the one supplied ** to the preupdate callback results in undefined and probably undesirable ** behavior. ** ** ^The [sqlite3_preupdate_count(D)] interface returns the number of columns ** in the row that is being inserted, updated, or deleted. ** ** ^The [sqlite3_preupdate_old(D,N,P)] interface writes into P a pointer to ** a [protected sqlite3_value] that contains the value of the Nth column of ** the table row before it is updated. The N parameter must be between 0 ** and one less than the number of columns or the behavior will be ** undefined. This must only be used within SQLITE_UPDATE and SQLITE_DELETE ** preupdate callbacks; if it is used by an SQLITE_INSERT callback then the ** behavior is undefined. The [sqlite3_value] that P points to ** will be destroyed when the preupdate callback returns. ** ** ^The [sqlite3_preupdate_new(D,N,P)] interface writes into P a pointer to ** a [protected sqlite3_value] that contains the value of the Nth column of ** the table row after it is updated. The N parameter must be between 0 ** and one less than the number of columns or the behavior will be ** undefined. This must only be used within SQLITE_INSERT and SQLITE_UPDATE ** preupdate callbacks; if it is used by an SQLITE_DELETE callback then the ** behavior is undefined. The [sqlite3_value] that P points to ** will be destroyed when the preupdate callback returns. ** ** ^The [sqlite3_preupdate_depth(D)] interface returns 0 if the preupdate ** callback was invoked as a result of a direct insert, update, or delete ** operation; or 1 for inserts, updates, or deletes invoked by top-level ** triggers; or 2 for changes resulting from triggers called by top-level ** triggers; and so forth. ** ** See also: [sqlite3_update_hook()]

69437. The first page of the wal-index must be mapped at this point.

69438. Copy entry from i2

69439. ** Initialize a WHERE clause scanner object. Return a pointer to the ** first match. Return NULL if there are no matches. ** ** The scanner will be searching the WHERE clause pWC. It will look ** for terms of the form "X <op> <expr>" where X is column iColumn of table ** iCur. Or if pIdx!=0 then X is column iColumn of index pIdx. pIdx ** must be one of the indexes of table iCur. ** ** The <op> must be one of the operators described by opMask. ** ** If the search is for X and the WHERE clause contains terms of the ** form X=Y then this routine might also return terms of the form ** "Y <op> <expr>". The number of levels of transitivity is limited, ** but is enough to handle most commonly occurring SQL statements. ** ** If X is not the INTEGER PRIMARY KEY then X must be compatible with ** index pIdx.

69440. ** This function is called as part of sqlite3rbu_open() when initializing ** an rbu handle in OAL stage. If the rbu update has not started (i.e. ** the rbu_state table was empty) it is a no-op. Otherwise, it arranges ** things so that the next call to sqlite3rbu_step() continues on from ** where the previous rbu handle left off. ** ** If an error occurs, an error code and error message are left in the ** rbu handle passed as the first argument.

69441. IN/OUT: Number of elements in aList[]

69442. Bytes of space to allocate for this node

69443. Pointer to the root node of a tree

69444. The database file to check

69445. The key for INTKEY tables, or nPayload otherwise

69446. Parsing context

69447. ** The code above is the AFP lock implementation. The code is specific ** to MacOSX and does not work on other unix platforms. No alternative ** is available. If you don't compile for a mac, then the "unix-afp" ** VFS is not available. ** ***** End of the AFP lock implementation *****

69448. Merge PMAs together

69449. Write the number of columns of result here

69450. Prior index

69451. xCheckReservedLock method

69452. ** Code an output subroutine for a coroutine implementation of a ** SELECT statement. ** ** The data to be output is contained in pIn->iSdst. There are ** pIn->nSdst columns to be output. pDest is where the output should ** be sent. ** ** regReturn is the number of the register holding the subroutine ** return address. ** ** If regPrev>0 then it is the first register in a vector that ** records the previous output. mem[regPrev] is a flag that is false ** if there has been no previous output. If regPrev>0 then code is ** generated to suppress duplicates. pKeyInfo is used for comparing ** keys. ** ** If the LIMIT found in p->iLimit is reached, jump immediately to ** iBreak.

69453. input

69454. Column ref to child table

69455. Address of top of loop

69456. Number of bytes in lhs input

69457. Return a pointer to the enlarged SrcList

69458. End of first segment of main-journal records

69459. Poslist buffer to iterate through

69460. 159

69461. 170

69462. Skip this term for now. We revisit it when we process the ** corresponding TERM_VIRTUAL term

69463. ** Do the comparison necessary to populate pIter->aFirst[iOut]. ** ** If the returned value is non-zero, then it is the index of an entry ** in the pIter->aSeg[] array that is (a) not at EOF, and (b) pointing ** to a key that is a duplicate of another, higher priority, ** segment-iterator in the pSeg->aSeg[] array.

69464. ** If the library is compiled to omit the full-scale date and time ** handling (to get a smaller binary), the following minimal version ** of the functions current_time(), current_date() and current_timestamp() ** are included instead. This is to support column declarations that ** include "DEFAULT CURRENT_TIME" etc. ** ** This function uses the C-library functions time(), gmtime() ** and strftime(). The format string to pass to strftime() is supplied ** as the user-data for the function.

69465. Argument to SQLITE_FCNTL_PRAGMA

69466. #include "fts5parse.h"

69467. Level/index to create new segment at

69468. 198
69469. Buffer to serialize record into
69470. ** xColumn - Return a column value.
69471. Columns that change in an UPDATE statement
69472. Callback return code
69473. ** Implementation of the matchinfo() function for FTS3
69474. **comment:** Serial types 12 or greater are strings and blobs (greater than ** numbers). Types 10 and 11 are currently "reserved for future ** use", so it doesn't really matter what the results of comparing ** them to numeric values are.
label: code-design
69475. Busy handler timeout, in msec
69476. 257
69477. Current parent of p
69478. Error code
69479. !SQLITE OMIT_TRIGGER
69480. ** This version of the memory allocator is the default. It is ** used when no other memory allocator is specified using compile-time ** macros.
69481. Date at which to calculate offset
69482. * This is cache size used in the calculation of the initial size of the * Win32-specific heap. It cannot be negative.
69483. NULL for tables. Points to definition if a view.
69484. Make sure every page in the file is referenced
69485. (313) nmnum ::= DEFAULT
69486. ** Return the number of terms in the iPhrase'th phrase in pExpr.
69487. Acts as query when no argument
69488. 217
69489. synopsis: if(cursor[P1].ctr++) pc = P2
69490. Records with omitted columns are only allowed for schema format ** version 2 and later (SQLite version 3.1.4, 2005-02-20).
69491. a: p0<<14 | p2 (unmasked)
69492. ** Release a page reference. *** If the number of references to the page drop to zero, then the ** page is added to the LRU list. When all references to all pages ** are released, a rollback occurs and the lock on the database is ** removed.
69493. Number of bytes of space to allocate
69494. Next registered auxiliary function
69495. ** 2007 October 14 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the C functions that implement a memory **
allocation subsystem for use by SQLite. *** This version of the memory allocation subsystem omits all ** use of malloc(). The application gives SQLite a block of memory ** before calling sqlite3_initialize() from which allocations ** are made and returned by the xMalloc() and xRealloc() ** implementations. Once sqlite3_initialize() has been called, ** the amount of memory available to SQLite is fixed and cannot ** be changed. *** This version of the memory allocation subsystem is included ** in the build only if SQLITE_ENABLE_MEMSYS5 is defined. *** This memory allocator uses the following algorithm: *** 1. All memory allocation sizes are rounded up to a power of 2. *** 2. If two adjacent free blocks are the halves of a larger block, ** then the two blocks are coalesced into the single larger block. *** 3. New memory is allocated from the first available free block. *** This algorithm is described in: J. M. Robson. "Bounds for Some Functions ** Concerning Dynamic Storage Allocation". Journal of the Association for ** Computing Machinery, Volume 21, Number 8, July 1974, pages 491-499. *** Let n be the size of the largest allocation divided by the minimum ** allocation size (after rounding all sizes up to a power of 2.) Let M ** be the maximum amount of memory ever outstanding at one time. Let ** N be the total amount of memory available for allocation. Robson ** proved that this memory allocator will never breakdown due to ** fragmentation as long as the following constraint holds: *** N >= M*(1 + log2(n)/2) - n + 1 *** The sqlite3_status() logic tracks the maximum values of n and M so ** that an application can, at any time, verify this constraint.
69496. FROM clause cannot contain a subquery
69497. **comment:** ** CAPI3REF: Formatted String Printing Functions *** These routines are work-alikes of the "printf()" family of functions ** from the standard C library. ** These routines understand most of the common K&R formatting options, ** plus some additional non-standard formats, detailed below. ** Note that some of the more obscure formatting options from recent ** C-library standards are omitted from this implementation. *** ^The sqlite3_mprintf() and sqlite3_vmpprintf() routines write their ** results into memory obtained from [sqlite3_malloc()]. ** The strings returned by these two routines should be ** released by [sqlite3_free()]. ^Both routines return a ** NULL pointer if [sqlite3_malloc()] is unable to allocate enough ** memory to hold the resulting string. **
*** ^The sqlite3_snprintf() routine is similar to "snprintf()" from ** the standard C library. The result is written into the ** buffer supplied as the second parameter whose size is given by ** the first parameter. Note that the order of the ** first two parameters is reversed from sprintf().^ This is an ** historical accident that cannot be fixed without breaking ** backwards compatibility. ^Note also that sqlite3_snprintf() ** returns a pointer to its buffer instead of the number of ** characters actually written into the buffer.^ We admit that ** the number of characters written would be a more useful return ** value but we cannot change the implementation of sqlite3_snprintf() ** now without breaking compatibility. *** ^As long as the buffer size is greater than zero, sqlite3_snprintf() ** guarantees that the buffer is always zero-terminated. ^The first ** parameter "n" is the total size of the buffer, including space for ** the zero terminator. So the longest string that can be completely ** written will be n-1 characters. *** ^The sqlite3_vsnprintf() routine is a varargs version of sqlite3_snprintf(). *** These routines all implement some additional formatting ** options that are useful for constructing SQL statements. ** All of the usual printf() formatting options apply. In addition, there ** is are "%q", "%Q", "%w" and "%z" options. *** ^The %q option works like %s in that it substitutes a null-terminated ** string from the argument list. But %q also doubles every \" character. ** %q is designed for use inside a string literal.^ By doubling each \" ** character it escapes that character and allows it to be inserted into ** the string. *** For example, assume the string variable zText contains text as follows: *** <blockquote><pre> ** char *zText = "It's a happy day!"; ** </pre></blockquote> *** One can use this text in an SQL statement as follows: *** <blockquote><pre> ** char *zSQL = sqlite3_mprintf("INSERT INTO table VALUES(%q)", zText); ** sqlite3_exec(db, zSQL, 0, 0, 0); ** sqlite3_free(zSQL); ** </pre></blockquote> *** Because the %q format string is used, the \" character in zText ** is escaped and the SQL generated is as follows: *** <blockquote><pre> ** INSERT INTO table1 VALUES('It's a happy day!') ** </pre></blockquote> *** This is correct. Had we used %s instead of %q, the generated SQL ** would have looked like this: *** <blockquote><pre> ** INSERT INTO table1 VALUES(It's a happy day!); ** </pre></blockquote> *** This second example is an SQL syntax error. As a general rule you should ** always use %q instead of %s when inserting text into a string literal. *** ^The %Q option works like %q except it also adds single quotes around ** the outside of the total string. Additionally, if the parameter in the ** argument list is a NULL pointer, %Q substitutes the text "NULL" (without ** single quotes).^ So, for example, one could say: *** <blockquote><pre> ** char *zSQL = sqlite3_mprintf("INSERT INTO table VALUES(%Q)", zText); ** sqlite3_exec(db, zSQL, 0, 0, 0); ** sqlite3_free(zSQL); ** </pre></blockquote> *** The code above will render a correct SQL statement in the zSQL ** variable even if the zText variable is a NULL pointer. *** ^The "%w" formatting option is like "%q" except that it expects to ** be contained within double-quotes instead of single quotes, and it ** escapes the double-quote character instead of the single-quote ** character.^ The "%w" formatting option is intended for safely inserting ** table and column names into a constructed SQL statement. *** ^The "%z" formatting option works like "%s" but with the ** addition that after the string has been read and copied into ** the result, [sqlite3_free()] is called on the input string.^
label: code-design
69498. Array of mxSample Stat4Sample objects
69499. Either DELETE or UPDATE. For err msgs.
69500. 288
69501. OF
69502. 1 if the conch is held, -1 if lockless
69503. WRITEABLE implies DIRTY
69504. If there are outstanding sqlite3_stmt or sqlite3_backup objects ** or if the connection has not yet been closed by sqlite3_close_v2(), ** then just leave the mutex and return.
69505. An array of root pages numbers for individual trees
69506. jump here if parent key found
69507. For looping over indices of the table
69508. ** The maximum number of arguments to an SQL function.
69509. Grow the sqlite3.aVTrans array if required

69510. The first fields of the two keys are equal and there are no trailing ** fields. Return pPKey2->default_rc in this case.
69511. Bytes of data before flushing
69512. ** Allowed values for BtShared.btsFlags
69513. 271
69514. ** Attempt to load an SQLite extension library contained in the file ** zFile. The entry point is zProc. zProc may be 0 in which case a ** default entry point name (sqlite3_extension_init) is used. Use ** of the default name is recommended. ** ** Return SQLITE_OK on success and SQLITE_ERROR if something goes wrong. ** ** If an error occurs and pzErrMsg is not 0, then fill *pzErrMsg with ** error message text. The calling function should free this memory ** by calling sqlite3DbFree(db,).
69515. a &= (0x7f<<28)|(0x7f<<14)|(0x7f);
69516. Write the BLOB here
69517. Condition is always true or false
69518. ** 2008 October 28 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file contains a no-op memory allocation drivers for use when ** SQLITE_ZERO_MALLOC is defined. The allocation drivers implemented ** here always fail. SQLite will not operate with these drivers. These ** are merely placeholders. Real drivers must be substituted using ** sqlite3_config() before SQLite will operate.
69519. ** Unlock all of the btrees previously locked by a call to sqlite3VdbeEnter().
69520. More terms for ARRAY and OBJECT
69521. EVIDENCE-OF: R-01506-11053 The first integer on a freelist trunk page ** is the page number of the next freelist trunk page in the list or ** zero if this is the last freelist trunk page.
69522. Database containing the table being updated
69523. The end of the for(;;) loop the loops through opcodes
69524. Default ON CONFLICT policy for trigger steps
69525. True if sqlite3RowSetNext() has been called
69526. Array of values to insert
69527. IMP: R-39564-36305 The sqlite_compileoption_used() SQL ** function is a wrapper around the sqlite3_compileoption_used() C/C++ ** function.
69528. ccons ::= REFERENCES nm eidlist_opt refargs
69529. Level to iterate (-1 for all)
69530. Do allow the journalmode of an in-memory database to be set to ** anything other than MEMORY or OFF
69531. If the right-hand side is also a column, then the affinities ** of both right and left sides must be such that no type ** conversions are required on the right. (Ticket #2249)
69532. ** The cache of the wal-index header must be valid to call this function. ** Return the page-size in bytes used by the database.
69533. If there is more than one table or sub-select in the FROM clause of ** this query, then it will not be possible to show that the DISTINCT ** clause is redundant.
69534. ** Advance the cursor to the next row in the %_content table that ** matches the search criteria. For a MATCH search, this will be ** the next row that matches. For a full-table scan, this will be ** simply the next row in the %_content table. For a docid lookup, ** this routine simply sets the EOF flag. ** ** Return SQLITE_OK if nothing goes wrong. SQLITE_OK is returned ** even if we reach end-of-file. The fts3EofMethod() will be called ** subsequently to determine whether or not an EOF was hit.
69535. pNew
69536. TRUE for a TEMPORARY view
69537. FTS3 Cursor
69538. Database key for encryption extension
69539. OUT: Array of boolean - true for PK cols
69540. **comment:** FTS5_PLAN_XXX value
 label: code-design
69541. Do a binary search to find the first sample greater than or equal ** to pRec. If pRec contains a single field, the set of samples to search ** is simply the aSample[] array. If the samples in aSample[] contain more ** than one fields, all fields following the first are ignored. ** ** If pRec contains N fields, where N is more than one, then as well as the ** samples in aSample[] (truncated to N fields), the search also has to ** consider prefixes of those samples. For example, if the set of samples ** in aSample is: ** ** aSample[0] = (a, 5) ** aSample[1] = (a, 10) ** aSample[2] = (b, 5) ** aSample[3] = (c, 100) ** aSample[4] = (c, 105) ** ** Then the search space should ideally be the samples above and the ** unique prefixes [a], [b] and [c]. But since that is hard to organize, ** the code actually searches this set: ** ** 0: (a) ** 1: (a, 5) ** 2: (a, 10) ** 3: (a, 10) ** 4: (b) ** 5: (b, 5) ** 6: (c) ** 7: (c, 100) ** 8: (c, 105) ** 9: (c, 105) ** ** For each sample in the aSample[] array, N samples are present in the ** effective sample array. In the above, samples 0 and 1 are based on ** sample aSample[0]. Samples 2 and 3 on aSample[1] etc. ** ** Often, sample i of each block of N effective samples has (i+1) fields. ** Except, each sample may be extended to ensure that it is greater than or ** equal to the previous sample in the array. For example, in the above, ** sample 2 is the first sample of a block of N samples, so at first it ** appears that it should be 1 field in size. However, that would make it ** smaller than sample 1, so the binary search would not work. As a result, ** it is extended to two fields. The duplicates that this creates do not ** cause any problems.
69542. Buffer containing position list
69543. IS
69544. Number of extra registers before regResult
69545. True for a freelist. False for overflow page list
69546. A column of pTab
69547. Name context for resolving pE
69548. First disjunct
69549. If there are TEMP triggers on this table, modify the sqlite_temp_master ** table. Don't do this if the table being ALTERed is itself located in ** the temp database.
69550. Name of content table
69551. Height of sub-tree rooted at pCell
69552. IMP: R-00293-64994
69553. ** Make sure the hard limits are set to reasonable values
69554. **comment:** TODO: Is it safe to use Pager.dbFileSize here?
 label: code-design
69555. ** Determine whether triggers are recursive by default. This can be ** changed at run-time using a pragma.
69556. Pointer to the start of payload
69557. Free the position-lists accumulated for each deferred token above.
69558. Read the PK into an array of temp registers.
69559. Rows come from source in GROUP BY order
69560. The table to be updated
69561. Cursor to iterate through aRight
69562. SQLITE OMIT_AUTOMATIC_INDEX
69563. ** Release all resources associated with an sqlite3_backup* handle.
69564. ** Change the lock state for a shared-memory segment. ** ** Note that the relationship between SHARED and EXCLUSIVE locks is a little ** different here than in posix. In xShmLock(), one can go from unlocked ** to shared and back or from unlocked to exclusive and back. But one may ** not go from shared to exclusive or from exclusive to shared.
69565. True if compound joined by UNION [ALL]
69566. Finish the loop through table entries that match term pOrTerm.
69567. **comment:** ** Initialize the object pIter to point to term pTerm/nTerm within the ** in-memory hash table. If there is no such term in the hash-table, the ** iterator is set to EOF. ** ** If an error occurs, Fts5Index.rc is set to an appropriate error code. If ** an error has already occurred when this function is called, it is a no-op.
 label: code-design
69568. tcons ::= UNIQUE LP sortlist RP onconff
69569. pgsize

69570. Number of nested calls to VdbeExec()
69571. Name of the file
69572. ** An instance of this structure holds information about the ** LIMIT clause of a SELECT statement.
69573. At this point p points to that preceding byte without the 0x80 bit ** set. So to find the start of the poslist, skip forward 2 bytes then ** over a varint. ***
Normally. The other case is that p==pStart and the poslist to return ** is the first in the doclist. In this case do not skip forward 2 bytes. ** The second part of the if condition (c==0 && *ppPoslist>&p[2]) ** is required for cases where the first byte of a doclist and the ** doclist is empty. For example, if the first docid is 10, a doclist ** that begins with: *** 0x0A 0x00 <next docid delta varint>
69574. Acquire a PENDING lock
69575. Keep track of the number of rows to be deleted
69576. For DOCID_CMP macro
69577. No table available. Use comparisons
69578. P1
69579. Neither table may have __hidden__ columns
69580. ** Implementation of the xBestIndex method.
69581. Db object for the newly attached database
69582. xGetLastError
69583. ** Lock the file with the lock specified by parameter eFileLock - one ** of the following: *** (1) SHARED_LOCK ** (2) RESERVED_LOCK ** (3) PENDING_LOCK ** (4) EXCLUSIVE_LOCK *** Sometimes when requesting one lock state, additional lock states ** are inserted in between. The locking might fail on one of the later ** transitions leaving the lock state different from what it started but ** still short of its goal. The following chart shows the allowed ** transitions and the inserted intermediate states: *** UNLOCKED -> SHARED ** SHARED -> RESERVED ** SHARED -> (PENDING) -> EXCLUSIVE ** RESERVED -> (PENDING) -> EXCLUSIVE ** PENDING -> EXCLUSIVE *** This routine will only increase a lock. Use the sqlite3OsUnlock() ** routine to lower a locking level.
69584. If not in RBU_STAGE_OAL, allow this call to pass through. Or, if this ** rbu is in the RBU_STAGE_OAL state, use heap memory for *-shm space ** instead of a file on disk.
69585. !defined(SQLITE OMIT CHECK)
69586. Page obtained by prior PcacheFetch() call
69587. ***** End of vdbesort.c *****
69588. If the cache has been modified but the savepoint cannot be rolled ** back journal_mode=off, put the pager in the error state. This way, ** if the VFS used by this pager includes ZipVFS, the entire transaction ** can be rolled back at the ZipVFS level.
69589. Otherwise see if some other process holds it.
69590. Check that malloc() has not failed. If it has, return early.
69591. Docid to add or remove data from
69592. Memory is not available in the SQLITE_CONFIG_PAGECACHE pool. Get ** it from sqlite3Malloc instead.
69593. Locate the required virtual table module
69594. ** Lower the locking level on file descriptor pFile to eFileLock. eFileLock ** must be either NO_LOCK or SHARED_LOCK. *** If the locking level of the file descriptor is already at or below ** the requested locking level, this routine is a no-op.
69595. Estimated number of rows in this table
69596. Next slot in aIndex[] not yet returned
69597. **comment:** Malloc must have failed inside HashInsert()
label: code-design
69598. ** Numeric constants that encode the journalmode. *** The numeric values encoded here (other than PAGER_JOURNALMODE_QUERY) ** are exposed in the API via the "PRAGMA journal_mode" command and ** therefore cannot be changed without a compatibility break.
69599. ** Apply advisory locks for all n bytes beginning at ofst.
69600. ** An instance of the following structure is used to store state while ** iterating through a multi-column position-list corresponding to the ** hits for a single phrase on a single row in order to calculate the ** values for a matchinfo() FTS3_MATCHINFO_LCS request.
69601. Number of pages to use in the cache
69602. ** Make sure pBt->pTmpSpace points to an allocation of ** MX_CELL_SIZE(pBt) bytes with a 4-byte prefix for a left-child ** pointer.
69603. True to trace changes
69604. ** Find (an approximate) sum of two LogEst values. This computation is ** not a simple "+" operator because LogEst is stored as a logarithmic ** value. **
69605. Root of NEAR expression
69606. **comment:** If this scan uses an index, make VDBE code substitutions to read data ** from the index instead of from the table where possible. In some cases ** this optimization prevents the table from ever being read, which can ** yield a significant performance boost. *** Calls to the code generator in between sqlite3WhereBegin and ** sqlite3WhereEnd will have created code that references the table ** directly. This loop scans all that code looking for opcodes ** that reference the table and converts them into opcodes that ** reference the index.
label: code-design
69607. Temp storage space for pCell, if needed
69608. Value returned by sqlite3OsAccess()
69609. Opcode: Permutation * * * P4 * * * Set the permutation used by the OP_Compare operator in the next ** instruction. The permutation is stored in the P4 operand. *** The permutation is only valid until the next OP_Compare that has ** the OPFLAG_PERMUTE bit set in P5. Typically the OP_Permutation should ** occur immediately prior to the OP_Compare. *** The first integer in the P4 integer array is the length of the array ** and does not become part of the permutation.
69610. const char*
69611. The p2 value always comes from a prior OP_CreateTable opcode and ** that opcode will always set the p2 value to 2 or more or else fail. ** If there were a failure, the prepared statement would have halted ** before reaching this instruction.
69612. OUT: New sqlite3_vtab object
69613. ** Close the mutex on database connection db. *** Furthermore, if database connection db is a zombie (meaning that there ** has been a prior call to sqlite3_close(db) or sqlite3_close_v2(db)) and ** every sqlite3_stmt has now been finalized and every sqlite3_backup has ** finished, then free all resources.
69614. View definition
69615. Get a pointer the VDBE under construction, allocating a new VDBE if one ** does not already exist
69616. USING
69617. Pager object to allocate and return
69618. expr ::= ID|INDEXED
69619. **comment:** TODO: What if the cursor is in CURSOR_REQUIRESEEK but all table entries ** have been deleted? This API will need to change to return an error code ** as well as the boolean result value.
label: code-design
69620. Maximum docid to return
69621. **comment:** This block handles pages with two or fewer free blocks and nMaxFrag ** or fewer fragmented bytes. In this case it is faster to move the ** two (or one) blocks of cells using memmove() and add the required ** offsets to each pointer in the cell-pointer array than it is to ** reconstruct the entire page.
label: code-design
69622. Enforced by btreeInitPage()
69623. The sqlite3ResultSetOfSelect() is only used n contexts where lookaside ** is disabled
69624. Drop and reload the internal table schema.
69625. 1530
69626. ** The maximum allowed sector size. 64KiB. If the xSectorsize() method ** returns a value larger than this, then MAX_SECTOR_SIZE is used instead. ** This could conceivably cause corruption following a power failure on ** such a system. This is currently an undocumented limit.
69627. Need to call sqlite3ExprDelete(db, pExpr)
69628. The WhereOrSet to be updated

69629. STEP 1: ** Look for an entry of the correct size in either the small ** chunk table or in the large chunk hash table. This is ** successful most of the time (about 9 times out of 10).

69630. This loop runs once for each leaf in the tree of eType nodes.

69631. Mapping from table to index column numbers

69632. Index of first table to join in pSrc

69633. xSync

69634. Find the ORDER BY term that corresponds to the j-th column ** of the index and mark that ORDER BY term off

69635. ** A search constraint.

69636. Check that the values read from the page-size and sector-size fields ** are within range. To be 'in range', both values need to be a power ** of two greater than or equal to 512 or 32, and not greater than their ** respective compile time maximum limits.

69637. vtabargtoken ::= lp anylist RP

69638. The converted number is in buf[] and zero terminated. Output it. ** Note that the number is in the usual order, not reversed as with ** integer conversions.

69639. Number of active VDBEs that read or write

69640. No. of ORDER BY terms satisfied. -1 for unknown

69641. The results are stored in a sequence of registers ** starting at pDest->iSdst. Then the co-routine yields.

69642. Check to see if an existing table or index can be used to ** satisfy the query. This is preferable to generating a new ** ephemeral table.

69643. ** 2015-08-12 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This SQLite extension implements JSON functions. The interface is ** modeled after MySQL JSON functions: ** ** <https://dev.mysql.com/doc/refman/5.7/en/json.html> ** ** For the time being, all JSON is stored as pure text. (We might add ** a JSONB type in the future which stores a binary encoding of JSON in ** a BLOB, but there is no support for JSONB in the current implementation. ** This implementation parses JSON text at 250 MB/s, so it is hard to see ** how JSONB might improve on that.)

69644. ** Link the SubProgram object passed as the second argument into the linked ** list at Vdbe.pSubProgram. This list is used to delete all sub-program ** objects when the VM is no longer required.

69645. b: p1<<28 | p3<<14 | p5 (unmasked)

69646. 213

69647. Modified table

69648. Opcode: BitAnd P1 P2 P3 * * * Synopsis: r[P3]=r[P1]&r[P2] * * * Take the bit-wise AND of the values in register P1 and P2 and ** store the result in register P3. ** If either input is NULL, the result is NULL.

69649. 189

69650. ** Each file opened by an rbu VFS is represented by an instance of ** the following structure.

69651. Jump here to continue next step of UPDATE loop

69652. ** Each recursive mutex is an instance of the following structure.

69653. ***** End of sqlite3_vfs methods *****

69654. The conflict algorithm (OE_Abort, OE_Replace, etc.)

69655. Value of RBU_STATE_STAGE field

69656. ** Join all outstanding threads launched by SorterWrite() to create ** level-0 PMAs.

69657. text of name of TABLE

69658. Attempt the match

69659. A transient duplicate expression

69660. ** The Fts3Cursor.eSearch member is always set to one of the following. ** Actualy, Fts3Cursor.eSearch can be greater than or equal to ** FTS3_FULLTEXT_SEARCH. If so, then Fts3Cursor.eSearch - 2 is the index ** of the column to be searched. For example, in ** ** CREATE VIRTUAL TABLE ex1 USING fts3(a,b,c,d), ** SELECT docid FROM ex1 WHERE b MATCH 'one two three'; ** ** Because the LHS of the MATCH operator is 2nd column "b", ** Fts3Cursor.eSearch will be set to FTS3_FULLTEXT_SEARCH+1. (+0 for a, ** +1 for b, +2 for c, +3 for d.) If the LHS of MATCH were "ex1" ** indicating that all columns should be searched, ** then eSearch would be set to FTS3_FULLTEXT_SEARCH+4.

69661. The trigger step be fixed to one database

69662. SYNC_NORMAL or SYNC_FULL for checkpoint

69663. Offset from start of page to first cell pointer

69664. Pointer to vfs object

69665. ** Attempt to extract a value from expression pExpr using the methods ** as described for sqlite3Stat4ProbeSetValue() above. ** ** If successful, set *ppVal to point to a new value object and return ** SQLITE_OK. If no value can be extracted, but no other error occurs ** (e.g. OOM), return SQLITE_OK and set *ppVal to NULL. Or, if an error ** does occur, return an SQLite error code. The final value of *ppVal ** is undefined in this case.

69666. Next chunk on list of them all

69667. Name of FTS index

69668. The underlying shared-memory file

69669. IcuCursor.aOffset[]

69670. Hint to COMDB2

69671. ** Each auxiliary data pointer stored by a user defined function ** implementation calling sqlite3_set_auxdata() is stored in an instance ** of this structure. All such structures associated with a single VM ** are stored in a linked list headed at Vdbe.pAuxData. All are destroyed ** when the VM is halted (if not before).

69672. List of entries using pRight

69673. Number of tokens in extracted snippet

69674. If there are no active statements, clear the interrupt flag at this ** point.

69675. ***** Interface to code in fts5_index.c. fts5_index.c contains code ** to access the data stored in the %_data table.

69676. ** The code in this file is only compiled if: ** ** * The FTS3 module is being built as an extension ** (in which case SQLITE_CORE is not defined), or ** ** * The FTS3 module is being built into the core of ** SQLite (in which case SQLITE_ENABLE_FTS3 is defined).

69677. On the first call to sqlite3_step(), pSub will hold a NULL. It is ** initialized to a BLOB by the P4_SUBPROGRAM processing logic below

69678. Buffer to read data from

69679. Size of changeset blob in bytes

69680. OUT: Allocated Fts3SegReader

69681. ** Verify that at least N opcode slots are available in p without ** having to malloc for more space (except when compiled using ** SQLITE_TEST_REALLOC_STRESS). This interface is used during testing ** to verify that certain calls to sqlite3VdbeAddOpList() can never ** fail due to a OOM fault and hence that the return value from ** sqlite3VdbeAddOpList() will always be non-NULL.

69682. Opcode: InsertInt P1 P2 P3 P4 P5 ** Synopsis: intkey=P3 data=r[P2] * * * This works exactly like OP_Insert except that the key is the ** integer value P3, not the value of the integer stored in register P3.

69683. update conch with host and path (this will fail if other process ** has a shared lock already), if the host id matches, use the big ** stick.

69684. Cursor of the index itself

69685. Used to move bytes around within data[]

69686. ** 2007 August 14 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the C functions that implement mutexes for Win32.

69687. Buffer used to hold most recent term

69688. GOTO label for end of CASE stmt

69689. ** Return a pointer to a TriggerPrg object containing the sub-program for ** trigger pTrigger with default ON CONFLICT algorithm orconf. If no such ** TriggerPrg object exists, a new object is allocated and populated before ** being returned.

69690. ** Update the contents of the rbu_state table within the rbu database. The ** value stored in the RBU_STATE_STAGE column is eStage. All other values ** are determined by inspecting the rbu handle passed as the first argument.

69691. synopsis: key=r[P2]

69692. Only flags that can be set are SAVEPOISSION and AUXDELETE

69693. **comment:** The following describes the implementation of the various locks and ** lock transitions in terms of the POSIX advisory shared and exclusive ** lock primitives (called read-locks and write-locks below, to avoid ** confusion with SQLite lock names). The algorithms are complicated ** slightly in order to be compatible with Windows95 systems simultaneously ** accessing the same database file, in case that is ever required. *** Symbols defined in os.h indentify the 'pending byte' and the 'reserved ** byte', each single bytes at well known offsets, and the 'shared byte ** range', a range of 510 bytes at a well known offset. *** To obtain a SHARED lock, a read-lock is obtained on the 'pending ** byte'. If this is successful, 'shared byte range' is read-locked ** and the lock on the 'pending byte' released. (Legacy note: When ** SQLite was first developed, Windows95 systems were still very common, ** and Widnows95 lacks a shared-lock capability. So on Windows95, a ** single randomly selected by from the 'shared byte range' is locked. ** Windows95 is now pretty much extinct, but this work-around for the ** lack of shared-locks on Windows95 lives on, for backwards ** compatibility.) *** A process may only obtain a RESERVED lock after it has a SHARED lock. ** A RESERVED lock is implemented by grabbing a write-lock on the ** 'reserved byte'. *** A process may only obtain a PENDING lock after it has obtained a ** SHARED lock. A PENDING lock is implemented by obtaining a write-lock ** on the 'pending byte'. This ensures that no new SHARED locks can be ** obtained, but existing SHARED locks are allowed to persist. A process ** does not have to obtain a RESERVED lock on the way to a PENDING lock. ** This property is used by the algorithm for rolling back a journal file ** after a crash. *** An EXCLUSIVE lock, obtained after a PENDING lock is held, is ** implemented by obtaining a write-lock on the entire 'shared byte ** range'. Since all other locks require a read-lock on one of the bytes ** within this range, this ensures that no other locks are held on the ** database.

label: code-design

69694. No way this can happen

69695. Version 3.14.0 and later

69696. errcode set by sqlite3_result_error_code()

69697. If the EP_Reduced flag is set in the Expr.flags mask, then no ** space is allocated for the fields below this point. An attempt to ** access them will result in a segfault or malfunction. *****

69698. ** 2007 May 1 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains code used to implement incremental BLOB I/O.

69699. Register used with OP_Gosub

69700. ** The code above is the NFS lock implementation. The code is specific ** to Mac OSX and does not work on other unix platforms. No alternative ** is available. *** End of the NFS lock implementation ***

69701. ** Structure used internally by this VFS to record the state of an ** open shared memory connection. *** The following fields are initialized when this object is created and ** are read-only thereafter: ** ** winShm.pShmNode ** winShm.id ** ** All other fields are read/write. The winShm.pShmNode->mutex must be held ** while accessing any read/write fields.

69702. AND

69703. Use the original text of the column expression as its name

69704. ***** End of delete.c *****

69705. ** CAPI3REF: Find The Database Handle Of A Prepared Statement ** METHOD: sqlite3_stmt *** ^The sqlite3_db_handle interface returns the [database connection] handle ** to which a [prepared statement] belongs. ^The [database connection] ** returned by sqlite3_db_handle is the same [database connection] ** that was the first argument ** to the [sqlite3_prepare_v2() call (or its variants) that was used to ** create the statement in the first place.

69706. 15

69707. "[" or 0

69708. trigger_cmd_list ::= trigger_cmd_list trigger_cmd SEMI

69709. ** The following variable is (normally) set once and never changes ** thereafter. It records whether the operating system is Win9x ** or WinNT. ** ** 0: Operating system unknown. ** 1: Operating system is Win9x. ** 2: Operating system is WinNT. *** In order to facilitate testing on a WinNT system, the test fixture ** can manually set this value to 1 to emulate Win98 behavior.

69710. Keyinfo for scanned index

69711. Wildcard characters

69712. Process the LIMIT and OFFSET clauses, if they exist

69713. At this point, we know the candidate directory exists and should ** be used. However, we may need to convert the string containing ** its name into UTF-8 (i.e. if it is UTF-16 right now).

69714. ** CAPI3REF: Retrieve the mutex for a database connection ** METHOD: sqlite3 *** ^This interface returns a pointer to the [sqlite3_mutex] object that ** serializes access to the [database connection] given in the argument ** when the [threading mode] is Serialized. ** ^If the [threading mode] is Single-thread or Multi-thread then this ** routine returns a NULL pointer.

69715. Count down to first I/O error

69716. ** Access the static variable through a macro for SQLITE OMIT_WSD.

69717. **comment:** Check to see if we need to simulate an interrupt. This only happens ** if we have a special test build.

label: test

69718. Coding an expression that is part of an index where column names ** in the index refer to the table to which the index belongs

69719. Object to allocate and return

69720. Name of rbu db table (or null)

69721. Like mgt0 above except we are looking for a value of m which is ** exactly 1

69722. ***** Begin file main.c *****

69723. Registers for holding the start boundary

69724. ** This function queries the database for the names of the columns of table ** zThis, in schema zDb. It is expected that the table has nCol columns. If ** not, SQLITE_SCHEMA is returned and none of the output variables are ** populated. *** Otherwise, if they are not NULL, variable *pnCol is set to the number ** of columns in the database table and variable *pzTab is set to point to a ** nul-terminated copy of the table name. *pazCol (if not NULL) is set to ** point to an array of pointers to column names. And *pabPK (again, if not ** NULL) is set to point to an array of booleans - true if the corresponding ** column is part of the primary key. *** For example, if the table is declared as: *** CREATE TABLE tbl1(w, x, y, z, PRIMARY KEY(w, z)); *** Then the four output variables are populated as follows: *** *pnCol = 4 *** *pzTab = "tbl1" *** *pazCol = {"w", "x", "y", "z"} *** *pabPK = {1, 0, 0, 1} *** All returned buffers are part of the same single allocation, which must ** be freed using sqlite3_free() by the caller. If pazCol was not NULL, then ** pointer *pazCol should be freed to release all memory. Otherwise, pointer ** *pabPK. It is illegal for both pazCol and pabPK to be NULL.

69725. Full path to database file

69726. Jump from here if number of rows is zero

69727. grab an exclusive lock

69728. ** Return true if the given Btree is read-only.

69729. **comment:** Fails on a lookaside memory leak

label: code-design

69730. **comment:** ** Generate code to implement the "ALTER TABLE xxx RENAME TO yyy" ** command.

label: code-design

69731. ***** Begin file mutex.h *****

69732. Debug listings of VDBE programs

69733. ** Forward references to structures

69734. Est. number of rows where the key equals this sample

69735. The error context into which to write the error

69736. **comment:** ** If they are not already populated, populate the pIter->azTblCol[], ** pIter->abTblPk[], pIter->nTblCol and pIter->bRowid variables according to ** the table (not index) that the iterator currently points to. *** Return SQLITE_OK if successful, or an SQLite error code otherwise. If ** an error does occur, an

error code and error message are also left in ** the RBU handle.

label: code-design

69737. **comment:** These look like magic numbers. But they are stable, as they are part ** of the definition of the SQLite file format, which may not change.

label: code-design

69738. ** Name of the master database table. The master database table ** is a special table that holds the names and attributes of all ** user tables and indices.

69739. ** Write a 32-bit integer into the given file descriptor. Return SQLITE_OK ** on success or an error code is something goes wrong.

69740. **comment:** ***** CUSTOM TOKENIZERS *** Applications may also register custom tokenizer types. A tokenizer ** is registered by providing fts5 with a populated instance of the ** following structure. All structure methods must be defined, setting ** any member of the fts5_tokenizer struct to NULL leads to undefined ** behaviour. The structure methods are expected to function as follows: *** xCreate: ** This function is used to allocate and initialize a tokenizer instance. ** A tokenizer instance is required to actually tokenize text. *** The first argument passed to this function is a copy of the (void*) ** pointer provided by the application when the fts5_tokenizer object ** was registered with FTS5 (the third argument to xCreateTokenizer()). ** The second and third arguments are an array of nul-terminated strings ** containing the tokenizer arguments, if any, specified following the ** tokenizer name as part of the CREATE VIRTUAL TABLE statement used ** to create the FTS5 table. *** The final argument is an output variable. If successful, (*ppOut) ** should be set to point to the new tokenizer handle and SQLITE_OK ** returned. If an error occurs, some value other than SQLITE_OK should ** be returned. In this case, fts5 assumes that the final value of *ppOut ** is undefined. *** xDelete: ** This function is invoked to delete a tokenizer handle previously ** allocated using xCreate(). Fts5 guarantees that this function will ** be invoked exactly once for each successful call to xCreate(). *** xTokenize: ** This function is expected to tokenize the nText byte string indicated ** by argument pText. pText may or may not be null-terminated. The first ** argument passed to this function is a pointer to an Fts5Tokenizer object ** returned by an earlier call to xCreate(). *** The second argument indicates the reason that FTS5 is requesting ** tokenization of the supplied text. This is always one of the following ** four values: *** FTS5_TOKENIZE_DOCUMENT - A document is being inserted into ** or removed from the FTS table. The tokenizer is being invoked to ** determine the set of tokens to add to (or delete from) the ** FTS index. *** FTS5_TOKENIZE_QUERY - A MATCH query is being executed ** against the FTS index. The tokenizer is being called to tokenize ** a bareword or quoted string specified as part of the query. *** (FTS5_TOKENIZE_QUERY | FTS5_TOKENIZE_PREFIX) - Same as ** FTS5_TOKENIZE_QUERY, except that the bareword or quoted string is ** followed by a "*" character, indicating that the last token ** returned by the tokenizer will be treated as a token prefix. *** FTS5_TOKENIZE_AUX - The tokenizer is being invoked to ** satisfy an fts5_api.xTokenize() request made by an auxiliary ** function. Or an fts5_api.xColumnSize() request made by the same ** on a columsize=0 database. *** ** For each token in the input string, the supplied callback xToken() must ** be invoked. The first argument to it should be a copy of the pointer ** passed as the second argument to xTokenize(). The third and fourth ** arguments are a pointer to a buffer containing the token text, and the ** size of the token in bytes. The 4th and 5th arguments are the byte offsets ** of the first byte of and first byte immediately following the text from ** which the token is derived within the input. *** The second argument passed to the xToken() callback ("tflags") should ** normally be set to 0. The exception is if the tokenizer supports ** synonyms. In this case see the discussion below for details. *** FTS5 assumes the xToken() callback is invoked for each token in the ** order that they occur within the input text. *** If an xToken() callback returns any value other than SQLITE_OK, then ** the tokenization should be abandoned and the xTokenize() method should ** immediately return a copy of the xToken() return value. Or, if the ** input buffer is exhausted, xTokenize() should return SQLITE_OK. Finally, ** if an error occurs with the xTokenize() implementation itself, it ** may abandon the tokenization and return any error code other than ** SQLITE_OK or SQLITE_DONE. *** SYNONYM SUPPORT *** Custom tokenizers may also support synonyms. Consider a case in which a ** user wishes to query for a phrase such as "first place". Using the ** built-in tokenizers, the FTS5 query 'first + place' will match instances ** of "first place" within the document set, but not alternative forms ** such as "1st place". In some applications, it would be better to match ** all instances of "first place" or "1st place" regardless of which form ** the user specified in the MATCH query text. *** There are several ways to approach this in FTS5: *** By mapping all synonyms to a single token. In this case, the ** In the above example, this means that the tokenizer returns the ** same token for inputs "first" and "1st". Say that token is in ** fact "first", so that when the user inserts the document "I won ** 1st place" entries are added to the index for tokens "i", "won", ** "first" and "place". If the user then queries for '1st + place', ** the tokenizer substitutes "first" for "1st" and the query works ** as expected. *** By adding multiple synonyms for a single term to the FTS index. ** In this case, when tokenizing query text, the tokenizer may ** provide multiple synonyms for a single term within the document. ** FTS5 then queries the index for each synonym individually. For ** example, faced with the query: *** <codeblock> ** ... MATCH 'first place'</codeblock> ** ** the tokenizer offers both "1st" and "first" as synonyms for the ** first token in the MATCH query and FTS5 effectively runs a query ** similar to: *** <codeblock> ** ... MATCH ('first OR 1st') place</codeblock> ** ** except that, for the purposes of auxiliary functions, the query ** still appears to contain just two phrases - "(first OR 1st)" ** being treated as a single phrase. *** By adding multiple synonyms for a single term to the FTS index. ** Using this method, when tokenizing document text, the tokenizer ** provides multiple synonyms for each token. So that when a ** document such as "I won first place" is tokenized, entries are ** added to the FTS index for "i", "won", "first", "1st" and ** "place". ** ** This way, even if the tokenizer does not provide synonyms ** when tokenizing query text (it should not - to do would be ** inefficient), it doesn't matter if the user queries for ** 'first + place' or '1st + place', as there are entries in the ** FTS index corresponding to both forms of the first token. ** ** ** Whether it is parsing document or query text, any call to xToken that ** specifies a <i>tflags</i> argument with the FTS5_TOKEN_COLOCATED bit ** is considered to supply a synonym for the previous token. For example, ** when parsing the document "I won first place", a tokenizer that supports ** synonyms would call xToken() 5 times, as follows: *** <codeblock> ** xToken(pCtx, 0, "i", 1, 0, 1); ** xToken(pCtx, 0, "won", 3, 2, 5); ** xToken(pCtx, 0, "first", 5, 6, 11); ** xToken(pCtx, FTS5_TOKEN_COLOCATED, "1st", 3, 6, 11); ** xToken(pCtx, 0, "place", 5, 12, 17); ** </codeblock> ** ** It is an error to specify the FTS5_TOKEN_COLOCATED flag the first time ** xToken() is called. Multiple synonyms may be specified for a single token ** by making multiple calls to xToken(FTS5_TOKEN_COLOCATED) in sequence. ** There is no limit to the number of synonyms that may be provided for a ** single token. ** ** In many cases, method (1) above is the best approach. It does not add ** extra data to the FTS index or require FTS5 to query for multiple terms, ** so it is efficient in terms of disk space and query speed. However, it ** does not support prefix queries very well. If, as suggested above, the ** token "first" is substituted for "1st" by the tokenizer, then the query: *** <codeblock> ** ... MATCH '1s*'</codeblock> ** ** will not match documents that contain the token "1st" (as the tokenizer ** will probably not map "1s" to any prefix of "first"). *** ** For full prefix support, method (3) may be preferred. In this case, ** because the index contains entries for both "first" and "1st", prefix ** queries such as 'fi*' or '1s*' will match correctly. However, because ** extra entries are added to the FTS index, this method uses more space ** within the database. *** ** Method (2) offers a midpoint between (1) and (3). Using this method, ** a query such as '1s*' will match documents that contain the literal ** token "1st", but not "first" (assuming the tokenizer is not able to ** provide synonyms for prefixes). However, a non-prefix query like '1st' ** will match against "1st" and "first". This method does not require ** extra disk space, as no extra entries are added to the FTS index. ** On the other hand, it may require more CPU cycles to run MATCH queries, ** as separate queries of the FTS index are required for each synonym. ** ** When using methods (2) or (3), it is important that the tokenizer only ** provide synonyms when tokenizing document text (method (2)) or query ** text (method (3)), not both. Doing so will not cause any errors, but is ** inefficient.

label: code-design

69741. Query Planner Stability Guarantee

69742. The database connection

69743. IN: Pointer to the SQLiteThread structure

69744. PRAGMA index_info (legacy version)

69745. A counter

69746. ***** Include sqliteLimit.h in the middle of sqliteInt.h *****

69747. All winShm objects pointing to this

69748. ** Run internal checks to ensure that the FTS index (a) is internally ** consistent and (b) contains entries for which the XOR of the checksums ** as calculated by sqlite3Fts5IndexEntryCksum() is cksum. ** ** Return SQLITE_CORRUPT if any of the internal checks fail, or if the ** checksum does not match. Return SQLITE_OK if all checks pass without ** error, or some other SQLite error code if another error (e.g. OOM) ** occurs.

69749. !defined(SQLITE_CORE) || defined(SQLITE_ENABLE_STMTVTAB)

69750. pPager->szMmap = SQLITE_DEFAULT_MMAP_SIZE // will be set by btree.c

69751. Cursor used for virtual table scan

69752. Read the header. This might happen concurrently with a write to the ** same area of shared memory on a different CPU in a SMP, ** meaning it is possible that an inconsistent snapshot is read ** from the file. If this happens, return non-zero. ** ** There are two copies of the header at the beginning of the wal-index. ** When reading, read [0] first then [1]. Writes are in the reverse order. ** Memory barriers are used to prevent the compiler or the hardware from ** reordering the reads and writes.

69753. If the old.* record has not yet been loaded into memory, do so now.

69754. Argument to profile function

69755. **comment:** ** Column affinity types. ** ** These used to have mnemonic name like 'i' for SQLITE_AFF_INTEGER and ** 't' for SQLITE_AFF_TEXT. But we can save a little space and improve ** the speed a little by numbering the values consecutively. ** ** But rather than start with 0 or 1, we begin with 'A'. That way,

** when multiple affinity types are concatenated into a string and ** used as the P4 operand, they will be more readable. *** Note also that the numeric types are grouped together so that testing ** for a numeric type is a single comparison. And the BLOB type is first.

label: code-design

69756. Quote character (if any)

69757. ** 2007 May 7 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file defines various limits of what SQLite can process.

69758. Pointer to wal-index content in memory

69759. 4: (start_constraints && !startEq && !bRev)

69760. String to return via *pzWhere

69761. **comment:** Tokens are never deferred for FTS tables created using the content=xxx ** option. The reason being that it is not guaranteed that the content ** table actually contains the same data as the index. To prevent this from ** causing any problems, the deferred token optimization is completely ** disabled for content=xxx tables.

label: code-design

69762. Bytes of space required for argv[0]

69763. True if pStmt is a seek

69764. The node into which the cell is to be written

69765. **comment:** This term is already coded

label: code-design

69766. szLookaside, nLookaside

69767. 320

69768. Return the Rtree of a RtreeCursor

69769. Enable fts3_tokenizer(2)

69770. Previous value written into pgidx

69771. Copy as much data as is available in the buffer into the start of ** p->aAlloc[].

69772. 3: (Istart_constraints && startEq && bRev)

69773. Docid for current row.

69774. ** Read a 32-bit variable-length integer from memory starting at p[0]. ** Return the number of bytes read. The value is stored in *v. *** If the varint stored in p[0] is larger than can fit in a 32-bit unsigned ** integer, then set *v to 0xffffffff. *** A MACRO version, getVarint32, is provided which inlines the ** single-byte case. All code should use the MACRO version as ** this function assumes the single-byte case has already been handled.

69775. ** Destroy a tokenizer

69776. LANGUAGEID

69777. True if all rows are being deleted

69778. Initialize the next page.

69779. **comment:** ** CAPI3REF: Memory Allocation Routines *** An instance of this object defines the interface between SQLite ** and low-level memory allocation routines. *** This object is used in only one place in the SQLite interface. ** A pointer to an instance of this object is the argument to ** [sqlite3_config()] when the configuration option is ** [SQLITE_CONFIG_MALLOC] or [SQLITE_CONFIG_GETMALLOC]. ** By creating an instance of this object ** and passing it to [sqlite3_config()][SQLITE_CONFIG_MALLOC] ** during configuration, an application can specify an alternative ** memory allocation subsystem for SQLite to use for all of its ** dynamic memory needs. *** Note that SQLite comes with several [built-in memory allocators] ** that are perfectly adequate for the overwhelming majority of applications ** and that this object is only useful to a tiny minority of applications ** with specialized memory allocation requirements. This object is ** also used during testing of SQLite in order to specify an alternative ** memory allocator that simulates memory out-of-memory conditions in ** order to verify that SQLite recovers gracefully from such ** conditions. *** The xMalloc, xRealloc, and xFree methods must work like the ** malloc(), realloc() and free() functions from the standard C library. ** ^SQLite guarantees that the second argument to ** xRealloc is always a value returned by a prior call to xRoundup. *** xSize should return the allocated size of a memory allocation ** previously obtained from xMalloc or xRealloc. The allocated size ** is always at least as big as the requested size but may be larger. *** The xRoundup method returns what would be the allocated size of ** a memory allocation given a particular requested size. Most memory ** allocators round up memory allocations at least to the next multiple ** of 8. Some allocators round up to a larger multiple or to a power of 2. ** Every memory allocation request coming in through [sqlite3_malloc()] ** or [sqlite3_realloc()] first calls xRoundup. If xRoundup returns 0, ** that causes the corresponding memory allocation to fail. *** The xInit method initializes the memory allocator. For example, ** it might allocate any require mutexes or initialize internal data ** structures. The xShutdown method is invoked (indirectly) by ** [sqlite3_shutdown()] and should deallocate any resources acquired ** by xInit. The pAppData pointer is used as the only parameter to ** xInit and xShutdown. *** SQLite holds the [SQLITE_MUTEX_STATIC_MASTER] mutex when it invokes ** the xInit method, so the xInit method need not be threadsafe. The ** xShutdown method is only called from [sqlite3_shutdown()] so it does ** not need to be threadsafe either. For all other methods, SQLite ** holds the [SQLITE_MUTEX_STATIC_MEM] mutex as long as the ** [SQLITE_CONFIG_MEMSTATUS] configuration option is turned on (which ** it is by default) and so the methods are automatically serialized. ** However, if [SQLITE_CONFIG_MEMSTATUS] is disabled, then the other ** methods must be threadsafe or else make their own arrangements for ** serialization. *** SQLite will never invoke xInit() more than once without an intervening ** call to xShutdown().

label: code-design

69780. ** Create a new BTree table. Write into *piTable the page ** number for the root page of the new table. *** The type of type is determined by the flags parameter. Only the ** following values of flags are currently in use. Other values for ** flags might not work: ** BTREE_INTKEY|BTREE_LEAFDATA Used for SQL tables with rowid keys ** BTREE_ZERO DATA Used for SQL indices

69781. Fill in the pNew->x.pSelect or pNew->x.pList member.

69782. Expression list containing only pSelectRowid

69783. Iterator pointer

69784. Assume this many buckets in hash table

69785. If pDbPage was a btree-page, then it may have child pages and/or cells ** that point to overflow pages. The pointer map entries for all these ** pages need to be changed. *** If pDbPage is an overflow page, then the first 4 bytes may store a ** pointer to a subsequent overflow page. If this is the case, then ** the pointer map needs to be updated for the subsequent overflow page.

69786. The pager state may be changed from PAGER_ERROR to PAGER_OPEN here ** without clearing the error code. This is intentional - the error ** code is cleared and the cache reset in the block below.

69787. ** CAPI3REF: Load The Difference Between Tables Into A Session *** If it is not already attached to the session object passed as the first ** argument, this function attaches table zTbl in the same manner as the ** [sqlite3session_attach()] function. If zTbl does not exist, or if it ** does not have a primary key, this function is a no-op (but does not return ** an error). *** Argument zFromDb must be the name of a database ("main", "temp" etc.) ** attached to the same database handle as the session object that contains ** a table compatible with the table attached to the session by this function. ** A table is considered compatible if it: *** ** Has the same name, ** Has the same set of columns declared in the same order, and ** Has the same PRIMARY KEY definition. ** ** *** If the tables are not compatible, SQLITE_SCHEMA is returned. If the tables ** are compatible but do not have any PRIMARY KEY columns, it is not an error ** but no changes are added to the session object. As with other session ** APIs, tables without PRIMARY KEYS are simply ignored. *** This function adds a set of changes to the session object that could be ** used to update the table in database zFrom (call this the "from-table") ** so that its content is the same as the table attached to the session ** object (call this the "to-table"). Specifically: *** ** For each row (primary key) that exists in the to-table but not in ** the from-table, an INSERT record is added to the session object. ** ** For each row (primary key) that exists in both tables, but features ** different non-PK values in each, an UPDATE record is added to the ** session. ** ** *** To clarify, if this function is called and then a changeset constructed ** using [sqlite3session_changeset()], then after applying that changeset to ** database zFrom the contents of the two compatible tables would be ** identical. *** It an error if database zFrom does not exist or does not contain the ** required compatible table. *** If the operation successful, SQLITE_OK is returned. Otherwise, an SQLite ** error code. In this case, if argument pzErrMsg is not NULL, *pzErrMsg ** may be set to point to a buffer containing an English language error ** message. It is the responsibility of the caller to free this buffer using ** sqlite3_free().

69788. The methods above are in version 1 of the sqlite_module object. Those ** below are for version 2 and greater.

69789. Loop through all entries in the %_segdir table corresponding to ** segments in this index on levels greater than iAbsLevel. If there is ** at least one such segment, and it is possible to determine that all ** such segments are smaller than nLimit bytes in size, they will be ** promoted to level iAbsLevel.

69790. 240

69791. Input path
69792. What to do with the query results
69793. ** CAPI3REF: Database Connection Handle ** KEYWORDS: {database connection} {database connections} *** Each open SQLite database is represented by a pointer to an instance of ** the opaque structure named "sqlite3". It is useful to think of an sqlite3 ** pointer as an object. The [sqlite3_open()], [sqlite3_open16()], and ** [sqlite3_open_v2()] interfaces are its constructors, and [sqlite3_close()] ** and [sqlite3_close_v2()] are its destructors. There are many other ** interfaces (such as ** [sqlite3_prepare_v2()], [sqlite3_create_function()], and ** [sqlite3_busy_timeout()] to name but three) that are methods on an ** sqlite3 object.
69794. Start writing at this offset
69795. Make sure the number of columns in the source data matches the number ** of columns to be inserted into the table.
69796. SQLITE_OMIT_WAL
69797. ** Delete a changegroup object.
69798. **comment:** ** Measure the number of characters needed to output the given ** identifier. The number returned includes any quotes used ** but does not include the null terminator. ** ** The estimate is conservative. It might be larger than what is ** really needed.
label: code-design
69799. if we failed to get the lock then someone else must have it
69800. **comment:** Avoid unnecessary sorts by preserving the ROWSET_SORTED flags ** where possible
label: code-design
69801. The SET expressions are not actually used inside the WHERE loop. ** So reset the colUsed mask. Unless this is a virtual table. In that ** case, set all bits of the colUsed mask (to ensure that the virtual ** table implementation makes all columns available).
69802. Loop exits by "break"
69803. ** Invoke the given busy handler. ** ** This routine is called when an operation failed with a lock. ** If this routine returns non-zero, the lock is retried. If it ** returns 0, the operation aborts with an SQLITE_BUSY error.
69804. ** Move the content of pSrc into pDest
69805. Default mmap_size setting
69806. Number of columns beyond the key columns
69807. registers holding insert rowid
69808. If the btree is already in a write-transaction, or it ** is already in a read-transaction and a read-transaction ** is requested, this is a no-op.
69809. **comment:** ** PRAGMA optimize ** PRAGMA optimize(MASK) ** PRAGMA schema.optimize ** PRAGMA schema.optimize(MASK) *** Attempt to optimize the database. All schemas are optimized in the first ** two forms, and only the specified schema is optimized in the latter two. ** ** The details of optimizations performed by this pragma are expected ** to change and improve over time. Applications should anticipate that ** this pragma will perform new optimizations in future releases. ** ** The optional argument is a bitmask of optimizations to perform: ** ** 0x0001 Debugging mode. Do not actually perform any optimizations ** but instead return one line of text for each optimization ** that would have been done. Off by default. ** ** 0x0002 Run ANALYZE on tables that might benefit. On by default. ** See below for additional information. ** ** 0x0004 (Not yet implemented) Record usage and performance ** information from the current session in the ** database file so that it will be available to "optimize" ** pragmas run by future database connections. ** ** 0x0008 (Not yet implemented) Create indexes that might have ** been helpful to recent queries ** ** The default MASK is and always shall be 0xffffe. 0xffffe means perform all ** of the optimizations listed above except Debug Mode, including new ** optimizations that have not yet been invented. If new optimizations are ** ever added that should be off by default, those off-by-default ** optimizations will have bitmasks of 0x10000 or larger. ** ** DETERMINATION OF WHEN TO RUN ANALYZE ** ** In the current implementation, a table is analyzed if only if all of ** the following are true: ** ** (1) MASK bit 0x02 is set. ** ** (2) The query planner used sqlite_stat1-style statistics for one or ** more indexes of the table at some point during the lifetime of ** the current connection. ** ** (3) One or more indexes of the table are currently unanalyzed OR ** the number of rows in the table has increased by 25 times or more ** since the last time ANALYZE was run. ** ** The rules for when tables are analyzed are likely to change in ** future releases.
label: code-design
69810. Which nested loop of the FROM we are coding
69811. OFF
69812. ** Append buffer nData/pData to buffer pBuf. If an OOM error occurs, set ** the error code in p. If an error has already occurred when this function ** is called, it is a no-op.
69813. **comment:** Before setting pChild->pParent, test that we are not creating a ** loop of references (as we would if, say, pChild==pParent). We don't ** want to do this as it leads to a memory leak when trying to delete ** the referenced counted node structures.
label: code-design
69814. Do not bother with a bulk allocation if the cache size very small
69815. Append to this JSON string
69816. ** State information local to the memory allocation subsystem.
69817. Register in which to assemble record
69818. ** Three SQL functions - stat_init(), stat_push(), and stat_get() - ** share an instance of the following structure to hold their state ** information.
69819. OUT: True for an 'indirect' change
69820. 0x60 .. 0x6F
69821. ***** End control #defines *****
69822. EVIDENCE-OF: R-12970-05880 SQLite will not use more than one scratch ** buffers per thread. ** ** This can only be checked in single-threaded mode.
69823. Number of instance matches this row
69824. First of nMem value in the unpacked pKey
69825. trigger_cmd ::= select
69826. Next method
69827. Populate the Select.zSelName[] string that is used to help with ** query planner debugging, to differentiate between multiple Select ** objects in a complex query. ** ** If the SELECT keyword is immediately followed by a C-style comment ** then extract the first few alphanumeric characters from within that ** comment to be the zSelName value. Otherwise, the label is #N where ** is an integer that is incremented with each SELECT statement seen.
69828. same as TK_BITNOT, in1, out2
69829. RIGHT JOIN not (yet) supported
69830. ** Every page in the cache is controlled by an instance of the following ** structure.
69831. Pick rows in this db only
69832. Next trigger associated with the table
69833. pgnoRoot is the page that will be used for the root-page of ** the new table (assuming an error did not occur). But we were ** allocated pgnoMove. If required (i.e. if it was not allocated ** by extending the file), the current page at position pgnoMove ** is already journaled.
69834. VALUES
69835. Argument to the trace function
69836. Memory available to be allocated
69837. Upon reaching the end of input, call the parser two more times ** with tokens TK_SEMI and 0, in that order.
69838. ** Disable all error recovery processing in the parser push-down ** automaton.
69839. 307
69840. Create a new savepoint structure.
69841. Columns of %_content table
69842. Display all of the WhereLoop objects
69843. ** Initialize this module.
69844. case_exprlist ::= case_exprlist WHEN expr THEN expr
69845. ** Move to the next matching term/rowid. Used by the fts5vocab module.
69846. If there are two or more cursors on the same btree, then all such ** cursors *must* have the BTCP_Multiple flag set.
69847. Used in RBU vacuum mode only
69848. ** 2012 May 25 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

69849. on no off false yes true extra full

69850. ** Each token coming out of the lexer is an instance of ** this structure. Tokens are also used as part of an expression. ** ** Note if Token.z==0 then Token.dyn and Token.n are undefined and ** may contain random values. Do not make any assumptions about Token.dyn ** and Token.n when Token.z==0.

69851. SQLITE_STATUS_SCRATCH_OVERFLOW

69852. (326) anylist ::=

69853. expr ::= expr LT|GT|GE|LE expr

69854. ** CAPI3REF: Set A Busy Timeout ** METHOD: sqlite3 *** ^This routine sets a [sqlite3_busy_handler | busy handler] that sleeps ** for a specified amount of time when a table is locked. ^The handler ** will sleep multiple times until at least "ms" milliseconds of sleeping ** have accumulated. ^After at least "ms" milliseconds of sleeping, ** the handler returns 0 which causes [sqlite3_step0()] to return ** [SQLITE_BUSY]. ** ** ^Calling this routine with an argument less than or equal to zero ** turns off all busy handlers. ** ** ^There can only be a single busy handler for a particular ** [database connection] at any given moment. If another busy handler ** was defined (using [sqlite3_busy_handler()]) prior to calling ** this routine, that other busy handler is cleared.) ** ** See also: [PRAGMA busy_timeout]

69855. ** Run the parser and code generator recursively in order to generate ** code for the SQL statement given onto the end of the pParse context ** currently under construction. When the parser is run recursively ** this way, the final OP_Halt is not appended and other initialization ** and finalization steps are omitted because those are handled by the ** outermost parser. ** ** Not everything is nestable. This facility is designed to permit ** INSERT, UPDATE, and DELETE operations against SQLITE_MASTER. Use ** care if you decide to try to use this routine for some other purposes.

69856. Page number of the next source page to copy

69857. **comment:** Opcode: Clear P1 P2 P3 *** ** Delete all contents of the database table or index whose root page ** in the database file is given by P1. But, unlike Destroy, do not ** remove the table or index from the database file. ** ** The table being clear is in the main database file if P2==0. If ** P2==1 then the table to be clear is in the auxiliary database file ** that is used to store tables create using CREATE TEMPORARY TABLE. ** ** If the P3 value is non-zero, then the table referred to must be an ** intkey table (an SQL table, not an index). In this case the row change ** count is incremented by the number of rows in the table being cleared. ** If P3 is greater than zero, then the value stored in register P3 is ** also incremented by the number of rows in the table being cleared. ** ** See also: Destroy

label: code-design

69858. ** Remove a trigger from the hash tables of the sqlite* pointer.

69859. ** Create an "ascii" tokenizer.

69860. If it is a RELEASE, then destroy the savepoint being operated on ** too. If it is a ROLLBACK TO, then set the number of deferred ** constraint violations present in the database to the value stored ** when the savepoint was created.

69861. The first cursor that needs saving

69862. not the primary delete operation

69863. Name of current table

69864. Buffer containing token

69865. If the mmap() above failed, assume that all subsequent mmap() calls ** will probably fail too. Fall back to using xRead/xWrite exclusively ** in this case.

69866. ** Generate VDBE code that prepares for doing an operation that ** might change the database. ** ** This routine starts a new transaction if we are not already within ** a transaction. If we are already within a transaction, then a checkpoint ** is set if the setStatement parameter is true. A checkpoint should ** be set for operations that might fail (due to a constraint) part of ** the way through and which will need to undo some writes without having to ** rollback the whole transaction. For operations where all constraints ** can be checked before any changes are made to the database, it is never ** necessary to undo a write and the checkpoint should not be set.

69867. If this frame set completes the first transaction in the WAL and ** if PRAGMA journal_size_limit is set, then truncate the WAL to the ** journal size limit, if possible.

69868. OP_Delete does pre-update-hook only

69869. Cursor

69870. The WHERE clause for DELETE or UPDATE steps

69871. dot lock style uses the locking context to store the dot lock ** file path

69872. The incremental merge did not copy all the data from this ** segment to the upper level. The segment is modified in place ** so that it contains no keys smaller than zTerm/nTerm.

69873. If there is an ORDER BY clause, then create an ephemeral index to ** do the sorting. But this sorting ephemeral index might end up ** being unused if the data can be extracted in pre-sorted order. ** If that is the case, then the OP_OpenEphemeral instruction will be ** changed to an OP_Noop once we figure out that the sorting index is ** not needed. The sSort.addrSortIndex variable is used to facilitate ** that change.

69874. Do the b-tree integrity checks

69875. Program instructions for parent frame

69876. ** If argument pNear is NULL, then a new Fts5ExprNearset object is allocated ** and populated with pPhrase. Or, if pNear is not NULL, phrase pPhrase is ** appended to it and the results returned. ** ** If an OOM error occurs, both the pNear and pPhrase objects are freed and ** NULL returned.

69877. First argument to xCallback()

69878. cmd ::= ANALYZE nm dbnm

69879. Number of terms

69880. NO

69881. Transient storage for serial_type in OP_MakeRecord

69882. SQLite function call context

69883. Mem.i contains count of 0s appended to blob

69884. The new opcode

69885. SQLITE_AUTH

69886. 122

69887. Use unixClose to clean up the resources added in fillInUnixFile ** and clear all the structure's references. Specifically, ** pFile->pMethods will be NULL so sqlite3OsClose will be a no-op

69888. ** The first parameter (pDef) is a function implementation. The ** second parameter (pExpr) is the first argument to this function. ** If pExpr is a column in a virtual table, then let the virtual ** table implementation have an opportunity to overload the function. ** ** This routine is used to allow virtual table implementations to ** overload MATCH, LIKE, GLOB, and REGEXP operators. ** ** Return either the pDef argument (indicating no change) or a ** new FuncDef structure that is marked as ephemeral using the ** SQLITE_FUNC_EPHEM flag.

69889. Generate column names for this SELECT statement

69890. xBegin

69891. ** This function is called when initializing an incremental-merge operation. ** It checks if the existing segment with index value iIdx at absolute level ** (iAbsLevel+1) can be appended to by the incremental merge. If it can, the ** merge-writer object *pWriter is initialized to write to it. ** ** An existing segment can be appended to by an incremental merge if: ** ** * It was initially created as an appendable segment (with all required ** space pre-allocated), and ** ** * The first key read from the input (arguments zKey and nKey) is ** greater than the largest key currently stored in the potential ** output segment.

69892. Input buffer

69893. Number of times this instruction was executed

69894. Index of root page

69895. Set nNewSize to the size allocated for the structure pointed to ** by pNew. This is either EXPR_FULLSCREEN, EXPR_REDUCEDSIZE or ** EXPR_TOKENONLYSIZE. nToken is set to the number of bytes consumed ** by the copy of the p->u.zToken string (if any).

69896. WAL connection

69897. 0x0e

69898. ** This routine sets up a SELECT statement for processing. The ** following is accomplished: ** ** * VDBE Cursor numbers are assigned to all FROM-clause terms. ** * Ephemeral Table objects are created for all FROM-clause subqueries. ** * ON and USING clauses are shifted into WHERE statements ** * Wildcards ** and "TABLE.**" in result sets are expanded. ** * Identifiers in expression are matched to tables. ** ** This routine acts recursively on all subqueries within the SELECT.

69899. Number of records written to sub-journal

69900. Previous with the same zTo
69901. The P2 table cursor (OP_DeferredSeek only)
69902. For looping over compound SELECTs in pSubq
69903. Cursor to read old values from
69904. Current number of savepoints
69905. ** Report the wrong number of arguments for json_insert(), json_replace() ** or json_set().
69906. occur in pairs
69907. ** The Vdbe.aColName array contains 5n Mem structures, where n is the ** number of columns of data returned by the statement.
69908. Connection to watch for unlock
69909. Delete the transient table structure associated with the ** subquery
69910. Free this when deleting the vdbe
69911. Do some checking to help insure the file we opened really is ** a valid database file.
69912. using_opt
69913. RHS is a blob
69914. Incremented with each busy call
69915. ** Allowed values for the flags parameter to sqlite3PagerOpen(). *** NOTE: These values must match the corresponding BTREE_ values in btree.h.
69916. Check if pExpr is a sub-select. If so, consider it variable.
69917. Opcode: String8 * P2 * P4 *** Synopsis: r[P2]='P4' *** P4 points to a nul terminated UTF-8 string. This opcode is transformed ** into a String opcode before it is executed for the first time. During ** this transformation, the length of string P4 is computed and stored ** as the P1 parameter.
69918. ***** Interface to code in fts5_varint.c.
69919. Set to true when thread finishes
69920. ** Collation sequence destructor function. The pCtx argument points to ** a UCollator structure previously allocated using ucol_open().
69921. ** Open a new Fts5Index handle. If the bCreate argument is true, create ** and initialize the underlying tables *** If successful, set *pp to point to the new object and return SQLITE_OK. ** Otherwise, set *pp to NULL and return an SQLite error code.
69922. Buffer to load terms into
69923. Set the usable flag on the subset of constraints identified by ** arguments mUsable and mExclude.
69924. Seek cursor iCur to the row to delete. If this row no longer exists ** (this can happen if a trigger program has already deleted it), do ** not attempt to delete it or fire any DELETE triggers.
69925. 153
69926. Construct the Index object to describe this index
69927. True after i'th new page is populated
69928. Open the connection to the log file. If this operation fails, ** (e.g. due to malloc() failure), return an error code.
69929. Common features
69930. The FROM clause term to search
69931. Size of term prefix in bytes
69932. Copy original pages out of the journal and back into the ** database file and/or page cache.
69933. ** Return the 'affinity' of the expression pExpr if any. *** If pExpr is a column, a reference to a column via an 'AS' alias, ** or a sub-select with a column as the return value, then the ** affinity of that column is returned. Otherwise, 0x00 is returned, ** indicating no affinity for the expression. *** i.e. the WHERE clause expressions in the following statements all ** have an affinity: *** CREATE TABLE t1(a); ** SELECT * FROM t1 WHERE a; ** SELECT a AS b FROM t1 WHERE b; ** SELECT * FROM t1 WHERE (select a from t1);
69934. A correlated subquery has been seen
69935. Allocated size of buffer z in bytes
69936. ***** End of utf.c *****
69937. **comment:** Unused opcode memory
label: code-design
69938. ** The DJGPP compiler environment looks mostly like Unix, but it ** lacks the fcntl() system call. So redefine fcntl() to be something ** that always succeeds. This means that locking does not occur under ** DJGPP. But it is DOS - what did you expect?
69939. Session object pTab is attached to
69940. setlist ::= LP idlist RP EQ expr
69941. A completely parsed JSON string
69942. ** Return the depth of a tree comprising nPMA PMAs, assuming a fanout of ** SORTER_MAX_MERGE_COUNT. The returned value does not include leaf nodes. *** i.e. *** nPMA<=16 -> TreeDepth() == 0 ** nPMA<=256 -> TreeDepth() == 1 ** nPMA<=65536 -> TreeDepth() == 2
69943. ** Kinds of hints that can be passed into the sqlite3BtreeCursorHint() ** interface. *** BTREE_HINT_RANGE (arguments: Expr*, Mem*) *** The first argument is an Expr* (which is guaranteed to be constant for ** the lifetime of the cursor) that defines constraints on which rows ** might be fetched with this cursor. The Expr* tree may contain ** TK_REGISTER nodes that refer to values stored in the array of registers ** passed as the second parameter. In other words, if Expr.op==TK_REGISTER ** then the value of the node is the value in Mem[pExpr.iTable]. Any ** TK_COLUMN node in the expression tree refers to the Expr.iColumn-th ** column of the b-tree of the cursor. The Expr tree will not contain ** any function calls nor subqueries nor references to b-trees other than ** the cursor being hinted. *** The design of the _RANGE hint is aid b-tree implementations that try ** to prefetch content from remote machines - to provide those ** implementations with limits on what needs to be prefetched and thereby ** reduce network bandwidth. *** Note that BTREE_HINT_FLAGS with BTREE_BULKLOAD is the only hint used by ** standard SQLite. The other hints are provided for extentions that use ** the SQLite parser and code generator but substitute their own storage ** engine.
69944. pSlot and pCArray->apCell[i] will never overlap on a well-formed ** database. But they might for a corrupt database. Hence use memmove() ** since memcpy() sends SIGABORT with overlapping buffers on OpenBSD
69945. Format string for SQL
69946. resolvetype ::= REPLACE
69947. Minimum file format for writable database files
69948. LOCATE_VIEW or LOCATE_NOERR
69949. Add the table to the in-memory representation of the database.
69950. OUT: First token of proposed snippet
69951. fts5YYNOERRORRECOVERY
69952. Return the sqlite3_file object for the WAL file
69953. Precomputed device characteristics
69954. Object to pass to xBestIndex()
69955. This is what we do if the grammar does not define ERROR: *** * Report an error message, and throw away the input token. *** * If the input token is \$, then fail the parse. *** As before, subsequent error messages are suppressed until ** three input tokens have been successfully shifted.
69956. ** Convert an ANSI string to Microsoft Unicode, using the ANSI or OEM ** code page. *** Space to hold the returned string is obtained from sqlite3_malloc().
69957. Mask of the bit in sqlite3.flags to set/clear
69958. Comparison operator
69959. ** The second parameter to sqlite3BtreeGetMeta or sqlite3BtreeUpdateMeta ** should be one of the following values. The integer values are assigned ** to constants so that the offset of the corresponding field in an ** SQLite database header may be found using the following formula: *** offset = 36 + (idx * 4) **
** For example, the free-page-count field is located at byte offset 36 of ** the database file header. The incr-vacuum-flag field is located at ** byte offset 64 (== 36+4*7). *** The BTREE_DATA_VERSION value is not really a value stored in the header. ** It is a read-only number computed by the pager. But we merge it with ** the header value access routines since its access pattern is the same. ** Call it a "virtual meta value".
69960. Run the program
69961. If this is an INSERT into a table b-tree and the table has an ** explicit INTEGER PRIMARY KEY, check that this is not an attempt ** to write a NULL into the IPK column. That is not permitted.
69962. Current tree root
69963. ** Return true if the p->aiException[] array contains the value iCode.

69964. Create a record from the argument register contents and insert it into ** the ephemeral table.
69965. List of dirty pages in LRU order
69966. ** CAPI3REF: Custom Page Cache Object ** ** The sqlite3_pcache type is opaque. It is implemented by ** the pluggable module. The SQLite core has no knowledge of ** its size or internal structure and never deals with the ** sqlite3_pcache object except by holding and passing pointers ** to the object. *** See [sqlite3_pcache_methods2] for additional information.
69967. ** This function is called while stepping or preparing a statement ** associated with connection db. The operation will return SQLITE_LOCKED ** to the user because it requires a lock that will not be available ** until connection pBlocker concludes its current transaction.
69968. ** The json_test1(JSON) function return true (1) if the input is JSON ** text generated by another json function. It returns (0) if the input ** is not known to be JSON.
69969. Do not write content to disk
69970. old.* record for second change
69971. Saved syscall error number
69972. Index of the column to extract
69973. 148
69974. Try to ORDER BY the result set to make distinct processing easier
69975. ** Close an open wal-index.
69976. Array of arguments
69977. Underlying VFS
69978. Saved value of p->iOffset
69979. Pseudo-random number used for sampling
69980. Value to return from this function
69981. True to run the ORDER BY search loop
69982. EXCLUSIVE
69983. Delete the index records
69984. Allocated size of aBuffer[] in bytes
69985. Number of slots in apHash[]
69986. "PRAGMA encoding"
69987. The database structure
69988. Tokenizer module
69989. The SQLITE_EXTRA_DURABLE compile-time option used to set the default ** synchronous setting to EXTRA. It is no longer supported.
69990. The expression mask set
69991. Year, month, and day
69992. Fire BEFORE or INSTEAD OF triggers
69993. xEof - check for end of scan
69994. xRowid - read data
69995. jump, in1
69996. ** Grow the db->aVTrans[] array so that there is room for at least one ** more v-table. Return SQLITE_NOMEM if a malloc fails, or SQLITE_OK otherwise.
69997. ** get2byteAligned(), unlike get2byte(), requires that its argument point to a ** two-byte aligned address. get2bytea() is only used for accessing the ** cell addresses in a btree header.
69998. The "wsdStat" macro will resolve to the status information ** state vector. If writable static data is unsupported on the target, ** we have to locate the state vector at run-time. In the more common ** case where writable static data is supported, wsdStat can refer directly ** to the "sqlite3Stat" state vector declared above.
69999. 830
70000. ** Execute zSql on database db. *** If zSql returns rows, then each row will have exactly one ** column. (This will only happen if zSql begins with "SELECT".) ** Take each row of result and call execSql() again recursively. *** The execSqlF() routine does the same thing, except it accepts ** a format string as its third argument
70001. The flattened query is distinct if either the inner or the ** outer query is distinct.
70002. Pager.pageSize bytes of space for tmp use
70003. Identify the output register
70004. Expression for MATCH queries
70005. ** The maximum number of attached databases. This must be between 0 ** and 125. The upper bound of 125 is because the attached databases are ** counted using a signed 8-bit integer which has a maximum value of 127 ** and we have to allow 2 extra counts for the "main" and "temp" databases.
70006. Used to iterate through pTrigger list
70007. One or more of FLAG_ constants below
70008. ** Free up as much memory as possible from the page cache.
70009. 132
70010. ** Flush the contents of in-memory hash table iHash to a new level-0 ** segment on disk. Also update the corresponding structure record. *** If an error occurs, set the Fts5Index.rc error code. If an error has ** already occurred, this function is a no-op.
70011. ** END OF CUSTOM TOKENIZERS *****
70012. Set up a new SrcList in pOrTab containing the table being scanned ** by this loop in the a[0] slot and all notReady tables in a[1..] slots. ** This becomes the SrcList in the recursive call to sqlite3WhereBegin().
70013. Set len to the maximum number of bytes required in the output buffer.
70014. The file descriptor for pFile
70015. If there were no rowids on the leaf page either and the doclist-index ** has already been started, append an 0x00 byte to it.
70016. Mask of TF_* values
70017. Opcode: IFNotZero P1 P2 * * * * Synopsis: if r[P1]!=0 then r[P1]--, goto P2 ** ** Register P1 must contain an integer. If the content of register P1 is ** initially greater than zero, then decrement the value in register P1. ** If it is non-zero (negative or positive) and then also jump to P2. ** If register P1 is initially zero, leave it unchanged and fall through.
70018. Value returned by sqlite3_total_changes()
70019. ** Turn bulk memory into a RowSet object. N bytes of memory ** are available at pSpace. The db pointer is used as a memory context ** for any subsequent allocations that need to occur. ** Return a pointer to the new RowSet object. *** It must be the case that N is sufficient to make a Rowset. If not ** an assertion fault occurs. *** If N is larger than the minimum, use the surplus as an initial ** allocation of entries available to be filled.
70020. Index of first token in snippet
70021. Release the reader lock held while backfilling
70022. Number of bits that are set - only valid for aHash ** element. Max is BITVEC_NINT. For BITVEC_SZ of 512, ** this would be 125.
70023. CAST => ID
70024. text encoding
70025. sqlite3WhereGetMask(&pWInfo->sMaskSet, iCur);
70026. number of named columns in virtual table
70027. ** Retrieve a page from the pager cache. If the requested page is not ** already in the pager cache return NULL. Initialize the MemPage.pBt and ** MemPage.aData elements if needed.
70028. Forward references to VFS helper methods used for temporary files
70029. ** This routine restores a cursor back to its original position after it ** has been moved by some outside activity (such as a btree rebalance or ** a row having been deleted out from under the cursor). *** On success, the *pDifferentRow parameter is false if the cursor is left ** pointing at exactly the same row. *pDifferntRow is the row the cursor ** was pointing to has been deleted, forcing the cursor to point to some ** nearby row. *** This routine should only be called for a cursor that just returned ** TRUE from sqlite3BtreeCursorHasMoved().
70030. expr ::= expr EQ|NE expr
70031. The data for the record immediately follows this header
70032. Calling function guarantees this much

70033. Empty output segment. This is a no-op.
70034. ** Sync an rbuVfs-file.
70035. b
70036. Starter space for aArg[]. No malloc required
70037. Allocate space for the WalIterator object.
70038. refarg ::= ON DELETE refact
70039. INNER
70040. typename ::= typename ID|STRING
70041. Zero or more SQLITE_PREPARE_* flags
70042. To avoid deadlock, first release all locks with a larger ** BtShared address. Then acquire our lock. Then reacquire ** the other BtShared locks that we used to hold in ascending ** order.
70043. testcase(IS_BIG_INT(iOffset)); // requires a 4GiB WAL
70044. In test mode, increase the size of this structure a bit so that ** it is larger than the struct CrashFile defined in test6.c.
70045. IN/OUT: Readr offset in pTask->file
70046. ** Close a connection to shared-memory. Delete the underlying ** storage if deleteFlag is true.
70047. now generate the new WHERE rowid IN clause for the DELETE/UPDATE
70048. ** CAPI3REF: Virtual Table Configuration Options *** These macros define the various options to the ** [sqlite3_vtab_config0] interface that [virtual table] implementations ** can use to customize and optimize their behavior. *** <dl> ** <dt>SQLITE_VTAB_CONSTRAINT_SUPPORT ** <dd>Calls of the form ** [sqlite3_vtab_config](db,SQLITE_VTAB_CONSTRAINT_SUPPORT,X) are supported, ** where X is an integer. If X is zero, then the [virtual table] whose ** [xCreate] or [xConnect] method invoked [sqlite3_vtab_config()] does not ** support constraints. In this configuration (which is the default) if ** a call to the [xUpdate] method returns [SQLITE_CONSTRAINT], then the entire ** statement is rolled back as if [ON CONFLICT | OR ABORT] had been ** specified as part of the users SQL statement, regardless of the actual ** ON CONFLICT mode specified. *** If X is non-zero, then the virtual table implementation guarantees ** that if [xUpdate] returns [SQLITE_CONSTRAINT], it will do so before ** any modifications to internal or persistent data structures have been made. ** If the [ON CONFLICT] mode is ABORT, FAIL, IGNORE or ROLLBACK, SQLite ** is able to roll back a statement or database transaction, and abandon ** or continue processing the current SQL statement as appropriate. ** If the ON CONFLICT mode is REPLACE and the [xUpdate] method returns ** [SQLITE_CONSTRAINT], SQLite handles this as if the ON CONFLICT mode ** had been ABORT. *** Virtual table implementations that are required to handle OR REPLACE ** must do so within the [xUpdate] method. If a call to the ** [sqlite3_vtab_on_conflict()] function indicates that the current ON ** CONFLICT policy is REPLACE, the virtual table implementation should ** silently replace the appropriate rows within the xUpdate callback and ** return SQLITE_OK. Or, if this is not possible, it may return ** SQLITE_CONSTRAINT, in which case SQLite falls back to OR ABORT ** constraint handling. ** </dl>
70049. SQLITE_STMTSTATUS_VM_STEP
70050. Set the WHERE_ONEROW flag if the xBestIndex() method indicated ** that the scan will visit at most one row. Clear it otherwise.
70051. ** Code an OP_Halt due to non-unique rowid.
70052. The BtShared object that owns pPage
70053. Make sure the ordering..
70054. ** The sqlite3KeywordCode function looks up an identifier to determine if ** it is a keyword. If it is a keyword, the token code of that keyword is ** returned. If the input is not a keyword, TK_ID is returned. *** The implementation of this routine was generated by a program, ** mkkeywordhash.c, located in the tool subdirectory of the distribution. ** The output of the mkkeywordhash.c program is written into a file ** named keywordhash.h and then included into this source file by ** #include below.
70055. Called only by OP_MaxPgcnt
70056. xSetSystemCall
70057. Address of the OP_Jump opcode
70058. ** Stub routines for all mutex methods. *** This routines provide no mutual exclusion or error checking.
70059. ** The sqlite3ValueBytes() routine returns the number of bytes in the ** sqlite3_value object assuming that it uses the encoding "enc". ** The valueBytes() routine is a helper function.
70060. ** Assign VdbeCursor index numbers to all tables in a SrcList
70061. SQLITE_ENABLE_DBSTAT_VTAB
70062. Busy callback
70063. List of global functions to be inserted
70064. ***** End of where.c *****
70065. **comment:** Processing for aggregates with GROUP BY is very different and ** much more complex than aggregates without a GROUP BY.
 label: code-design
70066. ** Return TRUE if the pager is in a state where it is OK to change the ** journalmode. Journalmode changes can only happen when the database ** is unmodified.
70067. ** Link the chunk at mem5.aPool[i] so that is on the iLogsize ** free list.
70068. ** This function is the implementation of the xUpdate callback used by ** FTS3 virtual tables. It is invoked by SQLite each time a row is to be ** inserted, updated or deleted.
70069. Destructor for the aux data
70070. Skip past any whitespace
70071. **comment:** First register in temporary register block
 label: code-design
70072. One usually wants to use hw.acctivecpu for MT decisions, but not here
70073. Add this many bytes to each in-memory page
70074. If the first page of the wal-index has been mapped, try to read the ** wal-index header immediately, without holding any lock. This usually ** works, but may fail if the wal-index header is corrupt or currently ** being modified by another thread or process.
70075. ** These macros define the location of the pointer-map entry for a ** database page. The first argument to each is the number of usable ** bytes on each page of the database (often 1024). The second is the ** page number to look up in the pointer map. *** PTRMAP_PAGENO returns the database page number of the pointer-map ** page that stores the required pointer. PTRMAP_PTROFFSET returns ** the offset of the requested map entry. *** If the pgno argument passed to PTRMAP_PAGENO is a pointer-map page, ** then pgno is returned. So (pgno==PTRMAP_PAGENO(pgsz, pgno)) can be ** used to test if pgno is a pointer-map page. PTRMAP_ISPAGE implements ** this test.
70076. **comment:** The following loop copies up to p->nBuffer bytes per iteration into ** the p->aAlloc[] buffer.
 label: code-design
70077. Index of the OR-term to be analyzed
70078. ** 2004 May 22 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
 ***** This file contains macros and a little bit of code that is common to ** all of the platform-specific files (os_*.c) and is #included into those ** files. *** This file should be #included by the os_*.c files only. It is not a ** general purpose header file.
70079. Code the LHS, the <expr> from "<expr> IN (...)" . If the LHS is a ** vector, then it is stored in an array of nVector registers starting ** at r1. ***
 sqlite3FindInIndex() might have reordered the fields of the LHS vector ** so that the fields are in the same order as an existing index. The ** aiMap[] array contains a mapping from the original LHS field order to ** the field order that matches the RHS index.
70080. Value of db->nChange
70081. ** Clear (destroy) the BtShared.pHasContent bitvec. This should be ** invoked at the conclusion of each write-transaction.
70082. The VDBE already created by calling function
70083. Number of result columns
70084. The database file is locked
70085. Handle extra flags for this index, if not NULL
70086. SQLITE OMIT_PRAGMA
70087. If the existing change is considered "indirect", but this current ** change is "direct", mark the change object as direct.

70088. No-op stubs to use when memory-mapped I/O is disabled
70089. Unix cannot, but some systems may return SQLITE_FULL from here. This ** line is to test that doing so does not cause any problems.
70090. Add an entry to each output position list
70091. ** Merge the two sorted lists p1 and p2 into a single list.
70092. EVIDENCE-OF: R-12793-43283 Every value in SQLite has one of five ** fundamental datatypes: 64-bit signed integer 64-bit IEEE floating ** point number string BLOB NULL
70093. PRIMARY KEY array
70094. ** Convert a UTF-8 string to a multi-byte character string. ** ** Space to hold the returned string is obtained from sqlite3_malloc().
70095. expr ::= nm DOT nm DOT nm
70096. Root of the new tree
70097. If control gets to this point, then actually go ahead and make ** operating system calls for the specified lock.
70098. Read in the WAL header.
70099. ** Return one of the WHERE_DISTINCT_xxxxx values to indicate how this ** WHERE clause returns outputs for DISTINCT processing.
70100. If the destination is an EXISTS(...) expression, the actual ** values returned by the SELECT are not required.
70101. xLock
70102. If bSkip is true, then the caller has already determined that the first ** two elements in the keys are equal. Fix the various stack variables so ** that this routine begins comparing at the second field.
70103. ** Attempt to move the phrase iterator to point to the next matching docid. ** If an error occurs, return an SQLite error code. Otherwise, return ** SQLITE_OK.
** If there is no "next" entry and no error occurs, then *pbEof is set to ** 1 before returning. Otherwise, if no error occurs and the iterator is ** successfully advanced, *pbEof is set to 0.
70104. Assume 32-bit assignment is atomic
70105. ** Each thread may only have a single outstanding allocation from ** xScratchMalloc(). We verify this constraint in the single-threaded ** case by setting scratchAllocOut to 1 when an allocation ** is outstanding clearing it when the allocation is freed.
70106. 327
70107. xRelease
70108. ** Search the wal file for page pgno. If found, set *piRead to the frame that ** contains the page. Otherwise, if pgno is not in the wal file, set *piRead ** to zero.
** ** Return SQLITE_OK if successful, or an error code if an error occurs. If an ** error does occur, the final value of *piRead is undefined.
70109. xCurrentTime
70110. Column number within the source table
70111. "ORDER" or "GROUP"
70112. ** 2001-09-15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This header file defines the interface that the SQLite library **
presents to client programs. If a C-function, structure, datatype, ** or constant definition does not appear in this file, then it is ** not a published API of SQLite, is
subject to change without ** notice, and should not be referenced by programs that use SQLite. ** ** Some of the definitions that are in this file are marked as **
"experimental". Experimental interfaces are normally new ** features recently added to SQLite. We do not anticipate changes ** to experimental interfaces but
reserve the right to make minor changes ** if experience from use "in the wild" suggest such changes are prudent. ** ** The official C-language API
documentation for SQLite is derived ** from comments in this file. This file is the authoritative source ** on how SQLite interfaces are supposed to operate. ** **
The name of this file under configuration management is "sqlite.h.in". ** The makefile makes some minor changes to this file (such as inserting ** the version
number) and changes its name to "sqlite3.h" as ** part of the build process.
70113. dbnm ::=
70114. 2x
70115. from ::=
70116. #ifndef SQLITE_UNTESTABLE
70117. ** This function invokes either the xRollback or xCommit method ** of each of the virtual tables in the sqlite3.aVTrans array. The method ** called is identified
by the second argument, "offset", which is ** the offset of the method to call in the sqlite3_module structure. ** ** The array is cleared after invoking the
callbacks.
70118. ** Query the database. But instead of invoking a callback for each row, ** malloc() for space to hold the result and return the entire results ** at the conclusion of
the call. ** ** The result that is written to ***pazResult is held in memory obtained ** from malloc(). But the caller cannot free this memory directly. ** Instead,
the entire table should be passed to sqlite3_free_table() when ** the calling procedure is finished using it.
70119. ** The following code executes when a syntax error first occurs.
70120. ** Close a file that uses proxy locks.
70121. ** Btree.inTrans may take one of the following values. ** ** If the shared-data extension is enabled, there may be multiple users ** of the Btree structure. At most
one of these may open a write transaction, ** but any number may have active read transactions.
70122. OUT: Ending offset of token
70123. Phrase to append to
70124. Number of columns in the recursive table
70125. NULLs can be safely trimmed from the end of the record, as long as ** as the schema format is 2 or more and none of the omitted columns ** have a non-NUL
default value. Also, the record must be left with ** at least one field. If P5>0 then it will be one more than the ** index of the right-most column with a non-NUL
default value
70126. Return code from service routines
70127. A single term of the WHERE clause
70128. Initial static space
70129. OUTPUT: Declared data type
70130. 280
70131. **comment:** If using the content=xxx option, assume the table is never empty
label: code-design
70132. ** Formulate and prepare an INSERT statement to add a record to table zTab. ** For example: *** INSERT INTO main."zTab" VALUES(?1, ?2, ?3 ...); ** ** If
successful, SQLITE_OK is returned and SessionApplyCtx.pInsert is left ** pointing to the prepared version of the SQL statement.
70133. Array of primary key flags
70134. Add an entry in sqlite_master for this index
70135. False to omit journal
70136. Restrict matches to this column
70137. Def. collation for column
70138. For tracing reduce actions, the names of all rules are required.
70139. If there are no more entries in the doclist, set pOffsetList to ** NULL. Otherwise, set Fts3SegReader.iDocid to the next docid and ** Fts3SegReader.pOffsetList to
point to the next offset list before ** returning.
70140. Used to divy up the pSpace memory
70141. 30
70142. Text of the token. Not NULL-terminated!
70143. Set the cursor upper and lower rowid limits. Only some strategies ** actually use them. This is ok, as the xBestIndex() method leaves the **
sqlite3_index_constraint.omit flag clear for range constraints ** on the rowid field.
70144. UTF-16 Big-endian -> UTF-8
70145. 118
70146. ifndef _SQLITE3RTREE_H_
70147. Return the value of a column
70148. ** Allocate a KeyInfo object sufficient for an index of N key columns and ** X extra columns.
70149. (278) trans_opt ::=

70150. Compare this many columns
70151. Counter for apVal[]
70152. **comment:** ** Make sure the page is marked as clean. If it isn't clean already, ** make it so.
 label: code-design
70153. OUT: Mallocoed result buffer
70154. The commit flag. Usually 0. >0 for commit
70155. DETACH => ID
70156. Discard the candidate path from further consideration
70157. ***** End of os.h *****
70158. Expression to be appended. Might be NULL
70159. **comment:** TODO: Check there is no doclist index
 label: requirement
70160. Cursor of table holding data.
70161. True if in a write transaction
70162. Pointer to child node
70163. **comment:** ** Return a human-readable name for a constraint resolution action.
 label: code-design
70164. ** Parse times of the form HH:MM or HH:MM:SS or HH:MM:SS.FFFF. ** The HH, MM, and SS must each be exactly 2 digits. The ** fractional seconds FFFF can be one or more digits. ** ** Return 1 if there is a parsing error and 0 on success.
70165. ** Load the content from either the sqlite_stat4 or sqlite_stat3 table ** into the relevant Index.aSample[] arrays. ** ** Arguments zSql1 and zSql2 must point to SQL statements that return ** data equivalent to the following (statements are different for stat3, ** see the caller of this function for details): ** ** zSql1:
 SELECT idx,count(*) FROM %Q.sqlite_stat4 GROUP BY idx ** zSql2: SELECT idx,neq,nlt,ndlt,sample FROM %Q.sqlite_stat4 ** ** where %Q is replaced with the database name before the SQL is executed.
70166. ** This function is similar to sqlite3BtreeGetReserve(), except that it ** may only be called if it is guaranteed that the b-tree mutex is already ** held. ** ** This is useful in one special case in the backup API code where it is ** known that the shared b-tree mutex is held, but the mutex on the ** database handle that owns *p is not. In this case if sqlite3BtreeEnter() ** were to be called, it might collide with some other operation on the ** database handle that owns *p, causing undefined behavior.
70167. Resolve the column names in the WHERE clause.
70168. Add TK_IF_NULL_ROW opcodes on each replacement
70169. **comment:** Suppress warning about unused %extra_argument var
 label: code-design
70170. Generate the code to do the search. Each iteration of the for ** loop below generates code for a single nested loop of the VM ** program.
70171. ** Release all of the apPage[] pages for a cursor.
70172. fts3_tokenizer.c
70173. True if journal page is synced
70174. Est number of iterations of a query (10*log2(N))
70175. Advance the 'head' iterator of each phrase to the first offset that ** is greater than or equal to (iNext+nSnippet).
70176. Block id of next free slot in %_segments
70177. ** This function is used to copy the contents of the b-tree node stored ** on page pFrom to page pTo. If page pFrom was not a leaf page, then ** the pointer-map entries for each child page are updated so that the ** parent page stored in the pointer map is page pTo. If pFrom contained ** any cells with overflow page pointers, then the corresponding pointer ** map entries are also updated so that the parent page is page pTo. ** ** If pFrom is currently carrying any overflow cells (entries in the ** MemPage.apOvfl[] array), they are not copied to pTo. ** ** Before returning, page pTo is reinitialized using btreeInitPage(). ** ** The performance of this function is not critical. It is only used by ** the balance_shallow() and balance_deeper() procedures, neither of ** which are called often under normal circumstances.
70178. True to truncate db on commit
70179. One of UPDATE, DELETE, INSERT
70180. True if remove_diacritics=1 is set
70181. **comment:** OFFSET expression. NULL means not used.
 label: code-design
70182. Used to iterate through non-root layers
70183. Bulk memory available for allocation
70184. SQL text "PRAGMA %Q.page_size"
70185. Mutex used by sqlite3_initialize()
70186. Right outer join
70187. A MATCH operator. The right-hand-side must be a blob that ** can be cast into an RtreeMatchArg object. One created using ** an sqlite3_rtree_geometry_callback() SQL user function.
70188. ***** Begin NFS Locking *****
70189. Now check that the iter.nEmpty leaves following the current leaf ** (a) exist and (b) contain no terms.
70190. **comment:** Temporary buffer to use
 label: code-design
70191. Copy of pParse->pNewTable
70192. OUT: Pointer to hash index
70193. Write the path here
70194. Nesting should only be of limited depth
70195. Database page number to read data for
70196. Scratch allocations
70197. Verify that no lookaside memory was used by schema tables
70198. ** information about each column of an SQL table is held in an instance ** of this structure.
70199. 4x
70200. A boolean value
70201. Number of enterances
70202. Index of column in table
70203. ** An instance of this structure can hold a simple list of identifiers, ** such as the list "a,b,c" in the following statements: ** ** INSERT INTO t(a,b,c) VALUES ...; ** CREATE INDEX idx ON t(a,b,c); ** CREATE TRIGGER trig BEFORE UPDATE ON t(a,b,c) ...; ** ** The IdList.a.idx field is used when the IdList represents the list of ** column names after a table name in an INSERT statement. In the statement ** ** INSERT INTO t(a,b,c) ...; ** ** If "a" is the k-th column of table "t", then IdList.a[0].idx==k.
70204. Next are the tables used to determine what action to take based on the ** current state and lookahead token. These tables are used to implement ** functions that take a state number and lookahead value and return an ** action integer. ** ** Suppose the action integer is N. Then the action is determined as ** follows ** ** 0 <= N <= YY_MAX_SHIFT Shift N. That is, push the lookahead ** token onto the stack and goto state N. ** ** N between YY_MIN_SHIFTREDUCE Shift to an arbitrary state then ** and YY_MAX_SHIFTREDUCE reduce by rule N-YY_MIN_SHIFTREDUCE. ** ** N between YY_MIN_REDUCE Reduce by rule N-YY_MIN_REDUCE ** and YY_MAX_REDUCE ** ** N == YY_ERROR_ACTION A syntax error has occurred. ** ** N == YY_ACCEPT_ACTION The parser accepts its input. ** ** N == YY_NO_ACTION No such action. Denotes unused ** slots in the yy_action[] table. ** ** The action table is constructed as a single large table named yy_action[]. ** Given state S and lookahead X, the action is computed as either: ** ** (A) N = yy_action[yy_shift_ofst[S] + X] ** (B) N = yy_default[S] ** ** The (A) formula is preferred. The B formula is used instead if: ** (1) The yy_shift_ofst[S]+X value is out of range, or ** (2) yy_lookahead[yy_shift_ofst[S]+X] is not equal to X, or ** (3) yy_shift_ofst[S] equal YY_SHIFT_USE_DFLT. ** (Implementation note: YY_SHIFT_USE_DFLT is chosen so that ** YY_SHIFT_USE_DFLT+X will be out of range for all possible lookahead X. ** Hence only tests (1) and (2) need to be evaluated.) ** ** The formulas above are for computing the action when the lookahead is ** a terminal symbol. If the lookahead is a non-terminal (as occurs after ** a reduce

action) then the yy_reduce_ofst[] array is used in place of ** the yy_shift_ofst[] array and YY_REDUCE_USE_DFLT is used in place of ** YY_SHIFT_USE_DFLT. *** The following are the tables generated in this section: *** yy_action[] A single table containing all actions. ** yy_lookahead[] A table containing the lookahead for each entry in ** yy_action. Used to detect hash collisions. ** yy_shift_ofst[] For each state, the offset into yy_action for ** shifting terminals. ** yy_reduce_ofst[] For each state, the offset into yy_action for ** shifting non-terminals after a reduce. ** yy_default[] Default action for each state. *** ***** Begin parsing tables *****

70205. ** Advance the iterator passed as the second argument until it is at or ** past rowid iFrom. Regardless of the value of iFrom, the iterator is ** always advanced at least once.
70206. Checksum based on %_content contents
70207. If the TERM_LIKECOND flag is set, that means that the range search ** is sufficient to guarantee that the LIKE operator is true, so we ** can skip the call to the like(A,B) function. But this only works ** for strings. So do not skip the call to the function on the pass ** that compares BLOBS.
70208. Opcode: Eq P1 P2 P3 P4 P5 *** Synopsis: IF r[P3]==r[P1] *** Compare the values in register P1 and P3. If reg(P3)==reg(P1) then ** jump to address P2. Or if the SQLITE_STOREP2 flag is set in P5, then ** store the result of comparison in register P2. *** The SQLITE_AFF_MASK portion of P5 must be an affinity character - ** SQLITE_AFF_TEXT, SQLITE_AFF_INTEGER, and so forth. An attempt is made ** to coerce both inputs according to this affinity before the ** comparison is made. If the SQLITE_AFF_MASK is 0x00, then numeric ** affinity is used. Note that the affinity conversions are stored ** back into the input registers P1 and P3. So this opcode can cause ** persistent changes to registers P1 and P3. *** Once any conversions have taken place, and neither value is NULL, ** the values are compared. If both values are blobs then memcmp() is ** used to determine the results of the comparison. If both values ** are text, then the appropriate collating function specified in ** P4 is used to do the comparison. If P4 is not specified then ** memcmp() is used to compare text string. If both values are ** numeric, then a numeric comparison is used. If the two values ** are of different types, then numbers are considered less than ** strings and strings are considered less than blobs. *** If SQLITE_NULLEQ is set in P5 then the result of comparison is always either ** true or false and is never NULL. If both operands are NULL then the result ** of comparison is true. If either operand is NULL then the result is false. ** If neither operand is NULL the result is the same as it would be if ** the SQLITE_NULLEQ flag were omitted from P5. *** If both SQLITE_STOREP2 and SQLITE_KEEPNULL flags are set then the ** content of r[P2] is only changed if the new value is NULL or 0 (false). ** In other words, a prior r[P2] value will not be overwritten by 1 (true).
70209. ** Initialize a PMA-writer object.
70210. Required mapping size
70211. First field to be combined into the record
70212. Buffer to hold apValue/zTab/abPK/
70213. ** Free an allocated buffer obtained from sqlite3PageMalloc().
70214. Object to populate
70215. ***** End of fts3.c *****
70216. Character iIn was the close quote.
70217. Leave STATIC_MASTER mutex
70218. Array of term hash entries to scan
70219. ** End of interface to code in fts5_hash.c. *****
70220. Array of page numbers.
70221. First search for an existing entry. If one is found, this call is ** a no-op. Return early.
70222. Different conflict resolution strategies
70223. The new trigger
70224. OP_Open** cursor uses EQ seek only
70225. ** Return the size of an outstanding allocation, in bytes.
70226. A region of shared-memory
70227. This call is to merge all segments in the database to a single ** segment. The level of the new segment is equal to the numerically ** greatest segment level currently present in the database for this ** index. The idx of the new segment is always 0.
70228. "main" is always an acceptable alias for the primary database ** even if it has been renamed using SQLITE_DBCONFIG_MAINDBNAME.
70229. If the first integer value is followed by a ',', read the second ** integer value.
70230. nul-terminated term
70231. Database header content
70232. 50
70233. ** The phrase iterator passed as the second argument: *** * features at least one token that uses an incremental doclist, and *** * does not contain any deferred tokens. *** Advance it to the next matching document in the database and populate ** the Fts3Doclist.pList and nList fields. *** If there is no "next" entry and no error occurs, then *pbEof is set to ** 1 before returning. Otherwise, if no error occurs and the iterator is ** successfully advanced, *pbEof is set to 0. *** If an error occurs, return an SQLite error code. Otherwise, return ** SQLITE_OK.
70234. Name of collating sequence
70235. Field in (?, ?, ?) IN (SELECT...) vector
70236. ** Append the current term and doclist pointed to by cursor pCsr to the ** appendable b-tree segment opened for writing by pWriter. *** Return SQLITE_OK if successful, or an SQLite error code otherwise.
70237. Number of output leaves just written
70238. Buffer containing old.* record
70239. **comment:** FTS5_DETAIL_XXX value
label: code-design
70240. Pcache object page handle
70241. 39
70242. Set appropriate defaults on all indexes not in the sqlite_stat1 table
70243. If the path starts with a drive letter followed by the colon ** character, assume it is already a native Win32 path; otherwise, ** it must be converted to a native Win32 path via the Cygwin API ** prior to using it.
70244. 4x the size of current chunk in Mem3Block elements
70245. ** Generate code that evaluates the given expression and puts the result ** in register target. *** Also make a copy of the expression results into another "cache" register ** and modify the expression so that the next time it is evaluated, ** the result is a copy of the cache register. *** This routine is used for expressions that are used multiple ** times. They are evaluated once and the results of the expression ** are reused.
70246. Size of token in bytes
70247. nOut before IN() and WHERE adjustments
70248. Level of segments to scan
70249. Nothing to do
70250. Number of fields in PRIMARY KEY 1 for ROWID tables
70251. ** CAPI3REF: Result Codes ** KEYWORDS: {result code definitions} *** Many SQLite functions return an integer result code from the set shown ** here in order to indicate success or failure. *** New error codes may be added in future versions of SQLite. *** See also: [extended result code definitions]
70252. **comment:** ** The following routines are convenience wrappers around methods ** of the sqlite3_file object. This is mostly just syntactic sugar. All ** of this would be completely automatic if SQLite were coded using ** C++ instead of plain old C.
label: code-design
70253. Check if any parent key columns are being modified.
70254. ** This type is used as an fts3ExprIterate() context object while ** accumulating the data returned by the matchinfo() function.
70255. Event counters from parent frame
70256. Saved xAuth pointer
70257. 64-bit only if requested at compile-time
70258. ifndef SQLITE OMIT_INCRBLOB
70259. Sign-extend on a right shift of a negative number
70260. OUT: Unique index on parent table
70261. ** CAPI3REF: Mutex Verification Routines *** The sqlite3_mutex_held() and sqlite3_mutex_noheld() routines ** are intended for use inside assert() statements. The SQLite core ** never uses these routines except inside an assert() and applications ** are advised to follow the lead of the core. The SQLite core

only ** provides implementations for these routines when it is compiled ** with the SQLITE_DEBUG flag. External mutex implementations ** are only required to provide these routines if SQLITE_DEBUG is ** defined and if NDEBUG is not defined. ** ** These routines should return true if the mutex in their argument ** is held or not held, respectively, by the calling thread. ** ** The implementation is not required to provide versions of these ** routines that actually work. If the implementation does not provide working ** versions of these routines, it should at least provide stubs that always ** return true so that one does not get spurious assertion failures. ** ** If the argument to sqlite3_mutex_held() is a NULL pointer then ** the routine should return 1. This seems counter-intuitive since ** clearly the mutex cannot be held if it does not exist. But ** the reason the mutex does not exist is because the build is not ** using mutexes. And we do not want the assert() containing the ** call to sqlite3_mutex_held() to fail, so a non-zero return is ** the appropriate thing to do. The sqlite3_mutex_notheld() ** interface should also return 1 when given a NULL pointer.

70262. A SELECT statement to use as the data source

70263. Size limit for persistent journal files

70264. ** Added for 3.7.4

70265. Array of first token in each sentence

70266. Cursor of the canonical data source

70267. 181

70268. ** Locate and return an entry from the db.aCollSeq hash table. If the entry ** specified by zName and nName is not found and parameter 'create' is ** true, then create a new entry. Otherwise return NULL. ** ** Each pointer stored in the sqlite3.aCollSeq hash table contains an ** array of three CollSeq structures. The first is the collation sequence ** preferred for UTF-8, the second UTF-16le, and the third UTF-16be. ** ** Stored immediately after the three collation sequences is a copy of ** the collation sequence name. A pointer to this string is stored in ** each collation sequence structure.

70269. A no-op destructor

70270. iVersion

70271. Otherwise, flush the current node of layer iLayer to disk. ** Then allocate a new, empty sibling node. The key will be written ** into the parent of this node.

70272. Begin scanning through hash table entries. This loop runs once for each ** term/doclist currently stored within the hash table.

70273. Do nothing if table already exists

70274. Clear all content from pragma virtual table cursor.

70275. jumps if either operand is NULL

70276. ** Manage pPage's participation on the dirty list. Bits of the addRemove ** argument determines what operation to do. The 0x01 bit means first ** remove pPage from the dirty list. The 0x02 means add pPage back to ** the dirty list. Doing both moves pPage to the front of the dirty list.

70277. Raw numeric value stored in s

70278. EVIDENCE-OF: R-36592-02772 The SQLITE_CONFIG_COVERING_INDEX_SCAN ** option takes a single integer argument which is interpreted as a ** boolean in order to enable or disable the use of covering indices for ** full table scans in the query optimizer.

70279. The vector. List of expressions or a sub-SELECT

70280. ** Take actions at the end of an API call to indicate an OOM error

70281. ** Advance to the next element in the sorter. Return value: ** ** SQLITE_OK success ** SQLITE_DONE end of data ** otherwise some kind of error.

70282. Invoke AFTER DELETE trigger programs.

70283. Back pointer of the same list

70284. This branch causes a *balanced* tree to be generated. A valid tree ** is still generated without this branch, but the tree is wildly ** unbalanced and inefficient.

70285. Mark off any ORDER BY term X that is a column in the table of ** the current loop for which there is term in the WHERE ** clause of the form X IS NULL or X=? that reference only outer ** loops.

70286. **comment:** Read the next journal header from the journal file. If there are ** not enough bytes left in the journal file for a complete header, or ** it is corrupted, then a process must have failed while writing it. ** This indicates nothing more needs to be rolled back.

label: code-design

70287. Enter STATIC_MASTER mutex

70288. Generate a subroutine that outputs the current row of the A ** select as the next output row of the compound select.

70289. **comment:** An unused file descriptor

label: code-design

70290. **comment:** ** chngToIN holds a set of tables that *might* satisfy case 1. But ** we have to do some additional checking to see if case 1 really ** is satisfied. ** ** chngToIN will hold either 0, 1, or 2 bits. The 0-bit case means ** that there is no possibility of transforming the OR clause into an ** IN operator because one or more terms in the OR clause contain ** something other than == on a column in the single table. The 1-bit ** case means that every term of the OR clause is of the form ** "table.column=expr" for some single table. The one bit that is set ** will correspond to the common table. We still need to check to make ** sure the same column is used on all terms. The 2-bit case is when ** the all terms are of the form "table1.column=table2.column". It ** might be possible to form an IN operator with either table1.column ** or table2.column as the LHS if either is common to every term of ** the OR clause. ** ** Note that terms of the form "table.column1=table.column2" (the ** same table on both sides of the ==) cannot be optimized.

label: code-design

70291. An expression being tested

70292. True for native byte-order, false for non-native

70293. INT

70294. Allocate any page

70295. ** Record the fact that an affinity change has occurred on iCount ** registers starting with iStart.

70296. Pages in wal - keys for the sort

70297. It may be that this trigger has already been coded (or is in the ** process of being coded). If this is the case, then an entry with ** a matching TriggerPrg.pTrigger field will be present somewhere ** in the Parse.pTriggerPrg list. Search for such an entry.

70298. Register holding previous generated key

70299. True to inhibit all file I/O

70300. ** Close an open database and invalidate all cursors.

70301. ** Try to obtain a lock of type locktype on the database file. If ** a similar or greater lock is already held, this function is a no-op ** (returning SQLITE_OK immediately). ** ** Otherwise, attempt to obtain the lock using sqlite3OsLock(). Invoke ** the busy callback if the lock is currently not available. Repeat ** until the busy callback returns false or until the attempt to ** obtain the lock succeeds. ** ** Return SQLITE_OK on success and an error code if we cannot obtain ** the lock. If the lock is obtained successfully, set the Pager.state ** variable to locktype before returning.

70302. It is an error to call this function if the page is already ** part of the PGroup LRU list.

70303. This branch loads the value of a column that will not be changed ** into a register. This is done if there are no BEFORE triggers, or ** if there are one or more BEFORE triggers that use this value via ** a new.* reference in a trigger program.

70304. **comment:** ** hwtme.h contains inline assembler code for implementing ** high-performance timing routines.

label: code-design

70305. Bytes of content, or number of sub-nodes

70306. One entry for each of nCol columns

70307. 205

70308. The name of the trigger

70309. Invalidate all prepared statements whenever the TEMP database ** schema is changed. Ticket #1644

70310. ** Delete any previous value and set the value of pMem to be an ** empty boolean index.

70311. Figure out how much space is required for each journal file-handle ** (there are two of them, the main journal and the sub-journal).

70312. For leaf pages, the min-heap has already been initialized and the ** cells have already been inserted. But for internal pages, that has ** not yet been done, so do it now

70313. ** Commit the transaction currently in progress. ** ** This routine implements the second phase of a 2-phase commit. The ** sqlite3BtreeCommitPhaseOne() routine does the first phase and should ** be invoked prior to calling this routine. The sqlite3BtreeCommitPhaseOne() ** routine did all the work of writing information out to disk and flushing the ** contents so that they are written onto the disk platter. All this ** routine has to do is delete or truncate or zero the header in the ** the rollback journal (which causes the transaction to commit) and ** drop locks. ** ** Normally, if an error occurs while the pager layer is attempting to ** finalize the underlying journal file, this function returns an error and ** the upper layer will attempt a rollback. However, if the second argument ** is non-zero then this b-tree transaction is part of a multi-file ** transaction. In this case, the transaction has already been committed ** (by deleting a master

journal file) and the caller will ignore this ** functions return code. So, even if an error occurs in the pager layer, ** reset the b-tree objects internal state to indicate that the write ** transaction has been closed. This is quite safe, as the pager will have ** transitioned to the error state. *** ** This will release the write lock on the database file. If there ** are no active cursors, it also releases the read lock.

70314. 0
70315. "INSERT ... %_idx VALUES(?, ?, ?, ?)"
70316. Delete shared-memory if true
70317. EVIDENCE-OF: R-41220-51800 This option sets the threading mode to ** Serialized.
70318. The new table
70319. ** Perhaps the name is a reference to the ROWID
70320. Mask of tokens to highlight in snippet
70321. ** This function is called to generate code that runs when table pTab is ** being dropped from the database. The SrcList passed as the second argument ** to this function contains a single entry guaranteed to resolve to ** table pTab. *** ** Normally, no code is required. However, if either *** ** (a) The table is the parent table of a FK constraint, or ** (b) The table is the child table of a deferred FK constraint and it is ** determined at runtime that there are outstanding deferred FK ** constraint violations in the database, *** ** then the equivalent of "DELETE FROM <tbl>" is executed before dropping ** the table from the database. Triggers are disabled while running this ** DELETE, but foreign key actions are not.
70322. colsetlist
70323. **comment:** These look like magic numbers. But they are stable, as they are part ** of the definition of the SQLite file format, which may not change.
label: code-design
70324. same as TK_PLUS, in1, in2, out3
70325. Opcode: Move P1 P2 P3 * * *** Synopsis: r[P2@P3]=r[P1@P3] ** ** Move the P3 values in register P1..P1+P3-1 over into ** registers P2..P2+P3-1. Registers P1..P1+P3-1 are ** left holding a NULL. It is an error for register ranges ** P1..P1+P3-1 and P2..P2+P3-1 to overlap. It is an error ** for P3 to be less than 1.
70326. ** xSetOutputs callback used by detail=col when there is a column filter ** and there are 100 or more columns. Also called as a fallback from ** fts5IterSetOutputs_Col100 if the column-list spans more than one page.
70327. ** The argument is the first in a linked list of dirty pages connected ** by the PgHdr.pDirty pointer. This function writes each one of the ** in-memory pages in the list to the database file. The argument may ** be NULL, representing an empty list. In this case this function is ** a no-op. *** ** The pager must hold at least a RESERVED lock when this function ** is called. Before writing anything to the database file, this lock ** is upgraded to an EXCLUSIVE lock. If the lock cannot be obtained, ** SQLITE_BUSY is returned and no data is written to the database file. *** ** If the pager is a temp-file pager and the actual file-system file ** is not yet open, it is created and opened before any data is ** written out. *** ** Once the lock has been upgraded and, if necessary, the file opened, ** the pages are written out to the database file in list order. Writing ** a page is skipped if it meets either of the following criteria: *** ** * The page number is greater than Pager.dbSize, or *** * The PGHDR_DONT_WRITE flag is set on the page. *** ** If writing out a page causes the database file to grow, Pager.dbFileSize ** is updated accordingly. If page 1 is written out, then the value cached ** in Pager.dbFileVers[] is updated to match the new value stored in ** the database file. *** ** If everything is successful, SQLITE_OK is returned. If an IO error ** occurs, an IO error code is returned. Or, if the EXCLUSIVE lock cannot ** be obtained, SQLITE_BUSY is returned.
70328. DELETE
70329. xClose
70330. The collation sequence must be defined at this point, even if ** the user deletes the collation sequence after the vdbe program is ** compiled (this was not always the case).
70331. OUT: Pointer to current position-list
70332. True to count changes
70333. OUT: Map of index columns in pFKey
70334. Search the rowid of the table
70335. **comment:** ** All individual term iterators in pNear are guaranteed to be valid when ** this function is called. This function checks if all term iterators ** point to the same rowid, and if not, advances them until they do. ** If an EOF is reached before this happens, *pbEof is set to true before ** returning. *** ** SQLITE_OK is returned if an error occurs, or an SQLite error code ** otherwise. It is not considered an error code if an iterator reaches ** EOF.
label: code-design
70336. EACH => ID
70337. Address of opcode that determines the IN is true
70338. If the temp database has been explicitly named as part of the ** pragma, make sure it is open.
70339. ** The maximum number of opcodes in a VDBE program. ** Not currently enforced.
70340. WHERE clause on PK columns
70341. Compensate for pPager->iDataVersion++;
70342. Copy of pStr1 - RHS of LIKE/GLOB operator
70343. Generate code that executes if the new index entry is not unique
70344. Skip leading zeros.
70345. Pointer to a new child page
70346. If the user has configured a chunk-size for this file, truncate the ** file so that it consists of an integer number of chunks (i.e. the ** actual file size after the operation may be larger than the requested ** size).
70347. Total number of I/O Errors
70348. IGNORE => ID
70349. Object to return
70350. Create an Expr object representing an SQL expression like: *** <parent-key1> = <child-key1> AND <parent-key2> = <child-key2> ... *** ** The collation sequence used for the comparison should be that of ** the parent key columns. The affinity of the parent key column should ** be applied to each child key value before the comparison takes place.
70351. Result written here
70352. ** Return the page number associated with frame iFrame in this WAL.
70353. ** This is called when the database connection passed as an argument is ** being closed. The connection is removed from the blocked list.
70354. The point is to increment the last character before the first ** wildcard. But if we increment '@', that will push it into the ** alphabetic range where case conversions will mess up the ** inequality. To avoid this, make sure to also run the full ** LIKE on all candidate expressions by clearing the isComplete flag
70355. Remove trailing zeros and the "." if no digits follow the ".".
70356. Mask of events to be traced
70357. Current lock held on database file
70358. __QNXNTO__
70359. Insert on a main table b-tree
70360. ** CAPI3REF: Obtaining SQL Values ** METHOD: sqlite3_value ** ** Summary: ** <blockquote><table border=0 cellpadding=0 cellspacing=0> ** <tr><td>sqlite3_value_blob<td>→;<td>BLOB value ** <tr><td>sqlite3_value_double<td>→;<td>REAL value ** <tr><td>sqlite3_value_int<td>→;<td>32-bit INTEGER value ** <tr><td>sqlite3_value_int64<td>→;<td>64-bit INTEGER value ** <tr><td>sqlite3_value_pointer<td>→;<td>Pointer value ** <tr><td>sqlite3_value_text<td>→;<td>UTF-8 TEXT value ** <tr><td>sqlite3_value_text16<td>→;<td>UTF-16 TEXT value in ** the native byteorder ** <tr><td>sqlite3_value_text16be<td>→;<td>UTF-16be TEXT value ** <tr><td>sqlite3_value_text16le<td>→;<td>UTF-16le TEXT value ** <tr><td> <td> ** <tr><td>sqlite3_value_bytes<td>→;<td>Size of a BLOB ** or a UTF-8 TEXT in bytes ** <tr><td>sqlite3_value_bytes16 </td> ** <td>→; <td>Size of UTF-16 ** TEXT in bytes ** <tr><td>sqlite3_value_type→;<td>Default ** datatype of the value ** <tr><td>sqlite3_value_numeric_type </td> ** <td>→; <td>Best numeric datatype of the value ** </table></blockquote> ** ** Details: ** ** These routines extract type, size, and content information from ** [protected sqlite3_value] objects. Protected sqlite3_value objects ** are used to pass parameter information into implementation of ** [application-defined SQL functions] and [virtual tables]. *** ** These routines work only with [protected sqlite3_value] objects. ** Any attempt to use these routines on an [unprotected sqlite3_value] ** is not threadsafe. *** ** ^These routines work just like the corresponding [column access functions] ** except that these routines take a single [protected sqlite3_value] object ** pointer instead of a [sqlite3_stmt*] pointer and an integer column number. *** ** ^The sqlite3_value_text16() interface extracts a UTF-16 string ** in the native byte-order of the host machine. ^The ** sqlite3_value_text16be() and sqlite3_value_text16le() interfaces ** extract UTF-16 strings as big-endian and little-endian respectively. *** ** ^If [sqlite3_value]

object V was initialized ** using [sqlite3_bind_pointer(S,I,P,X,D)] or [sqlite3_result_pointer(C,P,X,D)] ** and if X and Y are strings that compare equal according to strcmp(X,Y), ** then sqlite3_value_pointer(V,Y) will return the pointer P. ^Otherwise, ** sqlite3_value_pointer(V,Y) returns a NULL. The sqlite3_bind_pointer() ** routine is part of the [pointer passing interface] added for SQLite 3.20.0. *** ^The sqlite3_value_type(V) interface returns the ** [SQLITE_INTEGER | datatype code] for the initial datatype of the ** [sqlite3_value] object V. The returned value is one of [SQLITE_INTEGER], ** [SQLITE_FLOAT], [SQLITE_TEXT], [SQLITE_BLOB], or [SQLITE_NULL].^ ** Other interfaces might change the datatype for an sqlite3_value object. ** For example, if the datatype is initially SQLITE_INTEGER and ** sqlite3_value_text(V) is called to extract a text value for that ** integer, then subsequent calls to sqlite3_value_type(V) might return ** SQLITE_TEXT. Whether or not a persistent internal datatype conversion ** occurs is undefined and may change from one release of SQLite to the next. *** ^The sqlite3_value_numeric_type() interface attempts to apply ** numeric affinity to the value. This means that an attempt is ** made to convert the value to an integer or floating point. If ** such a conversion is possible without loss of information (in other ** words, if the value is a string that looks like a number) ** then the conversion is performed. Otherwise no conversion occurs. ** The [SQLITE_INTEGER | datatype] after conversion is returned.)^ ** Please pay particular attention to the fact that the pointer returned ** from [sqlite3_value_blob()], [sqlite3_value_text()], or ** [sqlite3_value_text16()] can be invalidated by a subsequent call to ** [sqlite3_value_bytes()], [sqlite3_value_bytes16()], [sqlite3_value_text()], ** or [sqlite3_value_text16()]. ** These routines must be called from the same thread as ** the SQL function that supplied the [sqlite3_value*] parameters.

70361. Number of cells
70362. ** Close an RBU handle. *** If the RBU update has been completely applied, mark the RBU database ** as fully applied. Otherwise, assuming no error has occurred, save the ** current state of the RBU update application to the RBU database. *** If an error has already occurred as part of an sqlite3rbu_step() ** or sqlite3rbu_open() call, or if one occurs within this function, an ** SQLite error code is returned. Additionally, if pzErrmsg is not NULL, ** *pzErrmsg may be set to point to a buffer containing a utf-8 formatted ** English language error message. It is the responsibility of the caller to ** eventually free any such buffer using sqlite3_free(). *** Otherwise, if no error occurs, this function returns SQLITE_OK if the ** update has been partially applied, or SQLITE_DONE if it has been ** completely applied.
70363. ** An array of names of all compile-time options. This array should ** be sorted A-Z. *** This array looks large, but in a typical installation actually uses ** only a handful of compile-time options, so most times this array is usually ** rather short and uses little memory space.
70364. Number of fields in the table or index
70365. Unpacked version of new.* record
70366. LIKE_KW => ID
70367. Jump here to exit loop
70368. Single VALUES term with multiple rows
70369. Changes whenever database content changes
70370. Size of vectors for this IN operator
70371. 0x16
70372. Output string - not real JSON
70373. ** Calling this function indicates that subsequent calls to ** fts3PendingTermsAdd() are to add term/position-list pairs for the ** contents of the document with docid iDocid.
70374. If non-NULL, points to sqlite3_vtab.base.zErrmsg. Often NULL.
70375. xFunc, void*
70376. EVIDENCE-OF: R-19854-42126 There are three arguments to ** SQLITE_CONFIG_HEAP: An 8-byte aligned pointer to the memory, the ** number of bytes in the memory buffer, and the minimum allocation size.
70377. ** This routine takes the module argument that has been accumulating ** in pParse->zArg[] and appends it to the list of arguments on the ** virtual table currently under construction in pParse->pTable.
70378. ** Return a pointer to the symbol zSymbol in the dynamic library pHandle.
70379. Index into mem5.aiFreelist[]
70380. Value to bind
70381. Append the position lists
70382. Above requires no mutex. Use mutex below for variable that follow.
70383. ** Convert a DbPage obtained from the pager into a MemPage used by ** the btree layer.
70384. True if we are at the end of input
70385. Estimated rows returned by this scan
70386. Cannot start a write transaction without first holding a read ** transaction.
70387. Cursor used to scan change record
70388. Function name (nul-terminated)
70389. Table columns
70390. ** This function returns a pointer to a nul-terminated string in memory ** obtained from sqlite3DbMalloc(). If sqlite3.nVdbeExec is 1, then the ** string contains a copy of zRawSql but with host parameters expanded to ** their current bindings. Or, if sqlite3.nVdbeExec is greater than 1, ** then the returned string holds a copy of zRawSql with "-- " prepended ** to each line of text. *** If the SQLITE_TRACE_SIZE_LIMIT macro is defined to an integer, then ** then long strings and blobs are truncated to that many bytes. This ** can be used to prevent unreasonably large trace strings when dealing ** with large (multi-megabyte) strings and blobs. *** The calling function is responsible for making sure the memory returned ** is eventually freed. ** ** ALGORITHM: Scan the input string looking for host parameters in any of ** these forms: ?, ?N, \$A, @A, :A. Take care to avoid text within ** string literals, quoted identifier names, and comments. For text forms, ** the host parameter index is found by scanning the prepared ** statement for the corresponding OP_Variable opcode. Once the host ** parameter index is known, locate the value in p->aVar[]. Then render ** the value as a literal in place of the host parameter name.
70391. Rowid in main table of the key
70392. If the RHS begins with a digit or a minus sign, then the LHS must ** be an ordinary column (not a virtual table column) with TEXT affinity. ** Otherwise the LHS might be numeric and "lhs >= rhs" would be false ** even though "lhs LIKE rhs" is true. But if the RHS does not start ** with a digit or '-', then "lhs LIKE rhs" will always be false if ** the LHS is numeric and so the optimization still works.
70393. ** Free memory. *** The outer layer memory allocator prevents this routine from ** being called with pPrior==0.
70394. ** Set the "use-threads" flag on object pIncr.
70395. ALTER
70396. The escape char (LIKE) or 't' (GLOB)
70397. Pointer to buffer containing string
70398. Attempt to write a readonly database
70399. _PCACHE_H_
70400. Page to edit
70401. If unable to open an sqlite3_blob on the desired row, that can only ** be because the shadow tables hold erroneous data.
70402. Handle that must hold lock
70403. Mask of TRIGGER_BEFORE|TRIGGER_AFTER
70404. ***** End of btreeInt.h *****
70405. Forward Declarations
70406. True for json_tree(). False for json_each()
70407. **comment:** ** The fts3BestSnippet() function often selects snippets that end with a ** query term. That is, the final term of the snippet is always a term ** that requires highlighting. For example, if 'X' is a highlighted term ** and '.' is a non-highlighted term, BestSnippet() may select: ** ***X.....X ** ** This function "shifts" the beginning of the snippet forward in the ** document so that there are approximately the same number of ** non-highlighted terms to the right of the final highlighted term as there ** are to the left of the first highlighted term. For example, to this: ** ***X.....X.... ** ** This is done as part of extracting the snippet text, not when selecting ** the snippet. Snippet selection is done based on doclists only, so there ** is no way for fts3BestSnippet() to know whether or not the document ** actually contains terms that follow the final highlighted term.
label: documentation
70408. SQLITE_STATUS_PAGECACHE_USED
70409. Cache zFilename in the locking context (APP and dotlock override) for ** proxyLock activation is possible (remote proxy is based on db name) ** zFilename remains valid until file is closed, to support
70410. Write-ahead log connection

70411. content=? parameter (or NULL)
70412. The proposed effective sample is a prefix of sample aSample[iSamp]. ** Specifically, the shortest prefix of at least (1 + iTest%nField) ** fields that is greater than the previous effective sample.
70413. Mem address which causes query abort if positive
70414. Copy entry from i1
70415. Data associated with this element
70416. **comment:** A complete hash table is an instance of the following structure. ** The internals of this structure are intended to be opaque -- client ** code should not attempt to access or modify the fields of this structure ** directly. Change this structure only by using the routines below. ** However, many of the "procedures" and "functions" for modifying and ** accessing this structure are really macros, so we can't really make ** this structure opaque.
label: code-design
70417. Add all table columns to the PRIMARY KEY index
70418. ** Cancel a prior call to sqlite3_auto_extension. Remove xInit from the ** set of routines that is invoked for each new database connection, if it ** is currently on the list. If xInit is not on the list, then this ** routine is a no-op. ** ** Return 1 if xInit was found on the list and removed. Return 0 if xInit ** was not on the list.
70419. Value to associate with zName
70420. Address of regReturn init
70421. ** This function is used to help iterate through a position-list. A position ** list is a list of unique integers, sorted from smallest to largest. Each ** element of the list is represented by an FTS3 varint that takes the value ** of the difference between the current element and the previous one plus ** two. For example, to store the position-list: ** ** 4 9 113 ** ** the three varints: ** ** 6 7 106 ** ** are encoded. ** ** When this function is called, *pp points to the start of an element of ** the list. *piPos contains the value of the previous entry in the list. ** After it returns, *piPos contains the value of the next element of the ** list and *pp is advanced to the following varint.
70422. True for 'notindexed' columns
70423. 2 -> COMPRESS
70424. The constraint to test
70425. First search the list for an existing lock on this table.
70426. Number of columns in index
70427. **comment:** Which level of pWInfo->a[] should be coded
label: code-design
70428. synopsis: r[P3]=mkrec(r[P1@P2])
70429. Ignore tables with no primary keys
70430. Drop the WAL write-lock, if any. Also, if the connection was in ** locking_mode=exclusive mode but is no longer, drop the EXCLUSIVE ** lock held on the database file.
70431. ** An in-memory list of objects to be sorted. ** ** If aMemory==0 then each object is allocated separately and the objects ** are connected using SorterRecord.u.pNext. If aMemory!=0 then all objects ** are stored in the aMemory[] bulk memory, one right after the other, and ** are connected using SorterRecord.u.iNext.
70432. Number of slots of aNode[] used
70433. Existing WITH clause, or NULL
70434. <rdar://problem/6778339>
70435. **comment:** ONEPASS_XXX value from where.c
label: code-design
70436. EVIDENCE-OF: R-51213-46414 The SQLITE_CONFIG_GETMALLOC option takes a ** single argument which is a pointer to an instance of the ** sqlite3_mem_methods structure. The sqlite3_mem_methods structure is ** filled with the currently defined memory allocation routines.
70437. (<tbl> MATCH ?)
70438. Size of buffer zNext in bytes
70439. Done with the mutex
70440. Flags that may be passed by the tokenizer implementation back to FTS5 ** as the third argument to the supplied xToken callback.
70441. Largest iWalFrame value in aFrame[]
70442. ** Close a file. Make sure the lock has been released before closing.
70443. OUT: Set to point to affinity string
70444. The Pager.sectorSize variable may have been updated while rolling ** back a journal created by a process with a different sector size ** value. Reset it to the correct value for this process.
70445. ** Return permyriadage progress indications for the two main stages of ** an RBU update.
70446. Used to iterate through instances
70447. See note about index shifting on OP_ReadCookie
70448. 43
70449. Number of bytes written
70450. 85
70451. The database schema changed
70452. defined(SQLITE_TEST)
70453. IN/OUT: Elements in *paRight
70454. Do not lock (except in WAL mode)
70455. ** Given a SELECT statement, generate a Table structure that describes ** the result set of that SELECT.
70456. Configuration cookie
70457. ** The ctype.h header is needed for non-ASCII systems. It is also ** needed by FTS3 when FTS3 is included in the amalgamation.
70458. pPage and up to NB siblings after balancing
70459. Collation and sort-order information
70460. Step 1: Search the hash table for an existing entry.
70461. Read the page number and page data from the journal or sub-journal ** file. Return an error code to the caller if an IO error occurs.
70462. Pointer to phrase expression
70463. Zero the newly allocated slots
70464. ** Allowed values for winFile.ctrlFlags
70465. Size of log file
70466. ncell
70467. **comment:** F: Old-style sqlite3_rtree_geometry_callback()
label: code-design
70468. No aggregate functions and no GROUP BY clause
70469. Value of 'mx_payload' column
70470. 1 -> PREFIX
70471. Used to loop through terms
70472. IMP: R-47871-25994
70473. Probability of truth for this expression
70474. For a virtual table, or a table with no primary key, the ** SELECT statement is: ** ** SELECT <cols>, rbu_control, rbu_rowid FROM ** ** Hence column_value(pIter->nCol+1).
70475. Fill in this structure
70476. Use of ONEPASS not allowed
70477. SQLITE_SOUNDEX
70478. ***** Interface to code in fts5_storage.c. fts5_storage.c contains contains ** code to access the data stored in the %_content and %_docszie tables.
70479. The specific ORDER BY term

70480. Copy data from this buffer to the file
70481. Ran out of input before finding an opening bracket. Return NULL.
70482. The columns of the result set
70483. SQLITE_PAGER_H
70484. isMallocInit
70485. ** A connection to a fulltext index is an instance of the following ** structure. The xCreate and xConnect methods create an instance ** of this structure and xDestroy and xDisconnect free that instance. ** All other methods receive a pointer to the structure as one of their ** arguments.
70486. Number of args (always 2)
70487. Address of OP_OpenEphemeral
70488. Code the OP_Affinity opcode if there is anything left to do.
70489. FTS5_STRING, AND, OR or NOT
70490. Decrement the free-list count by 1. Set iTrunk to the index of the ** first free-list trunk page. iPrevTrunk is initially 1.
70491. OP_MaxPgcnt enforces this
70492. Number of full pages written to DB
70493. Return the page in this parameter
70494. synopsis: fkctr[P1]+=P2
70495. Write result boolean here
70496. New db size (or 0 for non-commit frames)
70497. got the lock, unlock it
70498. One of TK_DELETE, TK_UPDATE, TK_INSERT, TK_SELECT
70499. Current WITH clause, or NULL
70500. name of the database file
70501. True if no such function exists
70502. **comment:** ** Malloc function used within this file to allocate space from the buffer ** configured using sqlite3_config(SQLITE_CONFIG_PAGECACHE) option. If no ** such buffer exists or there is no space left in it, this function falls ** back to sqlite3Malloc(). ** ** Multiple threads can run this routine at the same time. Global variables ** in pcache1 need to be protected via mutex.
label: code-design
70503. Content of the row to be inserted
70504. CASCADE
70505. Names of result columns
70506. ** Disconnect from or destroy a statsfs virtual table.
70507. ** Return TRUE if the mask of type in eType matches the type of the ** allocation p. Also return true if p==NULL. ** ** This routine is designed for use within an assert() statement, to ** verify the type of an allocation. For example: *** assert(sqlite3MemdebugHasType(p, MEMTYPE_HEAP));
70508. ** Read a 32-bit integer from the given file descriptor. Store the integer ** that is read in *pRes. Return SQLITE_OK if everything worked, or an ** error code is something goes wrong. ** ** All values are stored on disk as big-endian.
70509. Value to return via *pnBind
70510. defined(SQLITE_ENABLE_UPDATE_DELETE_LIMIT) && !defined(SQLITE OMIT_SUBQUERY)
70511. Create the expression "OLD.zToCol = zFromCol". It is important ** that the "OLD.zToCol" term is on the LHS of the = operator, so ** that the affinity and collation sequence associated with the ** parent table are used for the comparison.
70512. Cost multiplier for using this table
70513. 1160
70514. orderby_opt ::= ORDER BY sortlist
70515. Breakup of pTerm into subterms
70516. If the wal-index header is still malformed even while holding ** a WRITE lock, it can only mean that the header is corrupted and ** needs to be reconstructed. So run recovery to do exactly that.
70517. State 0. In this state the integer just read was a docid.
70518. ** Parameter aData must point to a buffer of pPager->pageSize bytes ** of data. Compute and return a checksum based on the contents of the ** page of data and the current value of pPager->cksumInit. ** ** This is not a real checksum. It is really just the sum of the ** random initial value (pPager->cksumInit) and every 200th byte ** of the page data, starting with byte offset (pPager->pageSize%200). ** Each byte is interpreted as an 8-bit unsigned integer. ** ** Changing the formula used to compute this checksum results in an ** incompatible journal file format. ** ** If journal corruption occurs due to a power failure, the most likely ** scenario is that one end or the other of the record will be changed. ** It is much less likely that the two ends of the journal record will be ** correct and the middle be corrupt. Thus, this "checksum" scheme, ** though fast and simple, catches the mostly likely kind of corruption.
70519. Size of aInst[] array (entries / 3)
70520. **comment:** Remove harmful bits from the flags parameter ** ** The SQLITE_OPEN_NOMUTEX and SQLITE_OPEN_FULLMUTEX flags were ** dealt with in the previous code block. Besides these, the only ** valid input flags for sqlite3_open_v2() are SQLITE_OPEN_READONLY, ** SQLITE_OPEN_READWRITE, SQLITE_OPEN_CREATE, SQLITE_OPEN_SHARED_CACHE, ** SQLITE_OPEN_PRIVATECACHE, and some reserved bits. Silently mask ** off all other flags.
label: code-design
70521. In order to be able to rollback, an in-memory database must journal ** the page we are moving from.
70522. ** Routines to read and write variable-length integers. These used to ** be defined locally, but now we use the varint routines in the util.c ** file.
70523. Values between 268435456 and 34359738367
70524. xRollback
70525. ** Open a database file. ** ** zFilename is the name of the database file. If zFilename is NULL ** then an ephemeral database is created. The ephemeral database might ** be exclusively in memory, or it might use a disk-based memory cache. ** Either way, the ephemeral database will be automatically deleted ** when sqlite3BtreeClose() is called. ** ** If zFilename is ":memory:" then an in-memory database is created ** that is automatically destroyed when it is closed. ** ** The "flags" parameter is a bitmask that might contain bits like ** BTREE OMIT_JOURNAL and/or BTREE MEMORY. ** ** If the database is already opened in the same database connection ** and we are in shared cache mode, then the open will fail with an ** SQLITE_CONSTRAINT error. We cannot allow two or more BtShared ** objects in the same database connection since doing so will lead ** to problems with locking.
70526. OUT: Final number of frames in log
70527. Array of mapped shared-memory regions
70528. This branch is needed to avoid a (harmless) buffer overread. The ** special comment alerts the mutation tester that the correct answer ** is obtained even if the branch is omitted
70529. One of the WhereLoop objects
70530. ** True for an RBU vacuum handle, or false otherwise.
70531. **comment:** Number of unused bytes on the page
label: code-design
70532. ** Indicate that the accumulator load should be skipped on this ** iteration of the aggregate loop.
70533. Do not defer seeks on main table
70534. 1 for 'tokenchars', 0 for 'separators'
70535. Cursor assigned to each bit
70536. Size of buffer aBuf
70537. Actual size of mapping at pMapRegion
70538. Iterator to read from
70539. ** Make a full copy of pFrom into pTo. Prior contents of pTo are ** freed before the copy is made.
70540. ** Sleep for a little while. Return the amount of time slept. ** The argument is the number of microseconds we want to sleep. ** The return value is the number of microseconds of sleep actually ** requested from the underlying operating system, a number which ** might be greater than or equal to the argument, but not less ** than the argument.
70541. ** Link the chunk at index i into either the appropriate ** small chunk list, or into the large chunk hash table.

70542. Potential covering index (or NULL)
 70543. If not negative, use aCol[iPKey] as the rowid
 70544. **comment:** Advance to the next document
label: documentation
 70545. One element of this array for each open cursor
 70546. Sanity check: For a non-corrupt database file one of the following ** must be true: ** (1) We found one or more cells (cntNew[0]>0), or ** (2) pPage is a virtual root page. A virtual root page is when ** the real root page is page 1 and we are the only child of ** that page.
 70547. ** Implementation of the printf() function.
 70548. Copy of 2nd argument to xTokenize()
 70549. Make sure "isView" and other macros defined above are undefined. Otherwise ** they may interfere with compilation of other functions in this file ** (or in another file, if this file becomes part of the amalgamation).
 70550. ** Restart iteration for expression pExpr so that the next call to ** fts3EvalNext() visits the first row. Do not allow incremental ** loading or merging of phrase doclists for this iteration. ** ** If *pRc is other than SQLITE_OK when this function is called, it is ** a no-op. If an error occurs within this function, *pRc is set to an ** SQLite error code before returning.
 70551. Number of entries in apCsr
 70552. This is a pointer type. There may be a flag to indicate what to ** do with the pointer.
 70553. ccons ::= DEFAULT PLUS term
 70554. NATURAL
 70555. ** The following routine destroys a virtual machine that is created by ** the sqlite3_compile() routine. The integer returned is an SQLITE_** success/failure code that describes the result of executing the virtual ** machine. ** ** This routine sets the error code and string returned by ** sqlite3_errcode(), sqlite3_errmsg() and sqlite3_errmsg16().
 70556. index leaf
 70557. Mutex controlling the lock
 70558. ** Query the size of the file in bytes.
 70559. The full-text index is stored in a series of b+tree (-like) ** structures called segments which map terms to doclists. The ** structures are like b+trees in layout, but are constructed from the ** bottom up in optimal fashion and are not updatable. Since trees ** are built from the bottom up, things will be described from the ** bottom up. *** **** Varints **** ** The basic unit of encoding is a variable-length integer called a ** varint. We encode variable-length integers in little-endian order ** using seven bits * per byte as follows: ** ** KEY: ** A = 0xxxxxxx 7 bits of data and one flag bit ** B = 1xxxxxxxx 7 bits of data and one flag bit *** ** 7 bits - A ** 14 bits - BA ** 21 bits - BBA ** and so on. **** This is similar in concept to how sqlite encodes "varints" but ** the encoding is not the same. SQLite varints are big-endian ** are limited to 9 bytes in length whereas FTS3 varints are ** little-endian and can be up to 10 bytes in length (in theory). *** ** Example encodings: *** ** 1: 0x01 ** 127: 0x7f ** 128: 0x81 0x00 ** *** **** Document lists **** ** A doclist (document list) holds a docid-sorted list of hits for a ** given term. Doclists hold docids and associated token positions. ** A docid is the unique integer identifier for a single document. ** A position is the index of a word within the document. The first ** word of the document has a position of 0. ** ** FTS3 used to optionally store character offsets using a compile-time ** option. But that functionality is no longer supported. ** ** A doclist is stored like this: ** ** array { ** varint docid; (delta from previous doclist) ** array { (position list for column 0) ** varint position; (2 more than the delta from previous position) ** } ** array { ** varint POS_COLUMN; (marks start of position list for new column) ** varint column; (index of new column) ** array { ** varint position; (2 more than the delta from previous position) ** } ** } ** varint POS_END; (marks end of positions for this document. ** } *** ** Here, array { X } means zero or more occurrences of X, adjacent in ** memory. A "position" is an index of a token in the token stream ** generated by the tokenizer. Note that POS_END and POS_COLUMN occur ** in the same logical place as the position element, and act as sentinals ** ending a position list array. POS_END is 0. POS_COLUMN is 1. ** The positions numbers are not stored literally but rather as two more ** than the difference from the prior position, or the just the position plus ** 2 for the first position. Example: *** ** label: A B C D E F G H I J K ** value: 123 5 9 1 1 14 35 0 234 72 0 ** ** The 123 value is the first docid. For column zero in this document ** there are two matches at positions 3 and 10 (5-2 and 9-2+3). The 1 ** at D signals the start of a new column; the 1 at E indicates that the ** new column is column number 1. There are two positions at 12 and 45 ** (14-2 and 35-2+12). The 0 at H indicate the end-of-document. The ** 234 at I is the delta to next docid (357). It has one position 70 ** (72-2) and then terminates with the 0 at K. ** ** A "position-list" is the list of positions for multiple columns for ** a single docid. A "column-list" is the set of positions for a single ** column. Hence, a position-list consists of one or more column-lists, ** a document record consists of a docid followed by a position-list and ** a doclist consists of one or more document records. ** ** A bare doclist omits the position information, becoming an ** array of varint-encoded docids. *** **** Segment leaf nodes **** ** Segment leaf nodes store terms and doclists, ordered by term. Leaf ** nodes are written using LeafWriter, and read using LeafReader (to ** iterate through a single leaf node's data) and LeavesReader (to ** iterate through a segment's entire leaf layer). Leaf nodes have ** the format: *** ** varint iHeight; (height from leaf level, always 0) ** varint nTerm; (length of first term) ** char pTerm[nTerm]; (content of first term) ** varint nDoclist; (length of term's associated doclist) ** char pDoclist[nDoclist]; (content of doclist) ** array { ** (further terms are delta-encoded) ** varint nPrefix; (length of prefix shared with previous term) ** varint nSuffix; (length of unshared suffix) ** char pTermSuffix[nSuffix]; (unshared suffix of next term) ** varint nDoclist; (length of term's associated doclist) ** char pDoclist[nDoclist]; (content of doclist) ** } *** ** Here, array { X } means zero or more occurrences of X, adjacent in ** memory. ** ** Leaf nodes are broken into blocks which are stored contiguously in ** the %_segments table in sorted order. This means that when the end ** of a node is reached, the next term is in the node with the next ** greater node id. *** ** New data is spilled to a new leaf node when the current node ** exceeds LEAF_MAX bytes (default 2048). New data which itself is ** larger than STANDALONE_MIN (default 1024) is placed in a standalone ** node (a leaf node with a single term and doclist). The goal of ** these settings is to pack together groups of small doclists while ** making it efficient to directly access large doclists. The ** assumption is that large doclists represent terms which are more ** likely to be query targets. ** ** TODO(shess) It may be useful for blocking decisions to be more ** dynamic. For instance, it may make more sense to have a 2.5k leaf ** node rather than splitting into 2k and .5k nodes. My intuition is ** that this might extend through 2x or 4x the pagesize. *** **** Segment interior nodes **** ** Segment interior nodes store blockids for subtree nodes and terms ** to describe what data is stored by the each subtree. Interior ** nodes are written using InteriorWriter, and read using ** InteriorReader. InteriorWriters are created as needed when ** SegmentWriter creates new leaf nodes, or when an interior node ** itself grows too big and must be split. The format of interior ** nodes: *** ** varint iHeight; (height from leaf level, always >0) ** varint iBlockid; (block id of node's leftmost subtree) ** optional { ** varint nTerm; (length of first term) ** char pTerm[nTerm]; (content of first term) ** array { ** (further terms are delta-encoded) ** varint nPrefix; (length of shared prefix with previous term) ** varint nSuffix; (length of unshared suffix) ** char pTermSuffix[nSuffix]; (unshared suffix of next term) ** } ** } *** ** Here, optional { X } means an optional element, while array { X } ** means zero or more occurrences of X, adjacent in memory. ** ** An interior node encodes n terms separating n+1 subtrees. The ** subtree blocks are contiguous, so only the first subtree's blockid ** is encoded. The subtree at iBlockid will contain all terms less ** than the first term encoded (or all terms if no term is encoded). ** Otherwise, for terms greater than or equal to pTerm[i] but less ** than pTerm[i+1], the subtree for that term will be rooted at ** iBlockid+i. Interior nodes only store enough term data to ** distinguish adjacent children (if the rightmost term of the left ** child is "something", and the leftmost term of the right child is ** "wicked", only "w" is stored). ** ** New data is spilled to a new interior node at the same height when ** the current node exceeds INTERIOR_MAX bytes (default 2048). ** INTERIOR_MIN_TERMS (default 7) keeps large terms from monopolizing ** interior nodes and making the tree too skinny. The interior nodes ** at a given height are naturally tracked by interior nodes at ** height+1, and so on. *** **** Segment directory **** ** The segment directory in table %_segdir stores meta-information for ** merging and deleting segments, and also the root node of the ** segment's tree. *** ** The root node is the top node of the segment's tree after encoding ** the entire segment, restricted to ROOT_MAX bytes (default 1024). ** This could be either a leaf node or an interior node. If the top ** node requires more than ROOT_MAX bytes, it is flushed to %_segments ** and a new root interior node is generated (which should always fit ** within ROOT_MAX because it only needs space for 2 varints, the ** height and the blockid of the previous root). *** ** The meta-information in the segment directory is: ** level - segment level (see below) ** idx - index within level ** - (level, idx uniquely identify a segment) ** start_block - first leaf node ** leaves_end_block - last leaf node ** end_block - last block (including interior nodes) ** root - contents of root node ** ** If the root node is a leaf node, then start_block, ** leaves_end_block, and end_block are all 0. *** **** Segment merging **** ** To amortize update costs, segments are grouped into levels and ** merged in batches. Each increase in level represents exponentially ** more documents. *** ** New documents (actually, document updates) are tokenized and ** written individually (using LeafWriter) to a level 0 segment, with ** incrementing idx. When idx reaches MERGE_COUNT (default 16), all ** level 0 segments are merged into a single level 1 segment. Level 1 ** is populated like level 0, and eventually MERGE_COUNT level 1 ** segments are merged to a single level 2 segment (representing ** MERGE_COUNT*2 updates), and so on. ** ** A segment merge traverses all segments at a given level in ** parallel, performing a straightforward sorted merge. Since segment ** leaf nodes are written in to the %_segments table in order, this ** merge traverses the underlying sqlite disk structures efficiently. ** After the merge, all segment blocks from the merged level are ** deleted. ** ** MERGE_COUNT controls how often we merge segments. 16 seems to be ** somewhat of a sweet spot for insertion performance. 32 and 64 show ** very similar performance numbers to 16 on insertion, though they're ** a tiny bit slower (perhaps due to more overhead in merge-time ** sorting). 8 is about 20% slower than 16, 4 about 50% slower than ** 16, 2 about 66% slower than 16. ** ** At query time, high MERGE_COUNT increases the number of segments ** which need to be scanned and merged. For instance, with 100k docs ** inserted: *** ** MERGE_COUNT segments ** 16 25 ** 8 12 ** 4 10 ** 2 6 ** ** This appears to have only a moderate impact on queries for

very ** frequent terms (which are somewhat dominated by segment merge ** costs), and infrequent and non-existent terms still seem to be fast ** even with many segments. *** TODO(shess) That said, it would be nice to have a better query-side ** argument for MERGE_COUNT of 16. Also, it is possible/likely that ** optimizations to things like doclist merging will swing the sweet ** spot around. *** *** **** Handling of deletions and updates **** ** Since we're using a segmented structure, with no docid-oriented ** index into the term index, we clearly cannot simply update the term ** index when a document is deleted or updated. For deletions, we ** write an empty doclist (varint(docid) varint(POS_END)), for updates ** we simply write the new doclist. Segment merges overwrite older ** data for a particular docid with newer data, so deletes or updates ** will eventually overtake the earlier data and knock it out. The ** query logic likewise merges doclists so that newer data knocks out ** older data.

70560. Number of extra columns needed

70561. ** No-op routine for the parse-tree walker. ** ** When this routine is the Walker.xExprCallback then expression trees ** are walked without any actions being taken at each node. Presumably, ** when this routine is used for Walker.xExprCallback then ** Walker.xSelectCallback is set to do something useful for every ** subquery in the parser tree.

70562. The SQLite database connection

70563. Use the built-in recursive mutexes if they are available.

70564. ** Added for 3.5.8

70565. & result codes with this before returning

70566. Array of cells and sizes

70567. First instruction of interior of the loop

70568. Global (connection wide) data

70569. Pages in wal

70570. ** Turn a relative pathname into a full pathname. The relative path ** is stored as a nul-terminated string in the buffer pointed to by ** zPath. *** ** zOut points to a buffer of at least sqlite3_vfs.mxPathname bytes ** (in this case, MAX_PATHNAME bytes). The full-path is written to ** this buffer before returning.

70571. If nSeg is less than zero, then there is no level with at least ** nMin segments and no hint in the %_stat table. No work to do. ** Exit early in this case.

70572. ***** Begin file pager.c *****

70573. Number of compound terms processed so far

70574. If there are outstanding locks, do not actually close the file just ** yet because that would clear those locks. Instead, add the file ** descriptor to pInode->pUnused list. It will be automatically closed ** when the last lock is cleared.

70575. ** An rtree cursor object.

70576. When SQLITE_BUG_COMPATIBLE_20160819 is defined, unrecognized arguments ** to VACUUM are silently ignored. This is a back-out of a bug fix that ** occurred on 2016-08-19 (<https://www.sqlite.org/src/info/083f9e6270>). ** The buggy behavior is required for binary compatibility with some ** legacy applications.

70577. Before the first write, give the VFS a hint of what the final ** file size will be.

70578. ** SELECTTRACE_ENABLED will be either 1 or 0 depending on whether or not ** the Select query generator tracing logic is turned on.

70579. The cell to be inserted

70580. aSample[] for the upper bound

70581. One of PASSIVE, FULL and RESTART

70582. ** If an SQLITE_CONFIG_SQLLOG hook is registered and the VM has been run, ** invoke it.

70583. Opcode: IntegrityCk P1 P2 P3 P4 P5 *** Do an analysis of the currently open database. Store in ** register P1 the text of an error message describing any problems. ** If no problems are found, store a NULL in register P1. *** The register P3 contains one less than the maximum number of allowed errors. ** At most reg(P3) errors will be reported. ** In other words, the analysis stops as soon as reg(P1) errors are ** seen. Reg(P1) is updated with the number of errors remaining. *** The root page numbers of all tables in the database are integers ** stored in P4_INTARRAY argument. *** If P5 is not zero, the check is done on the auxiliary database ** file, not the main database file. *** This opcode is used to implement the integrity_check pragma.

70584. Buffer to append text to

70585. The new key

70586. Grow the Ft5Hash.aSlot[] array if necessary.

70587. If any BEFORE triggers were coded, then seek the cursor to the ** row to be deleted again. It may be that the BEFORE triggers moved ** the cursor or already deleted the row that the cursor was ** pointing to. *** Also disable the iIdxNoSeek optimization since the BEFORE trigger ** may have moved that cursor.

70588. Table into which text will be inserted

70589. Number of columns that show through to the output. ** Additional columns are used only as parameters to ** aggregate functions

70590. Block of regs holding rowid+data being inserted

70591. This table is not a VIEW

70592. Index of rbu_control column

70593. ** Write a 64-bit variable-length integer to memory starting at p[0]. ** The length of data written will be between 1 and FTS3_VARINT_MAX bytes. ** The number of bytes written is returned.

70594. ** Render a string given by "fmt" into the StrAccum object.

70595. Forward declarations

70596. Used by TK_AGG_COLUMN and TK_AGG_FUNCTION

70597. ifexists ::= IF EXISTS

70598. **comment:** Prevent unnecessary deep recursion when we run out of entries

label: code-design

70599. Set after incrementing the change-counter

70600. ** Implementation of the NULLIF(x,y) function. The result is the first ** argument if the arguments are different. The result is NULL if the ** arguments are equal to each other.

70601. 0x30..0x3F

70602. #include "vdbeInt.h"

70603. Return true if the argument is non-NULL and the WAL module is using ** heap-memory for the wal-index. Otherwise, if the argument is NULL or the ** WAL module is using shared-memory, return false.

70604. cmd ::= VACUUM nm

70605. Amount to increment (*pHash->pnByte) by

70606. "INSERT ... %_data VALUES(?,?)"

70607. Number of columns in the constructed index

70608. ** The first parameter is a pointer to an output buffer. The second ** parameter is a pointer to an integer that contains the offset at ** which to write into the output buffer. This function copies the ** nul-terminated string pointed to by the third parameter, zSignedIdent, ** to the specified offset in the buffer and updates *pldx to refer ** to the first byte after the last byte written before returning. *** If the string zSignedIdent consists entirely of alpha-numeric ** characters, does not begin with a digit and is not an SQL keyword, ** then it is copied to the output buffer exactly as it is. Otherwise, ** it is quoted using double-quotes.

70609. Database of the table

70610. Set the doNotSpill NOSYNC bit to 1. This is because we cannot allow ** a journal header to be written between the pages journaled by ** this function.

70611. ** Implementation of the json_mergepatch(JSON1,JSON2) function. Return a JSON ** object that is the result of running the RFC 7396 MergePatch() algorithm ** on the two arguments.

70612. column names. malloced

70613. Check to see the left operand is a column in a virtual table

70614. ** Context object passed to the fts5SentenceFinderCb() function.

70615. The roles of pLower and pUpper are swapped for a DESC index

70616. 2nd and subsequent times through the loop

70617. **comment:** OUT: sqlite3_malloc'd error message

label: code-design

70618. Currently this interface is only called by the OP_IfSmaller ** opcode, and in that case the cursor will always be valid and ** will always point to a leaf node.

70619. String defining the affinity of each column

70620. If this is a virtual table, invoke the xRename() function if ** one is defined. The xRename() callback will modify the names ** of any resources used by the v-table implementation (including other ** SQLite tables) that are identified by the name of the virtual table.

70621. An overflow has occurred

70622. ** Change the maximum size of any memory mapping made of the database file.

70623. Make sure page is marked dirty

70624. ** This is a variant of sqlite3PagerWrite() that runs when the sector size ** is larger than the page size. SQLite makes the (reasonable) assumption that ** all bytes of a sector are written together by hardware. Hence, all bytes of ** a sector need to be journaled in case of a power loss in the middle of ** a write. ** ** Usually, the sector size is less than or equal to the page size, in which ** case pages can be individually written. This routine only runs in the ** exceptional case where the page size is smaller than the sector size.

70625. 570

70626. **comment:** TODO(shess) Delimiters need to remain the same from run to run, ** else we need to reindex. One solution would be a meta-table to ** track such information in the database, then we'd only want this ** information on the initial create.
label: code-design

70627. If SQLITE_UTF16 is specified as the encoding type, transform this ** to one of SQLITE_UTF16LE or SQLITE_UTF16BE using the ** SQLITE_UTF16NATIVE macro. SQLITE_UTF16 is not used internally. ** ** If SQLITE_ANY is specified, add three versions of the function ** to the hash table.

70628. Check cell size

70629. The pager being played back

70630. Candidate for the LIKE optimization

70631. sclp ::=

70632. Saved key that was cursor last known position

70633. Table has keys only - no data

70634. Attempt to reduce exponent. ** ** Branches that are not required for the correct answer but which only ** help to obtain the correct answer faster are marked with special ** comments, as a hint to the mutation tester.

70635. OUT: True if iterator is at EOF

70636. xFinalize

70637. ** Return the offset of the sector boundary at or immediately ** following the value in pPager->journalOff, assuming a sector ** size of pPager->sectorSize bytes. ** ** i.e for a sector size of 512: ** ** Pager.journalOff Return value ** ----- ** 0 0 ** 512 512 ** 100 512 ** 2000 2048 **

70638. **comment:** Connection to notify of malloc failures
label: code-design

70639. ** Rerun the compilation of a statement after a schema change. ** ** If the statement is successfully recompiled, return SQLITE_OK. Otherwise, ** if the statement cannot be recompiled because another connection has ** locked the sqlite3_master table, return SQLITE_LOCKED. If any other error ** occurs, return SQLITE_SCHEMA.

70640. ** This routine frees the BLOB that is returned by geomCallback().

70641. ** Free a JsonParse object that was obtained from sqlite3_malloc().

70642. ** Maximum segments permitted in a single index

70643. ***** Begin file os.c *****

70644. Duplicates are scattered

70645. 0 -> copy from page, 1 -> copy to page

70646. ASCII range token characters

70647. True to omit locking primitives

70648. If nSize==0, then the %_segdir.end_block field does not not ** contain a size value. This happens if it was written by an ** old version of FTS. In this case it is not possible to determine ** the size of the segment, and so segment promotion does not ** take place.

70649. Number of bytes in aData

70650. VDBE under construction

70651. IMP: R-64894-50321 The string "?000" is returned if the argument ** is NULL or contains no ASCII alphabetic characters.

70652. CONFLICT => ID

70653. Check if this is a recursive CTE.

70654. Full-text index search

70655. ** There is no "best-index". This virtual table always does a linear ** scan. However, a schema=? constraint should cause this table to ** operate on a different database schema, so check for it. ** ** idxNum is normally 0, but will be 1 if a schema=? constraint exists.

70656. ** Advance the iterator object passed as the only argument. Return true ** if the iterator reaches EOF, or false otherwise.

70657. ** isNot: ** This variable is used by function getNextNode(). When getNextNode() is ** called, it sets ParseContext.isNot to true if the 'next node' is a ** FTSQUERY_PHRASE with a unary "-" attached to it. i.e. "mysql" in the ** FTS3 query "sqlite -mysql". Otherwise, ParseContext.isNot is set to ** zero.

70658. ** Move the iterator to the next entry at or following iMatch.

70659. zero to read. non-zero to write.

70660. SQLITE OMIT_SHARED_CACHE

70661. ** Merge all level iLevel segments in the database into a single ** iLevel+1 segment. Or, if iLevel<0, merge all segments into a ** single segment with a level equal to the numerically largest level ** currently present in the database. ** ** If this function is called with iLevel<0, but there is only one ** segment in the database, SQLITE_DONE is returned immediately. ** Otherwise, if successful, SQLITE_OK is returned. If an error occurs, ** an SQLite error code is returned.

70662. Maximum number of values to append

70663. The escape character string must consist of a single UTF-8 character. ** Otherwise, return an error.

70664. Sharable content of this btree

70665. All SQL text past the last semicolon parsed

70666. Map the requested memory region into this processes address space.

70667. **comment:** ** Attempt to apply the change that the iterator passed as the first argument ** currently points to to the database. If a conflict is encountered, invoke ** the conflict handler callback. ** ** If argument pbRetry is NULL, then ignore any CHANGESET_DATA conflict. If ** one is encountered, update or delete the row with the matching primary key ** instead. Or, if pbRetry is not NULL and a CHANGESET_DATA conflict occurs, ** invoke the conflict handler. If it returns CHANGESET_REPLACE, set *pbRetry ** to true before returning. In this case the caller will invoke this function ** again, this time with pbRetry set to NULL. ** ** If argument pbReplace is NULL and a CHANGESET_CONFLICT conflict is ** encountered invoke the conflict handler with CHANGESET_CONSTRAINT instead. ** Or, if pbReplace is not NULL, invoke it with CHANGESET_CONFLICT. If such ** an invocation returns SQLITE_CHANGESET_REPLACE, set *pbReplace to true ** before retrying. In this case the caller attempts to remove the conflicting ** row before invoking this function again, this time with pbReplace set ** to NULL. ** ** If any conflict handler returns SQLITE_CHANGESET_ABORT, this function ** returns SQLITE_ABORT. Otherwise, if no error occurs, SQLITE_OK is ** returned.
label: code-design

70668. Cursor number of the temporary table holding result

70669. Send an SQLITE_FCNTL_PRAGMA file-control to the underlying VFS ** connection. If it returns SQLITE_OK, then assume that the VFS ** handled the pragma and generate a no-op prepared statement. ** ** IMPLEMENTATION-OF: R-12238-55120 Whenever a PRAGMA statement is parsed, ** an SQLITE_FCNTL_PRAGMA file control is sent to the open sqlite3_file ** object corresponding to the database file to which the pragma ** statement refers. ** ** IMPLEMENTATION-OF: R-29875-31678 The argument to the SQLITE_FCNTL_PRAGMA ** file control is an array of pointers to strings (char**) in which the ** second element of the array is the name of the pragma and the third ** element is the argument to the pragma or NULL if the pragma has no ** argument.

70670. Number of bytes in buffer zRoot

70671. nArg of -2 is a special case

70672. Indicates a reserved lock has been obtained

70673. ***** Begin file func.c *****

70674. Number of values in record

70675. TUNING: If both iUpper and iLower are derived from the same ** sample, then assume they are 4x more selective. This brings ** the estimated selectivity more in line with what it would be ** if estimated without the use of STAT3/4 tables.

70676. Unlock the system-level locks
70677. Must be a compound subquery
70678. 158
70679. 101
70680. Variable iLower will be set to the estimate of the number of rows in ** the index that are less than the lower bound of the range query. The ** lower bound being the concatenation of \$P and \$L, where \$P is the ** key-prefix formed by the nEq values matched against the nEq left-most ** columns of the index, and \$L is the value in pLower. *** Or, if pLower is NULL or \$L cannot be extracted from it (because it ** is not a simple variable or literal value), the lower bound of the ** range is \$P. Due to a quirk in the way whereKeyStats() works, even ** if \$L is available, whereKeyStats() is called for both (\$P) and ** (\$P:\$L) and the larger of the two returned values is used. *** Similarly, iUpper is to be set to the estimate of the number of rows ** less than the upper bound of the range query. Where the upper bound ** is either (\$P) or (\$P:\$U). Again, even if \$U is available, both values ** of iUpper are requested of whereKeyStats() and the smaller used. ***
The number of rows between the two bounds is then just iUpper-iLower.

70681. pInitMutex
70682. ** A "Collating Sequence" is defined by an instance of the following ** structure. Conceptually, a collating sequence consists of a name and ** a comparison routine that defines the order of that sequence. *** If CollSeq.xCmp is NULL, it means that the ** collating sequence is undefined. Indices built on an undefined ** collating sequence may not be read or written.
70683. ** Return the total number of pages in the cache.
70684. ** Specify the key for an encrypted database. This routine should be ** called right after sqlite3_open(). *** The code to implement this API is not available in the public release ** of SQLite.
70685. **** Start of ascii tokenizer implementation.
70686. Checksum based on FTS index contents
70687. True if all changes are indirect
70688. Link pNew element into the hash table pH. If pEntry!=0 then also ** insert pNew into the pEntry hash bucket.
70689. #include "fts5.h"
70690. Name of column in child table
70691. **comment:** Make sure a corrupt database has not given us an oversize header. ** Do this now to avoid an oversize memory allocation. *** Type entries can be between 1 and 5 bytes each. But 4 and 5 byte ** types use so much data space that there can only be 4096 and 32 of ** them, respectively. So the maximum header length results from a ** 3-byte type for each of the maximum of 32768 columns plus three ** extra bytes for the header length itself. $32768 \times 3 + 3 = 98307$.
label: code-design
70692. ***** Begin file btree.c *****
70693. Memory register holding the rowid counter
70694. If the file was opened successfully, read the schema for the new database. ** If this fails, or if opening the file failed, then close the file and ** remove the entry from the db->aDb[] array. i.e. put everything back the way ** we found it.
70695. 24
70696. "neq" column of stat[34] entry
70697. Total number of bytes in the record
70698. OUT: Permissions to open file with
70699. It is safe to always return here, but the resulting tree ** would be unbalanced
70700. Iterator for this term
70701. The rtree may have between 1 and RTREE_MAX_DIMENSIONS dimensions.
70702. Load doclist-index for this leaf
70703. Memory cell used to store aggregate context
70704. Last character before the first wildcard
70705. ** Argument pTerm must be a synonym iterator.
70706. ** Wrapper around the pluggable caches xUnpin method. If the cache is ** being used for an in-memory database, this function is a no-op.
70707. Value returned by SetFilePointer()
70708. ** Valid values for the second argument to fts3SqlStmt().
70709. If this is an RBU vacuum operation and the state table was empty ** when this handle was opened, create the target database schema.
70710. ** Return the number of phrases in expression pExpr.
70711. ** The MergeEngine object is used to combine two or more smaller PMAs into ** one big PMA using a merge operation. Separate PMAs all need to be ** combined into one big PMA in order to be able to step through the sorted ** records in order. *** The aReadr[] array contains a PmaReader object for each of the PMAs being ** merged. An aReadr[] object either points to a valid key or else is at EOF. ** ("EOF" means "End Of File". When aReadr[] is at EOF there is no more data.) ** For the purposes of the paragraphs below, we assume that the array is ** actually N elements in size, where N is the smallest power of 2 greater ** to or equal to the number of PMAs being merged. The extra aReadr[] elements ** are treated as if they are empty (always at EOF). *** The aTree[] array is also N elements in size. The value of N is stored in ** the MergeEngine.nTree variable. *** The final (N/2) elements of aTree[] contain the results of comparing ** pairs of PMA keys together. Element i contains the result of ** comparing aReadr[2*i-N] and aReadr[2*i-N+1]. Whichever key is smaller, the ** aTree element is set to the index of it. *** For the purposes of this comparison, EOF is considered greater than any ** other key value. If the keys are equal (only possible with two EOF ** values), it doesn't matter which index is stored. *** The (N/4) elements of aTree[] that precede the final (N/2) described ** above contains the index of the smallest of each block of 4 PmaReaders. ** And so on. So that aTree[1] contains the index of the PmaReader that ** currently points to the smallest key value. aTree[0] is unused. *** Example: *** aReadr[0] -> Banana ** aReadr[1] -> Feijo ** aReadr[2] -> Elderberry ** aReadr[3] -> Currant ** aReadr[4] -> Grapefruit ** aReadr[5] -> Apple ** aReadr[6] -> Durian ** aReadr[7] -> EOF *** aTree[] = { X, 0, 0, 5, 0, 3, 5, 6 } *** The current element is "Apple" (the value of the key indicated by ** PmaReader 5). When the Next() operation is invoked, PmaReader 5 will ** be advanced to the next key in its segment. Say the next key is ** "Eggplant": *** aReadr[5] -> Eggplant *** The contents of aTree[] are updated first by comparing the new PmaReader ** 5 key to the current key of PmaReader 4 (still "Grapefruit"). The PmaReader ** 5 value is still smaller, so aTree[6] is set to 5. And so on up the tree. ** The value of PmaReader 6 - "Durian" - is now smaller than that of PmaReader ** 5, so aTree[3] is set to 6. Key 0 is smaller than key 6 (Banana < Durian), ** so the value written into element 1 of the array is 0. As follows: *** aTree[] = { X, 0, 0, 6, 0, 3, 5, 6 } *** In other words, each time we advance to the next sorter element, log2(N) ** key comparison operations are required, where N is the number of segments ** being merged (rounded up to the next power of 2).
70712. ** This routine runs an extensive test of the Bitvec code. *** The input is an array of integers that acts as a program ** to test the Bitvec. The integers are opcodes followed ** by 0, 1, or 3 operands, depending on the opcode. Another ** opcode follows immediately after the last operand. *** There are 6 opcodes numbered from 0 through 5. 0 is the ** "halt" opcode and causes the test to end. *** 0 Halt and return the number of errors ** 1 N S X Set N bits beginning with S and incrementing by X ** 2 N S X Clear N bits beginning with S and incrementing by X ** 3 N Set N randomly chosen bits ** 4 N Clear N randomly chosen bits ** 5 N S X Set N bits from S increment X in array only, not in bitvec ** ** The opcodes 1 through 4 perform set and clear operations are performed ** on both a Bitvec object and on a linear array of bits obtained from malloc. ** Opcode 5 works on the linear array only, not on the Bitvec. ** Opcode 5 is used to deliberately induce a fault in order to ** confirm that error detection works. *** At the conclusion of the test the linear array is compared ** against the Bitvec object. If there are any differences, ** an error is returned. If they are the same, zero is returned. *** If a memory allocation error occurs, return -1.
70713. Content for INT, REAL, and STRING
70714. ***** Begin file tokenize.c *****
70715. term
70716. ** Set the LIKEOPT flag on the 2-argument function with the given name.
70717. If we reach this point, it means that the database connection has ** closed all sqlite3_stmt and sqlite3_backup objects and has been ** passed to sqlite3_close (meaning that it is a zombie). Therefore, ** go ahead and free all resources.
70718. Bytes of available memory
70719. nRowLogEst set from sqlite_stat1
70720. Maximum first byte of cell for a 1-byte payload
70721. 233
70722. ** Return values of columns for the row at which the stmt_cursor ** is currently pointing.
70723. OUT: Trbul number of columns
70724. The size of a single frame
70725. Add opcodes to the prepared statement

70726. ** Do an authorization check using the code and arguments given. Return ** either SQLITE_OK (zero) or SQLITE_IGNORE or SQLITE_DENY. If SQLITE_DENY ** is returned, then the error count and error message in pParse are ** modified appropriately.

70727. Enter the mutexes

70728. Most sig. 32 bits of new offset

70729. Evaluate the equality constraints

70730. True to accumulate patches

70731. ** Flush any data cached by the writer object to the database. Free any ** allocations associated with the writer.

70732. ** Obtain a changeset object containing all changes recorded by the ** session object passed as the first argument. *** It is the responsibility of the caller to eventually free the buffer ** using sqlite3_free().

70733. Index of new.* value to retrieve

70734. Value returned by system call

70735. DISTINCT using indexes

70736. ** Return true (non-zero) if the input is an integer that is too large ** to fit in 32-bits. This macro is used inside of various testcase() ** macros to verify that we have tested SQLite for large-file support.

70737. Check to see if we need to create an sqlite_sequence table for ** keeping track of autoincrement keys.

70738. No candidate table+column was found. This can only occur ** on the second iteration

70739. ** Make sure the Tcl calling convention macro is defined. This macro is ** only used by test code and Tcl integration code.

70740. ** PRAGMA [schema.]lock_proxy_file ** PRAGMA [schema.]lock_proxy_file = ":auto:"|"lock_file_path" *** Return or set the value of the lock_proxy_file flag. Changing ** the value sets a specific file to be used for database access locks. **

70741. ** A read-lock must be held on the pager when this function is called. If ** the pager is in WAL mode and the WAL file currently contains one or more ** frames, return the size in bytes of the page images stored within the ** WAL frames. Otherwise, if this is not a WAL database or the WAL file ** is empty, return 0.

70742. Opcode: Yield P1 P2 * * * * Swap the program counter with the value in register P1. This ** has the effect of yielding to a coroutine. *** If the coroutine that is launched by this instruction ends with ** Yield or Return then continue to the next instruction. But if ** the coroutine launched by this instruction ends with ** EndCoroutine, then jump to P2 rather than continuing with the ** next instruction. *** See also: InitCoroutine

70743. Size of BTM vector

70744. ** Extract the next token from a tokenization cursor. The cursor must ** have been opened by a prior call to simpleOpen().

70745. Split operator. TK_AND or TK_OR

70746. Result of last comparison

70747. Address of the select-B coroutine

70748. The rowid we are to seek to

70749. Thread holding this mutex

70750. At this point this is probably a keyword. But for that to be true, ** the next byte must contain either whitespace, an open or close ** parenthesis, a quote character, or EOF.

70751. ** Enlarge the memory allocation on a StrAccum object so that it is ** able to accept at least N more bytes of text. *** Return the number of bytes of text that StrAccum is able to accept ** after the attempted enlargement. The value returned might be zero.

70752. There is now an entry for pExpr in pAggInfo->aCol[] (either ** because it was there before or because we just created it). ** Convert the pExpr to be a TK_AGG_COLUMN referring to that ** pAggInfo->aCol[] entry.

70753. ** This function is called by the VDBE to adjust the internal schema ** used by SQLite when the btree layer moves a table root page. The ** root-page of a table or index in database iDb has changed from iFrom ** to iTo. *** Ticket #1728: The symbol table might still contain information ** on tables and/or indices that are the process of being deleted. ** If you are unlucky, one of those deleted indices or tables might ** have the same rootpage number as the real table or index that is ** being moved. So we cannot stop searching after the first match ** because the first match might be for one of the deleted indices ** or tables and not the table/index that is actually being moved. ** We must continue looping until all tables and indices with ** rootpage==iFrom have been converted to have a rootpage of iTo ** in order to be certain that we got the right one.

70754. ** Read a single page from either the journal file (if isMainJrn==1) or ** from the sub-journal (if isMainJrn==0) and playback that page. ** The page begins at offset *pOffset into the file. The *pOffset ** value is increased to the start of the next page in the journal. *** The main rollback journal uses checksums - the statement journal does ** not. *** If the page number of the page record read from the (sub-)journal file ** is greater than the current value of Pager.dbSize, then playback is ** skipped and SQLITE_OK is returned. *** If pDone is not NULL, then it is a record of pages that have already ** been played back. If the page at *pOffset has already been played back ** (if the corresponding pDone bit is set) then skip the playback. ** Make sure the pDone bit corresponding to the *pOffset page is set ** prior to returning. *** If the page record is successfully read from the (sub-)journal file ** and played back, then SQLITE_OK is returned. If an IO error occurs ** while reading the record from the (sub-)journal file or while writing ** to the database file, then the IO error code is returned. If data ** is successfully read from the (sub-)journal file but appears to be ** corrupted, SQLITE_DONE is returned. Data is considered corrupted in ** two circumstances: ** *** * If the record page-number is illegal (0 or PAGER_MJ_PGNO), or ** * If the record is being rolled back from the main journal file ** and the checksum field does not match the record content. *** Neither of these two scenarios are possible during a savepoint rollback. *** If this is a savepoint rollback, then memory may have to be dynamically ** allocated by this function. If this is the case and an allocation fails, ** SQLITE_NOMEM is returned.

70755. ***** Begin control #defines *****

70756. ** Two variations on the public interface for closing a database ** connection. The sqlite3_close() version returns SQLITE_BUSY and ** leaves the connection option if there are unfinalized prepared ** statements or unfinished sqlite3_backups. The sqlite3_close_v2() ** version forces the connection to become a zombie if there are ** unclosed resources, and arranges for deallocation when the last ** prepare statement or sqlite3_backup closes.

70757. ** Rtree virtual table module xCreate method.

70758. Get the results of the thread

70759. A-overwrites-T

70760. Next outer name context. NULL for outermost

70761. All cells begin balanced

70762. ** A lists of all unixInodeInfo objects.

70763. All mutexes are required for schema access. Make sure we hold them.

70764. If we are doing a reverse order scan on an ascending index, or ** a forward order scan on a descending index, interchange the ** start and end terms (pRangeStart and pRangeEnd).

70765. Page to delete cell from

70766. Page number for first page in the list

70767. Number of columns of data

70768. ** Only one of SQLITE_ENABLE_STAT3 or SQLITE_ENABLE_STAT4 can be defined. ** Priority is given to SQLITE_ENABLE_STAT4. If either are defined, also ** define SQLITE_ENABLE_STAT3_OR_STAT4

70769. Linked list of all foreign keys in this table

70770. ** pOrderBy is an ORDER BY or GROUP BY clause in SELECT statement pSelect. ** The Name context of the SELECT statement is pNC. zType is either ** "ORDER" or "GROUP" depending on which type of clause pOrderBy is. *** This routine resolves each term of the clause into an expression. ** If the order-by term is an integer I between 1 and N (where N is the ** number of columns in the result set of the SELECT) then the expression ** in the resolution is a copy of the I-th result-set expression. If ** the order-by term is an identifier that corresponds to the AS-name of ** a result-set expression, then the term resolves to a copy of the ** result-set expression. Otherwise, the expression is resolved in ** the usual way - using sqlite3ResolveExprNames(). *** This routine returns the number of errors. If errors occur, then ** an appropriate error message might be left in pParse. (OOM errors ** excepted.)

70771. ** Add code that will check to make sure the N registers starting at iMem ** form a distinct entry. iTab is a sorting index that holds previously ** seen combinations of the N values. A new entry is made in iTab ** if the current N values are new. *** A jump to addrRepeat is made and the N+1 values are popped from the ** stack if the top N elements are not distinct.

70772. Number of memory cells in the row key

70773. Profiling function

70774. ccons ::= COLLATE ID|STRING

70775. ** Move the iterator passed as the first argument to the next term in the ** segment. If successful, SQLITE_OK is returned. If there is no next term, ** SQLITE_DONE. Otherwise, an SQLite error code.

70776. Used to iterate through aTask[]

70777. ** Read the content for page pPg out of the database file and into ** pPg->pData. A shared lock or greater must be held on the database ** file before this function is called. *** If page 1 is read, then the value of Pager.dbFileVers[] is set to ** the value read from the database file. *** If an IO error occurs, then the IO error is returned to the caller. ** Otherwise, SQLITE_OK is returned.

70778. Bucket of terms object used by the integrity-check in offsets=0 mode.

70779. ** Return the number of the variable named zName, if it is in VList. ** or return 0 if there is no such variable.

70780. RHS is real

70781. ** Convert an integer into a LogEst. In other words, compute an ** approximation for $10^{\log_2(x)}$.

70782. True if a journal file is present

70783. Methods for an open file

70784. expr ::= RAISE LP IGNORE RP

70785. Index of current page in apPage

70786. ** If this is a WAL database, obtain a snapshot handle for the snapshot ** currently open. Otherwise, return an error.

70787. Form 1: Analyze everything

70788. Set stack variable q to the close-quote character

70789. Array of active savepoints

70790. True if the previous character was uEsc

70791. ** Set the sort order for the last element on the given ExprList.

70792. 380

70793. ** Indicate that registers between iReg..iReg+nReg-1 are being overwritten. ** Purge the range of registers from the column cache.

70794. Rows processed for current object

70795. when_clause ::=

70796. id value

70797. Index to read column names from

70798. tridxby ::= INDEXED BY nm

70799. ** Minimum chunk size used by streaming versions of functions.

70800. OUTPUT: True if NOT NULL constraint exists

70801. **comment:** This is a helper function for sqlite3BtreeLock(). By moving ** complex, but seldom used logic, out of sqlite3BtreeLock() and ** into this routine, we avoid unnecessary stack pointer changes ** and thus help the sqlite3BtreeLock() routine to run much faster ** in the common case.
label: code-design

70802. Number of memory locations currently allocated

70803. Initialize a lookahead iterator for each phrase. After passing the ** buffer and buffer size to the lookaside-reader init function, zero ** the phrase poslist buffer. The new poslist for the phrase (containing ** the same entries as the original with some entries removed on account ** of the NEAR constraint) is written over the original even as it is ** being read. This is safe as the entries for the new poslist are a ** subset of the old, so it is not possible for data yet to be read to ** be overwritten.

70804. List of frame objects to free on VM reset

70805. Output poslist

70806. Run the BEFORE and INSTEAD OF triggers, if there are any

70807. Number of valid entries in array aNew[]

70808. The name of the trigger

70809. ** Discard all data currently cached in the hash-tables.

70810. OUT: Visibility

70811. Used to iterate through constraint array

70812. Arguments to the format string

70813. Trace function

70814. ** Close a tokenization cursor previously opened by a call to icuOpen().

70815. sqlite3GetVdbe cannot fail: VDBE already allocated

70816. Number of usable bytes on each page

70817. ** Query the hash table for a doclist associated with term pTerm/nTerm.

70818. Free up the array of auxiliary databases

70819. ** Flush all unreferenced dirty pages to disk.

70820. OUT: Set to true if really a match

70821. Input JSON text to be parsed

70822. Next with a different name but the same hash

70823. Set nByte to the number of bytes required to store the expanded blob.

70824. True if a warning has been issued

70825. **comment:** G: New-style sqlite3_rtree_query_callback()
label: code-design

70826. ** For chunk size nChunkSize, return the number of bytes that should ** be allocated for each FileChunk structure.

70827. Opcode: CreateIndex P1 P2 * * * * Synopsis: [P2]=root iDb=P1 *** Allocate a new index in the main database file if P1==0 or in the ** auxiliary database file if P1==1 or in an attached database if ** P1>1. Write the root page number of the new table into ** register P2. *** See documentation on OP_CreateTable for additional information.

70828. Allocate a buffer in which to accumulate data

70829. All other Windows platforms expect GetProcAddress() to take ** an ANSI string regardless of the _UNICODE setting

70830. Real value used when MEM_Real is set in flags

70831. SQLITE_OMIT_HEX_INTEGER

70832. ** The journal file must be open when this routine is called. A journal ** header (JOURNAL_HDR_SZ bytes) is written into the journal file at the ** current location. *** The format for the journal header is as follows: ** - 8 bytes: Magic identifying journal format. ** - 4 bytes: Number of records in journal, or -1 no sync mode is on. ** - 4 bytes: Random number used for page hash. ** - 4 bytes: Initial database page count. ** - 4 bytes: Sector size used by the process that wrote this journal. ** - 4 bytes: Database page size. *** Followed by (JOURNAL_HDR_SZ - 28) bytes of unused space.

70833. This vtab delivers always results in "ORDER BY term ASC" order.

70834. EVIDENCE-OF: R-18143-12121 Value is the integer 1.

70835. the hash table

70836. Bytes to read from disk

70837. sqlite3_user_data() context

70838. Begin by rolling back records from the main journal starting at ** PagerSavepoint.iOffset and continuing to the next journal header. ** There might be records in the main journal that have a page number ** greater than the current database size (pPager->dbSize) but those ** will be skipped automatically. Pages are added to pDone as they ** are played back.

70839. **comment:** Allocated size of a[] (nAlloc>=n)
label: code-design

70840. ** Return non-zero if a transaction is active.

70841. Type of file to open

70842. <0 means "need to query"

70843. aKWNNext[] forms the hash collision chain. If aKWHHash[i]==0 ** then the i-th keyword has no more hash collisions. Otherwise, ** the next keyword with the same hash is aKWHHash[i]-1.

70844. Currently always set to 3

70845. Write the new column list here

70846. ** Convert an sqlite3_value into an RtreeValue (presumably a float) ** while taking care to round toward negative or positive, respectively.

70847. End of {...}

70848. Use this case if we have ANSI headers
70849. **comment:** ** The database page the PENDING_BYTE occupies. This page is never used.
 label: code-design
70850. Extension to which this belongs
70851. Split the WHERE clause into separate subexpressions where each ** subexpression is separated by an AND operator.
70852. If this is a release of the outermost savepoint, truncate ** the sub-journal to zero bytes in size.
70853. The page number of a page in journal
70854. Reinitialize page pTo so that the contents of the MemPage structure ** match the new data. The initialization of pTo can actually fail under ** fairly obscure circumstances, even though it is a copy of initialized ** page pFrom.
70855. ICU library collation object
70856. cmd ::= with DELETE FROM fullname indexed_opt where_opt
70857. Configure the text that the regular expression operates on.
70858. The VFS that created this unixFile
70859. ** Return a copy of input string zInput enclosed in double-quotes ("") and ** with all double quote characters escaped. For example: *** *** fts3QuoteId("un \"zip\"") -> "un \"\"zip\"\""
 *** The pointer returned points to memory obtained from sqlite3_malloc(). It ** is the callers responsibility to call sqlite3_free() to release this ** memory.
70860. defined(SQLITE_ENABLE_ICU)
70861. Next with the same zTo. Next child of zTo.
70862. Create other tables
70863. Size of pList as PMA in bytes
70864. ** Read a 64-bit big-endian integer value from buffer aRec[]. Return ** the value read.
70865. **comment:** It's odd to simulate an io-error here, but really this is just ** using the io-error infrastructure to test that SQLite handles this ** function failing.
 label: code-design
70866. First part of index name. May be NULL
70867. **comment:** ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ***
 ***** Utility functions used throughout sqlite. *** This file contains functions for allocating memory, comparing ** strings, and stuff like that. **
 label: code-design
70868. Register to hold Stat4Accum object
70869. Root page of b-tree
70870. The cursor to compare against
70871. **comment:** Opcode: DeferredSeek P1 * P3 P4 * ** Synopsis: Move P3 to P1.rowid if needed *** P1 is an open index cursor and P3 is a cursor on the corresponding ** table. This opcode does a deferred seek of the P3 table cursor ** to the row that corresponds to the current row of P1. *** This is a deferred seek. Nothing actually happens until ** the cursor is used to read a record. That way, if no reads ** occur, no unnecessary I/O happens. *** P4 may be an array of integers (type P4_INTARRAY) containing ** one entry for each column in the P3 table. If array entry a(i) ** is non-zero, then reading column a(i)-1 from cursor P3 is ** equivalent to performing the deferred seek and then reading column i ** from P1. This information is stored in P3 and used to redirect ** reads against P3 over to P1, thus possibly avoiding the need to ** seek and read cursor P3.
 label: code-design
70872. True -> read/write access, false -> read-only
70873. iVersion
70874. Generate code to destroy the database record of the trigger.
70875. Use STAT3 or STAT4 data
70876. Get the precision
70877. If at least one field has been modified, this is not a no-op.
70878. If table pTab has not been used in a way that would benefit from ** having analysis statistics during the current session, then skip it. ** This also has the effect of skipping virtual tables and views
70879. **comment:** It's odd to simulate an io-error here, but really this is just ** using the io-error infrastructure to test that SQLite handles this ** function failing.
 label: code-design
70880. OUT: Size of output buffer in bytes
70881. ** Open and return a new RBU handle.
70882. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, append a string to the buffer. All bytes in the string ** up to (but not including) the nul-terminator are written to the buffer. *** If an OOM condition is encountered, set *pRc to SQLITE_NOMEM before ** returning.
70883. This trick assumes that both the page-size and sector-size are ** an integer power of 2. It sets variable pg1 to the identifier ** of the first page of the sector pPg is located on.
70884. ** PRAGMA [schema.]secure_delete ** PRAGMA [schema.]secure_delete=ON/OFF/FAST *** The first form reports the current setting for the ** secure_delete flag. The second form changes the secure_delete ** flag setting and reports the new value.
70885. 960
70886. ** pEList is a list of expressions which are really the result set of the ** a SELECT statement. pE is a term in an ORDER BY or GROUP BY clause. ** This routine checks to see if pE is a simple identifier which corresponds ** to the AS-name of one of the terms of the expression list. If it is, ** this routine return an integer between 1 and N where N is the number of ** elements in pEList, corresponding to the matching entry. If there is ** no match, or if pE is not a simple identifier, then this routine ** return 0. *** pEList has been resolved. pE has not.
70887. Check if data must be read/written to/from the btree page itself.
70888. Next entry in Parse.pTriggerPrg list
70889. ** For MinGW version 4.x (and higher), check to see if the _USE_32BIT_TIME_T ** define is required to maintain binary compatibility with the MSVC runtime ** library in use (e.g. for Windows XP).
70890. ** Clear all column cache entries.
70891. 0x02
70892. Save profiling information from this VDBE run.
70893. Column to extract from poslist
70894. OUT: Docid for row just inserted
70895. The VFS used to open this file
70896. IMP: R-29431-39229
70897. Text encoding handled by xCmp()
70898. ** The ISAUTOVACUUM macro is used within balance_nonroot() to determine ** if the database supports auto-vacuum or not. Because it is used ** within an expression that is an argument to another macro ** (sqliteMallocRaw), it is not possible to use conditional compilation. ** So, this macro is defined instead.
70899. pPage->aData
70900. ** Each SQL function is defined by an instance of the following ** structure. For global built-in functions (ex: substr(), max(), count()) ** a pointer to this structure is held in the sqlite3BuiltinFunctions object. ** For per-connection application-defined functions, a pointer to this ** structure is held in the db->aHash hash table. *** The u.pHash field is used by the global built-ins. The u.pDestructor ** field is used by per-connection app-def functions.
70901. zKWText[] encodes 834 bytes of keyword text in 554 bytes
70902. Return value
70903. If this is an RBU vacuum operation and this is the target database, ** pretend that it has at least one page. Otherwise, SQLite will not ** check for the existance of a *.wal file. rbuVfsRead() contains ** similar logic.
70904. Always cast the getpid() return type for compatibility with ** kernel modules in VxWorks.
70905. Existing index ASCENDING
70906. Array of nLevel level objects
70907. The program counter

70908. Create a snapshot object. The content of a snapshot is opaque to ** every other subsystem, so the WAL module can put whatever it needs ** in the object.
70909. Pointer to the rbu_vfs object
70910. Function name
70911. ** The next group of routines are convenience wrappers around the ** VFS methods.
70912. ** An implementation of a min-heap. ** ** aHeap[0] is the number of elements on the heap. aHeap[1] is the ** root element. The daughter nodes of aHeap[N] are aHeap[N*2] ** and aHeap[N*2+1]. ** ** The heap property is this: Every node is less than or equal to both ** of its daughter nodes. A consequence of the heap property is that the ** root node aHeap[1] is always the minimum value currently in the heap. ** ** The btreeHeapInsert() routine inserts an unsigned 32-bit number onto ** the heap, preserving the heap property. The btreeHeapPull() routine ** removes the root element from the heap (the minimum value in the heap) ** and then moves other nodes around as necessary to preserve the heap ** property. ** ** This heap is used for cell overlap and coverage testing. Each u32 ** entry represents the span of a cell or freeblock on a btree page. ** The upper 16 bits are the index of the first byte of a range and the ** lower 16 bits are the index of the last byte of that range.
70913. First operand
70914. **comment:** ** FTS4 is really an extension for FTS3. It is enabled using the ** SQLITE_ENABLE_FTS3 macro. But to avoid confusion we also call ** the SQLITE_ENABLE_FTS4 macro to serve as an alias for SQLITE_ENABLE_FTS3.
label: code-design
70915. The column value supplied by SQLite must be in range.
70916. Next divider slot in pParent->aCell[]
70917. Skip over any TK_COLLATE nodes
70918. IMP: R-26507-47431
70919. Next backup associated with source pager
70920. ** Insert the contents of cell pCell into node pNode. If the insert ** is successful, return SQLITE_OK. ** ** If there is not enough free space in pNode, return SQLITE_FULL.
70921. ** Return the name of the i-th column of the pIdx index.
70922. ** The pMem is known to contain content that needs to be destroyed prior ** to a value change. So invoke the destructor, then set the value to ** a 64-bit integer.
70923. The file I/O methods to use.
70924. Pointer to one byte after EOF
70925. For looping over all elements of the table
70926. Bitvec of pages already played back
70927. Delete existing record, then do INSERT or UPDATE
70928. ***** End of complete.c *****
70929. This block deals with an obscure problem. If the last connection ** that wrote to this database was operating in persistent-journal ** mode, then the journal file may at this point actually be larger ** than Pager.journalOff bytes. If the next thing in the journal ** file happens to be a journal-header (written as part of the ** previous connection's transaction), and a crash or power-failure ** occurs after nRec is updated but before this connection writes ** anything else to the journal file (or commits/rolls back its ** transaction), then SQLite may become confused when doing the ** hot-journal rollback following recovery. It may roll back all ** of this connection's data, then proceed to rolling back the old, ** out-of-date data that follows it. Database corruption. ** ** To work around this, if the journal file does appear to contain ** a valid header following Pager.journalOff, then write a 0x00 ** byte to the start of it to prevent it from being recognized. ** ** Variable iNextHdrOffset is set to the offset at which this ** problematic header will occur, if it exists. aMagic is used ** as a temporary buffer to inspect the first couple of bytes of ** the potential journal header.
70930. Alternatively, if bExtend is true, extend the file. Do this by ** writing a single byte to the end of each (OS) page being ** allocated or extended. Technically, we need only write to the ** last page in order to extend the file. But writing to all new ** pages forces the OS to allocate them immediately, which reduces ** the chances of SIGBUS while accessing the mapped region later on.
70931. "0" digits after the decimal point but before the first ** significant digit of the number
70932. ** Include code that is common to all os_*.c files
70933. Used to iterate through segment-readers
70934. BLOB returned by geometry function
70935. on_opt
70936. ** Return true if pNew is to be preferred over pOld. ** ** This function assumes that for each argument sample, the contents of ** the anEq[] array from pSample->anEq[pSample->iCol] onwards are valid.
70937. **comment:** Term iterator to advance
label: code-design
70938. Cursor used to iterate through aDoclist
70939. Incrementally write one PMA
70940. Check that the internal nodes of each segment match the leaves
70941. ** This function is called after an "ALTER TABLE ... ADD" statement ** has been parsed. Argument pColDef contains the text of the new ** column definition. ** ** The Table structure pParse->pNewTable was extended to include ** the new column during parsing.
70942. Allocate temporary working space.
70943. synopsis: nColumn=P2
70944. 'autmerge' setting
70945. Register to copy from
70946. ** Resolve all names in all expressions of a SELECT and in all ** descendants of the SELECT, including compounds off of p->pPrior, ** subqueries in expressions, and subqueries used as FROM clause ** terms. ** ** See sqlite3ResolveExprNames() for a description of the kinds of ** transformations that occur. ** ** All SELECT statements should have been expanded using ** sqlite3SelectExpand() prior to invoking this routine.
70947. Cascade the changes
70948. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, append a single byte to the buffer. ** ** If an OOM condition is encountered, set *pRc to SQLITE_NOMEM before ** returning.
70949. Make the page containing the entry to be deleted writable. Then free any ** overflow pages associated with the entry and finally remove the cell ** itself from within the page.
70950. Term to iterate through a doclist for
70951. Create the virtual table wrapper around the hash-table and overload ** the four scalar functions. If this is successful, register the ** module with sqlite.
70952. Value of each argument
70953. ***** Begin file fts3_snippet.c *****
70954. 0: Acquire a read or write lock
70955. The child table to be scanned
70956. Individual characters in zCharSet
70957. ** Read data from a file into a buffer. Return SQLITE_OK if all ** bytes were read successfully and SQLITE_IOERR if anything goes ** wrong.
70958. Check if pExpr is identical to any GROUP BY term. If so, consider ** it constant.
70959. 1260
70960. Drop the table and index from the internal schema.
70961. aOffset[i] is offset to start of data for i-th column
70962. ***** Interface to code in fts5.c.
70963. Opcode: Transaction P1 P2 P3 P4 P5 ** ** Begin a transaction on database P1 if a transaction is not already ** active. ** If P2 is non-zero, then a write-transaction is started, or if a ** read-transaction is already active, it is upgraded to a write-transaction. ** If P2 is zero, then a read-transaction is started. ** ** P1 is the index of the database file on which the transaction is ** started. Index 0 is the main database file and index 1 is the ** file used for temporary tables. Indices of 2 or more are used for ** attached databases. ** ** If a write-transaction is started and the Vdbe.usesModuleJournal flag is ** true (this flag is set if the Vdbe may modify more than one row and may ** throw an ABORT exception), a statement transaction may also be opened. ** More specifically, a statement transaction is opened iff the database ** connection is currently not in autocommit mode, or if there are other ** active statements. A statement transaction allows the changes made by this ** VDBE to be rolled back after an error without having to roll back the ** entire transaction. If no error is encountered, the statement transaction ** will automatically commit when the VDBE halts. ** ** If P5!=0 then this opcode also checks the schema cookie against P3 ** and the schema generation counter

against P4. ** The cookie changes its value whenever the database schema changes. ** This operation is used to detect when that the cookie has changed ** and that the current process needs to reread the schema. If the schema ** cookie in P3 differs from the schema cookie in the database header or ** if the schema generation counter in P4 differs from the current ** generation counter, then an SQLITE_SCHEMA error is raised and execution ** halts. The sqlite3_step() wrapper function might then reprepare the ** statement and rerun it from the beginning.

70964. What to do with a uniqueness conflict

70965. ** While a SrcList can in general represent multiple tables and subqueries ** (as in the FROM clause of a SELECT statement) in this case it contains ** the name of a single table, as one might find in an INSERT, DELETE, ** or UPDATE statement. Look up that table in the symbol table and ** return a pointer. Set an error message and return NULL if the table ** name is not found or if any other error occurs. ** ** The following fields are initialized appropriate in pSrc: ** ** pSrc->a[0].pTab Pointer to the Table object ** pSrc->a[0].pIndex Pointer to the INDEXED BY index, if there is one **

70966. Temporary buffer space

70967. ** Create an RBU VFS named zName that accesses the underlying file-system ** via existing VFS zParent. The new object is registered as a non-default ** VFS with SQLite before returning.

70968. If this process is running as root, make sure that the SHM file ** is owned by the same user that owns the original database. Otherwise, ** the original owner will not be able to connect.

70969. ** Arguments aldx, aCell and aSpare all point to arrays of size ** nIdx. The aldx array contains the set of integers from 0 to ** (nIdx-1) in no particular order. This function sorts the values ** in aldx according to dimension iDim of the cells in aCell. The ** minimum value of dimension iDim is considered first, the ** maximum used to break ties. ** ** The aSpare array is used as temporary working space by the ** sorting algorithm.

70970. Closing highlight

70971. ** Default maximum size for persistent journal files. A negative ** value means no limit. This value may be overridden using the ** sqlite3PagerJournalSizeLimit() API. See also "PRAGMA journal_size_limit".

70972. If p2 is at EOF

70973. Terms that make up this phrase

70974. Value is a real number

70975. (284) columnlist ::= columnlist COMMA columnname carglist

70976. One of the RTREE_COORD_* constants

70977. Used to test for reference loops

70978. Set the szLeaf header field.

70979. ** CAPI3REF: String LIKE Matching * ** ^The [sqlite3_strlike(P,X,E)] interface returns zero if and only if ** string X matches the [LIKE] pattern P with escape character E. ** ^The definition of [LIKE] pattern matching used in ** [sqlite3_strlike(P,X,E)] is the same as for the "X LIKE P ESCAPE E" ** operator in the SQL dialect understood by SQLite. ^For "X LIKE P" without ** the ESCAPE clause, set the E parameter of [sqlite3_strlike(P,X,E)] to 0. ** ^As with the LIKE operator, the [sqlite3_strlike(P,X,E)] function is case ** insensitive - equivalent upper and lower case ASCII characters match ** one another. ** ** ^The [sqlite3_strlike(P,X,E)] function matches Unicode characters, though ** only ASCII characters are case folded. ** ** Note that this routine returns zero on a match and non-zero if the strings ** do not match, the same as [sqlite3_stricmp()] and [sqlite3_strnicmp()]. ** ** See also: [sqlite3_strglob()].

70980. A row is being removed from the child table. Search for the parent. ** If the parent does not exist, removing the child row resolves an ** outstanding foreign key constraint violation.

70981. Complete text of a module argument

70982. Initialize the BLOB value of a ROWID

70983. Size of master journal file

70984. ** Append term pTerm/nTerm to the segment being written by the writer passed ** as the second argument. ** ** If an error occurs, set the Fts5Index.rc error code. If an error has ** already occurred, this function is a no-op.

70985. True if an error has been encountered

70986. Grab the index number and argc parameters

70987. ** Each cursor has a (possibly empty) linked list of the following objects.

70988. Set a SQLITE_TOOBIG error if no NULL

70989. !defined(SQLITE OMIT_FOREIGN_KEY)

70990. ** Interpret the given string as an auto-vacuum mode value. ** ** The following strings, "none", "full" and "incremental" are ** acceptable, as are their numeric equivalents: 0, 1 and 2 respectively.

70991. Translate from this opcode to the end

70992. ** Permitted values of the SrcList.a.jointype field

70993. Offset of journal header being read

70994. An index is used

70995. Case 2: We can directly reference a single row using an ** equality comparison against the ROWID field. Or ** we reference multiple rows using a "rowid IN (...)** construct.

70996. ** Purge the unixShmNodeList list of all entries with unixShmNode.nRef==0. ** ** This is not a VFS shared-memory method; it is a utility function called ** by VFS shared-memory methods.

70997. Default case. Handles, amongst others, "nfs". ** Test byte-range lock using fcntl(). If the call succeeds, ** assume that the file-system supports POSIX style locks.

70998. ** Generate code to do constraint checks prior to an INSERT or an UPDATE ** on table pTab. ** ** The regOldData parameter is the first register in a range that contains ** the data to be inserted or the data after the update. There will be ** pTab->nCol+1 registers in this range. The first register (the one ** that regOldData points to) will contain the new rowid, or NULL in the ** case of a WITHOUT ROWID table. The second register in the range will ** contain the content of the first table column. The third register will ** contain the content of the second table column. And so forth. ** ** The regOldData parameter is similar to regOldData except that it contains ** the data prior to an UPDATE rather than afterwards. regOldData is zero ** for an INSERT. This routine can distinguish between UPDATE and INSERT by ** checking regOldData for zero. ** ** For an UPDATE, the pkChng boolean is true if the true primary key (the ** rowid for a normal table or the PRIMARY KEY for a WITHOUT ROWID table) ** might be modified by the UPDATE. If pkChng is false, then the key of ** the iDataCur content table is guaranteed to be unchanged by the UPDATE. ** ** For an INSERT, the pkChng boolean indicates whether or not the rowid ** was explicitly specified as part of the INSERT statement. If pkChng ** is zero, it means that the either rowid is computed automatically or ** that the table is a WITHOUT ROWID table and has no rowid. On an INSERT, ** pkChng will only be true if the INSERT statement provides an integer ** value for either the rowid column or its INTEGER PRIMARY KEY alias. ** ** The code generated by this routine will store new index entries into ** registers identified by aRegIdx[]. No index entry is created for ** indices where aRegIdx[i]==0. The order of indices in aRegIdx[] is ** the same as the order of indices on the linked list of indices ** at pTab->pIndex. ** ** The caller must have already opened writeable cursors on the main ** table and all applicable indices (that is to say, all indices for which ** aRegIdx[] is not zero). iDataCur is the cursor for the main table when ** inserting or updating a rowid table, or the cursor for the PRIMARY KEY ** index when operating on a WITHOUT ROWID table. iIdxCur is the cursor ** for the first index in the pTab->pIndex list. Cursors for other indices ** are at iIdxCur+N for the N-th element of the pTab->pIndex list. ** ** This routine also generates code to check constraints. NOT NULL, ** CHECK, and UNIQUE constraints are all checked. If a constraint fails, ** then the appropriate action is performed. There are five possible ** actions: ROLLBACK, ABORT, FAIL, REPLACE, and IGNORE. ** ** Constraint type Action What Happens ** ----- ** any ROLLBACK The current transaction is rolled back and ** sqlite3_step() returns immediately with a ** return code of SQLITE_CONSTRAINT. ** ** any ABORT Back out changes from the current command ** only (do not do a complete rollback) then ** cause sqlite3_step() to return immediately ** with SQLITE_CONSTRAINT. ** ** any FAIL. Sqlite3_step() returns immediately with a ** return code of SQLITE_CONSTRAINT. The ** transaction is not rolled back and any ** changes to prior rows are retained. ** ** any IGNORE The attempt to insert or update the current ** row is skipped, without throwing an error. ** Processing continues with the next row. ** (There is an immediate jump to ignoreDest.) ** ** NOT NULL REPLACE The NULL value is replace by the default ** value for that column. If the default value ** is NULL, the action is the same as ABORT. ** ** UNIQUE REPLACE The other row that conflicts with the row ** being inserted is removed. ** ** CHECK REPLACE Illegal. The results in an exception. ** ** Which action to take is determined by the overrideError parameter. ** Or if overrideError==OE_Default, then the pParse->onError parameter ** is used. Or if pParse->onError==OE_Default then the onError value ** for the constraint is used.

70999. **comment:** ** Assert that the pointer X is aligned to an 8-byte boundary. This ** macro is used only within assert() to verify that the code gets ** all alignment restrictions correct. ** ** Except, if SQLITE_4_BYTE_ALIGNED_MALLOC is defined, then the ** underlying malloc() implementation might return us 4-byte aligned ** pointers. In that case, only verify 4-byte alignment.

label: code-design

71000. x<EXPR and/or x>EXPR

71001. Pointer to first byte after the header
71002. Argument to the function
71003. **comment:** *** Change the P4 operand of the most recently coded instruction ** to the value defined by the arguments. This is a high-speed ** version of sqlite3VdbeChangeP4(). *** The P4 operand must not have been previously defined. And the new ** P4 must not be P4_INT32. Use sqlite3VdbeChangeP4() in either of ** those cases.
label: code-design
71004. ***** FTS5 EXTENSION REGISTRATION API
71005. ** Return the collation sequence for the expression pExpr. If ** there is no defined collating sequence, return NULL. *** The collating sequence might be determined by a COLLATE operator ** or by the presence of a column with a defined collating sequence. ** COLLATE operators take first precedence. Left operands take ** precedence over right operands.
71006. The computed hash
71007. Condition 2
71008. 187
71009. paren_explist ::=
71010. Allocate an array and populate it with a copy of pCell and ** all cells from node pLeft. Then zero the original node.
71011. Determine whether or not this is a transaction savepoint. If so, ** and this is a RELEASE command, then the current transaction ** is committed.
71012. Increment the value just read and write it back to byte 24.
71013. ** The root-page of the master database table.
71014. In theory, the call to unixFileLock() cannot fail because another ** process is holding an incompatible lock. If it does, this ** indicates that the other process is not following the locking ** protocol. If this happens, return SQLITE_IOERR_RDLOCK. Returning ** SQLITE_BUSY would confuse the upper layer (in practice it causes ** an assert to fail).
71015. The first to decrement to 0 does actual shutdown ** (which should be the last to shutdown.)
71016. Number of columns must be the same in tab1 and tab2
71017. Compare against this WhereLoop
71018. trigger_decl ::= temp TRIGGER ifnotexists nm dbnm trigger_time trigger_event ON fullname foreach_clause when_clause
71019. The element that matches key (if any)
71020. content table
71021. Next in list of all winShmNode objects
71022. Drop any table triggers from the internal schema.
71023. ** The second argument to this function, op, is always SAVEPOINT_ROLLBACK ** or SAVEPOINT_RELEASE. This function either releases or rolls back the ** savepoint identified by parameter iSavepoint, depending on the value ** of op. *** Normally, iSavepoint is greater than or equal to zero. However, if op is ** SAVEPOINT_ROLLBACK, then iSavepoint may also be -1. In this case the ** contents of the entire transaction are rolled back. This is different ** from a normal transaction rollback, as no locks are released and the ** transaction remains open.
71024. Total cache hits, misses and writes
71025. Affinity to use for comparison
71026. cmd ::= SAVEPOINT nm
71027. Possible covering index for WHERE_MULTI_OR
71028. ***** Begin file fts3_unicode2.c *****
71029. ** Return the minimum number of 32KB shm regions that should be mapped at ** a time, assuming that each mapping must be an integer multiple of the ** current system page-size. *** Usually, this is 1. The exception seems to be systems that are configured ** to use 64KB pages - in this case each mapping must cover at least two ** shm regions.
71030. Allowed values for Column.colFlags:
71031. ** Candidate values for Fts3Query.eType. Note that the order of the first ** four values is in order of precedence when parsing expressions. For ** example, the following: *** "a OR b AND c NOT d NEAR e" *** is equivalent to: *** "a OR (b AND (c NOT (d NEAR e)))"
71032. The pointer map page number
71033. ** Parameter eMode must be one of the PAGER_JOURNALMODE_XXX constants ** defined in pager.h. This function returns the associated lowercase ** journal-mode name.
71034. Allocate and initialize the iterator structure.
71035. Matchinfo context object
71036. True if resolved to a trigger column
71037. Index used by loop (if any)
71038. 238
71039. The julian day number times 86400000
71040. Result code from subroutine call
71041. ** Explicitly call the 64-bit version of lseek() on Android. Otherwise, lseek() ** is the 32-bit version, even if _FILE_OFFSET_BITS=64 is defined.
71042. Wildcard of the form "?". Assign the next variable number
71043. FTS Cursor handle
71044. xOpen flags
71045. ** Implementation of xCommit() method. This is a no-op. The contents of ** the pending-terms hash-table have already been flushed into the database ** by fts5SyncMethod().
71046. ***** Include vdbeInt.h in the middle of status.c *****
71047. Forward references to VFS helper methods used for memory mapped files
71048. Neither operand is NULL. Do a comparison.
71049. trigger_cmd_list ::= trigger_cmd SEMI
71050. Used to iterate through change record
71051. **comment:** ** Remove all redundant columns from the PRIMARY KEY. For example, change ** "PRIMARY KEY(a,b,a,b,c,b,c,d)" into just "PRIMARY KEY(a,b,c,d)". Later ** code assumes the PRIMARY KEY contains no repeated columns.
label: code-design
71052. Fall through into the next case
71053. Opcode: ColumnsUsed P1 * * P4 * *** This opcode (which only exists if SQLite was compiled with ** SQLITE_ENABLE_COLUMN_USED_MASK) identifies which columns of the ** table or index for cursor P1 are used. P4 is a 64-bit integer ** (P4_INT64) in which the first 63 bits are one for each of the ** first 63 columns of the table or index that are actually used ** by the cursor. The high-order bit is set if any column after ** the 64th is used.
71054. shared memory is disabled
71055. New estimate of the number of rows
71056. Mask against sqlite3_index_constraint.op
71057. If the requested key is one more than the previous key, then ** try to get there using sqlite3BtreeNext() rather than a full ** binary search. This is an optimization only. The correct answer ** is still obtained without this case, only a little more slowly
71058. Cursors from source and destination
71059. OUTPUT: True if column part of PK
71060. OUT: bm25-data object for this query
71061. 'C'
71062. ** Characters that can be part of a token. We assume any character ** whose value is greater than 0x80 (any UTF character) can be ** part of a token. In other words, delimiters all must have ** values of 0x7f or lower.
71063. ** If *pRc is not SQLITE_OK when this function is called, it is a no-op. ** Otherwise, append an entry to the hint stored in blob *pHint. Each entry ** consists of two varints, the absolute level number of the input segments ** and the number of input segments. *** If successful, leave *pRc set to SQLITE_OK and return. If an error occurs, ** set *pRc to an SQLite error code before returning.
71064. Number of rows in index
71065. Enable mutex on connections

71066. This savepoint was opened immediately after the write-transaction ** was started. Right after that, the writer decided to wrap around ** to the start of the log.
 Update the savepoint values to match.

71067. ** Number of entries in the cursor RtreeNode cache. The first entry is ** used to cache the RtreeNode for RtreeCursor.sPoint. The remaining ** entries cache the RtreeNode for the first elements of the priority queue.

71068. restrictions (1) and (2)

71069. Size of term on leaf in bytes

71070. ** Close a cursor. The read lock on the database file is released ** when the last cursor is closed.

71071. ** Allocate a new RtreeSearchPoint and return a pointer to it. Return ** NULL if malloc fails.

71072. Info about index keys needed by index cursors

71073. Pointer to space allocated by sqlite3BtreeSchema()

71074. True if last operation was a delete

71075. Largest leaf block id written to db

71076. Set up a loop over the rowids/primary-keys that were found in the ** where-clause loop above.

71077. Store the result as data using a unique key.

71078. A type code from the record header

71079. setlist ::= nm EQ expr

71080. List of tables to select from

71081. Opcode: LoadAnalysis P1 * * * * * Read the sqlite_stat1 table for database P1 and load the content ** of that table into the internal index hash table. This will cause ** the analysis to be used when preparing all subsequent queries.

71082. ** Return the current value of a status parameter. The caller must ** be holding the appropriate mutex.

71083. ** Zero the iterator passed as the only argument.

71084. ** Return TRUE if the name zCol occurs anywhere in the USING clause. ** ** Return FALSE if the USING clause is NULL or if it does not contain ** zCol.

71085. ** apCell[] and szCell[] contains pointers to and sizes of all cells in the ** pages being balanced. The current page, pPg, has pPg->nCell cells starting ** with apCell[iOld]. After balancing, this page should hold nNew cells ** starting at apCell[iNew]. ** ** This routine makes the necessary adjustments to pPg so that it contains ** the correct cells after being balanced. ** ** The pPg->nFree field is invalid when this function returns. It is the ** responsibility of the caller to set it correctly.

71086. ** Locate the in-memory structure that describes ** a particular index given the name of that index ** and the name of the database that contains the index. ** Return NULL if not found. ** ** If zDatabase is 0, all databases are searched for the ** table and the first matching index is returned. (No checking ** for duplicate index names is done.) The search order is ** TEMP first, then MAIN, then any auxiliary databases added ** using the ATTACH command.

71087. ** Record in the column cache that a particular column from a ** particular table is stored in a particular register.

71088. Restriction (11)

71089. The docid column

71090. EVIDENCE-OF: R-38229-40159 If the callback function to ** sqlite3_exec() returns non-zero, then sqlite3_exec() will ** return SQLITE_ABORT.

71091. Pointer to entire doclist

71092. virtual table name

71093. SQLITE_FULL

71094. Pointer to the subexpression that is this term

71095. Temporary file object wrapper

71096. ***** End of mutex.c *****

71097. ** An object used to accumulate the text of a string where we ** do not necessarily know how big the string will be in the end.

71098. **comment:** ** The dupedExpr*Size() routines each return the number of bytes required ** to store a copy of an expression or expression tree. They differ in ** how much of the tree is measured. ** ** dupedExprStructSize() Size of only the Expr structure ** dupedExprNodeSize() Size of Expr + space for token ** dupedExprSize() Expr + token + subtree components **** The dupedExprStructSize() function returns two values OR-ed together: ** (1) the space required for a copy of the Expr structure only and ** (2) the EP_xxx flags that indicate what the structure size should be. ** The return values is always one of: ** ** EXPR_FULLSCREEN ** EXPR_REDUCEDSIZE | EP_Reduced ** EXPR_TOKENONLYSIZE | EP_TokenOnly ** ** The size of the structure can be found by masking the return value ** of this routine with 0xffff. The flags can be found by masking the ** return value with EP_Reduced|EP_TokenOnly. ** ** Note that with flags==EXPRDUP_REDUCE, this routines works on full-size ** (unreduced) Expr objects as they or originally constructed by the parser. ** During expression analysis, extra information is computed and moved into ** later parts of teh Expr object and that extra information might get chopped ** off if the expression is reduced. Note also that it does not work to ** make an EXPRDUP_REDUCE copy of a reduced expression. It is only legal ** to reduce a pristine expression tree from the parser. The implementation ** of dupedExprStructSize() contain multiple assert() statements that attempt ** to enforce this constraint.

label: code-design

71099. Session table handle

71100. ** VDBE_DISPLAY_P4 is true or false depending on whether or not the ** "explain" P4 display logic is enabled.

71101. ***** Begin file select.c *****

71102. 1100

71103. ** Make sure the page is marked as dirty. If it isn't dirty already, ** make it so.

71104. ** 2006 Oct 10 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** **** Implementation of the "simple" full-text-search tokenizer.

71105. **comment:** ***** Posix Advisory Locking ***** POSIX advisory locks are broken by design. ANSI STD 1003.1 (1996) ** section 6.5.2.2 lines 483 through 490 specify that when a process ** sets or clears a lock, that operation overrides any prior locks set ** by the same process. It does not explicitly say so, but this implies ** that it overrides locks set by the same process using a different ** file descriptor. Consider this test case: ** ** int fd1 = open("./file1", O_RDWR|O_CREAT, 0644); ** int fd2 = open("./file2", O_RDWR|O_CREAT, 0644); ** ** Suppose ./file1 and ./file2 are really the same file (because ** one is a hard or symbolic link to the other) then if you set ** an exclusive lock on fd1, then try to get an exclusive lock ** on fd2, it works. I would have expected the second lock to ** fail since there was already a lock on the file due to fd1. ** But not so. Since both locks came from the same process, the ** second overrides the first, even though they were on different ** file descriptors opened on different file names. ** ** This means that we cannot use POSIX locks to synchronize file access ** among competing threads of the same process. POSIX locks will work fine ** to synchronize access for threads in separate processes, but not ** threads within the same process. ** ** To work around the problem, SQLite has to manage file locks internally ** on its own. Whenever a new database is opened, we have to find the ** specific inode of the database file (the inode is determined by the ** st_dev and st_ino fields of the stat structure that fstat() fills in) ** and check for locks already existing on that inode. When locks are ** created or removed, we have to look at our own internal record of the ** locks to see if another thread has previously set a lock on that same ** inode. ** ** (Aside: The use of inode numbers as unique IDs does not work on VxWorks. ** For VxWorks, we have to use the alternative unique ID system based on ** canonical filename and implemented in the previous division.) ** ** The sqlite3_file structure for POSIX is no longer just an integer file ** descriptor. It is now a structure that holds the integer file ** descriptor and a pointer to a structure that describes the internal ** locks on the corresponding inode. There is one locking structure ** per inode, so if the same inode is opened twice, both unixFile structures ** point to the same locking structure. The locking structure keeps ** a reference count (so we will know when to delete it) and a "cnt" ** field that tells us its internal lock status. cnt==0 means the ** file is unlocked. cnt==1 means the file has an exclusive lock. ** cnt>0 means there are cnt shared locks on the file. ** ** Any attempt to lock or unlock a file first checks the locking ** structure. The fcntl() system call is only invoked to set a ** POSIX lock if the internal lock structure transitions between ** a locked and an unlocked state. ** ** But wait: there are yet more problems with POSIX advisory locks. ** ** If you close a file descriptor that points to a file that has locks, ** all locks on that file that are owned by the current process are ** released. To work around this problem, each unixInodeInfo object ** maintains a count of the number of pending locks on the inode. ** When an attempt is made to close an unixFile, if there are ** other unixFile open on the same inode that are holding locks, the call ** to close() the file descriptor is deferred until all of the locks clear. ** The unixInodeInfo structure keeps a list of file descriptors that need to ** be closed and that list is walked (and cleared) when the last lock ** clears. ** ** Yet another problem: LinuxThreads do not play well with posix locks. ** ** Many older versions of linux use the LinuxThreads library which is ** not posix compliant. Under LinuxThreads, a lock created by thread ** A cannot be modified or overridden by a different thread B. ** Only thread A can modify the lock. Locking behavior is correct ** if the application uses the newer Native Posix Thread Library (NPTL) ** on linux - with NPTL a lock created by thread A can override locks ** in thread B. But there is no way to know at compile-time which ** threading library is being used. So there is no way to know at ** compile-time whether or not thread A can override locks on thread B. **

One has to do a run-time check to discover the behavior of the ** current process. *** SQLite used to support LinuxThreads. But support for LinuxThreads ** was dropped beginning with version 3.7.0. SQLite will still work with ** LinuxThreads provided that (1) there is no more than one connection ** per database file in the same process and (2) database connections ** do not move across threads.

label: code-design

71106. synopsis: iplan=r[P3] zplan='P4'

71107. ** Virtual table structure.

71108. Number of entries in array

71109. The flags parameter to sqlite3BtreeOpen can be the bitwise or of the ** following values. *** NOTE: These values must match the corresponding PAGER_ values in ** pager.h.

71110. All tables indexed by name

71111. Mark every PRIMARY KEY column as NOT NULL (except for imposter tables)

71112. ** If the cell pCell, part of page pPage contains a pointer ** to an overflow page, insert an entry into the pointer-map ** for the overflow page.

71113. Write the database source cursor number here

71114. One of TK_ALL, TK_UNION, TK_EXCEPT, TK_INTERSECT

71115. True to omit the xSync on the db file

71116. Base cost: N*3

71117. YYDEBUG

71118. First doclist

71119. Table has a primary key

71120. Mapping from ORDER BY terms to result set columns

71121. Control reaches here if the candidate path is better than the ** pTo path. Replace pTo with the candidate.

71122. This branch happens for multiple negative signs. Ex: $\{(-5)\}$

71123. Name context for container

71124. Next cursor in Fts5Cursor.pCsr list

71125. Connection holding shared memory

71126. SQLITE_OMIT_TRIGGER

71127. If the source page-size is smaller than the destination page-size, ** two extra things may need to happen: *** * The destination may need to be truncated, and ** * Data stored on the pages immediately following the ** pending-byte page in the source database may need to be ** copied into the destination database.

71128. **comment:** WAL magic value. Either this value, or the same value with the least ** significant bit also set (WAL_MAGIC | 0x00000001) is stored in 32-bit ** big-endian format in the first 4 bytes of a WAL file. *** If the LSB is set, then the checksums for each frame within the WAL ** file are calculated by treating all data as an array of 32-bit ** big-endian words. Otherwise, they are calculated by interpreting ** all data as 32-bit little-endian words.

label: code-design

71129. ** Return the number of bytes required to store a JournalFile that uses vfs ** pVfs to create the underlying on-disk files.

71130. Negative of the number of RHS symbols in the rule

71131. ** CAPI3REF: Prepared Statement Scan Status Opcodes ** KEYWORDS: {scanstatus options} *** The following constants can be used for the T parameter to the ** [sqlite3_stmt_scanstatus(S,X,T,V)] interface. Each constant designates a ** different metric for sqlite3_stmt_scanstatus() to return. *** When the value returned to V is a string, space to hold that string is ** managed by the prepared statement S and will be automatically freed when ** S is finalized. *** <dd> ** [[SQLITE_SCANSTAT_NLOOP]] <dt>SQLITE_SCANSTAT_NLOOP</dt> ** <dd>^The [sqlite3_int64] variable pointed to by the T parameter will be ** set to the total number of times that the X-th loop has run.</dd> *** [[SQLITE_SCANSTAT_NVISIT]] <dt>SQLITE_SCANSTAT_NVISIT</dt> ** <dd>^The [sqlite3_int64] variable pointed to by the T parameter will be set ** to the total number of rows examined by all iterations of the X-th loop.</dd> *** <dd>^The [[SQLITE_SCANSTAT_EST]] <dt>SQLITE_SCANSTAT_EST</dt> ** <dd>^The "double" variable pointed to by the T parameter will be set to the ** query planner's estimate for the average number of rows output from each ** iteration of the X-th loop. If the query planner's estimates was accurate, ** then this value will approximate the quotient NVISIT/NLOOP and the ** product of this value for all prior loops with the same SELECTID will ** be the NLOOP value for the current loop. *** [[SQLITE_SCANSTAT_NAME]] <dt>SQLITE_SCANSTAT_NAME</dt> ** <dd>^The "const char **" variable pointed to by the T parameter will be set ** to a zero-terminated UTF-8 string containing the name of the index or table ** used for the X-th loop. *** [[SQLITE_SCANSTAT_EXPLAIN]] <dt>SQLITE_SCANSTAT_EXPLAIN</dt> ** <dd>^The "const char **" variable pointed to by the T parameter will be set ** to a zero-terminated UTF-8 string containing the [EXPLAIN QUERY PLAN] ** description for the X-th loop. *** [[SQLITE_SCANSTAT_SELECTID]] <dt>SQLITE_SCANSTAT_SELECT</dt> ** <dd>^The "int" variable pointed to by the T parameter will be set to the ** "select-id" for the X-th loop. The select-id identifies which query or ** subquery the loop is part of. The main query has a select-id of zero. ** The select-id is the same value as is output in the first column ** of an [EXPLAIN QUERY PLAN] query. *** </dl>

71132. ** Lock the file with the lock specified by parameter eFileLock - one ** of the following: *** (1) SHARED_LOCK ** (2) RESERVED_LOCK ** (3) PENDING_LOCK ** (4) EXCLUSIVE_LOCK ** ** Sometimes when requesting one lock state, additional lock states ** are inserted in between. The locking might fail on one of the later ** transitions leaving the lock state different from what it started but ** still short of its goal. The following chart shows the allowed ** transitions and the inserted intermediate states: *** UNLOCKED -> SHARED ** SHARED -> RESERVED ** SHARED -> (PENDING) -> EXCLUSIVE ** RESERVED -> (PENDING) -> EXCLUSIVE ** PENDING -> EXCLUSIVE ** ** flock() only really support EXCLUSIVE locks. We track intermediate ** lock states in the sqlite3_file structure, but all locks SHARED or ** above are really EXCLUSIVE locks and exclude all other processes from ** access the file. ** ** This routine will only increase a lock. Use the sqlite3OsUnlock() ** routine to lower a locking level.

71133. ** Allowed return values from sqlite3FindInIndex()

71134. Index.azColl

71135. sign of significand

71136. 0 for normal table, 1 for WITHOUT ROWID table

71137. Number of bytes in pList

71138. nRefInitMutex

71139. 670

71140. Frame data

71141. Index of current sample accessed by stat_get()

71142. OUT: Store cursor handle here

71143. languageId=xxx option, or NULL

71144. nexplist

71145. Table <table>

71146. Size of buffer pData in bytes

71147. ** Create and return a deep copy of the object passed as the second ** argument. If an OOM condition is encountered, NULL is returned ** and the db->mallocFailed flag set.

71148. SQL Collation sequence name (eg. "japanese")

71149. ** True if heap is nearly "full" where "full" is defined by the ** sqlite3_soft_heap_limit() setting.

71150. ***** Begin file mutex.c *****

71151. Functions used to obtain and release page references.

71152. ** Maximum number of tokens a phrase may have to be considered for the ** incremental doclists strategy.

71153. fall through

71154. First arg for xInput

71155. Open connection to the WAL file

71156. Next free slot

71157. PRAGMA index_xinfo (newer version with more rows and columns)

71158. Disable the DISTINCT optimization if SQLITE_DistinctOpt is set via ** sqlite3_test_ctrl(SQLITE_TESTCTRL_OPTIMIZATIONS,...)

71159. Map from vector field to index column

71160. True if a new wal-index header is loaded

71161. ** Virtual-table object.

71162. ** Apply the changeset passed via pChangeset/nChangeset to the main database ** attached to handle "db". Invoke the supplied conflict handler callback ** to resolve any conflicts encountered while applying the change.

71163. VFS flags returned by xOpen()

71164. If the source pager is currently in a write-transaction, return ** SQLITE_BUSY immediately.

71165. IN/OUT: Previous value written to list

71166. ** The WHERE clause processing routine has two halves. The ** first part does the start of the WHERE loop and the second ** half does the tail of the WHERE loop. An instance of ** this structure is returned by the first half and passed ** into the second half to give some continuity. ** ** An instance of this object holds the complete state of the query ** planner.

71167. ** Rowid iRowid has just been appended to the current leaf page. It is the ** first on the page. This function appends an appropriate entry to the current ** doclist-index.

71168. ** Obtain a pointer to the iPage'th page of the wal-index. The wal-index ** is broken into pages of WALINDEX_PGSZ bytes. Wal-index pages are ** numbered from zero. ** ** If this call is successful, *ppPage is set to point to the wal-index ** page and SQLITE_OK is returned. If an error (an OOM or VFS error) occurs, ** then an SQLite error code is returned and *ppPage is set to 0.

71169. Current overflow page number

71170. Index to search (from 0 to p->nIndex-1)

71171. Check that the doclist does not appear to extend past the end of the ** b-tree node. And that the final byte of the doclist is 0x00. If either ** of these statements is untrue, then the data structure is corrupt.

71172. **comment:** ** This structure is used to implement an iterator that loops through ** all frames in the WAL in database page order. Where two or more frames ** correspond to the same database page, the iterator visits only the ** frame most recently written to the WAL (in other words, the frame with ** the largest index). ** ** The internals of this structure are only accessed by: ** ** walIteratorInit() - Create a new iterator, ** walIteratorNext() - Step an iterator, ** walIteratorFree() - Free an iterator. ** ** This functionality is used by the checkpoint code (see walCheckpoint()).

label: code-design

71173. Register(s) holding the LHS values

71174. True to truncate WAL file on commit

71175. docid<=?

71176. ** Close all open savepoints. This function only manipulates fields of the ** database handle object, it does not close any savepoints that may be open ** at the b-tree/pager level.

71177. ** Skip over any TK_COLLATE operators and any unlikely() ** or likelihood() function at the root of an expression.

71178. Invoked to set output variables.

71179. Number of users of pInitMutex

71180. ** Object for iterating through the merged results of one or more segments, ** visiting each term/rowid pair in the merged data. ** ** nSeg is always a power of two greater than or equal to the number of ** segments that this object is merging data from. Both the aSeg[] and ** aFirst[] arrays are sized at nSeg entries. The aSeg[] array is padded ** with zeroed objects - these are handled as if they were iterators opened ** on empty segments. ** ** The results of comparing segments aSeg[N] and aSeg[N+1], where N is an ** even number, is stored in aFirst[(nSeg+N)/2]. The "result" of the ** comparison in this context is the index of the iterator that currently ** points to the smaller term/rowid combination. Iterators at EOF are ** considered to be greater than all other iterators. ** ** aFirst[1] contains the index in aSeg[] of the iterator that points to ** the smallest key overall. aFirst[0] is unused. ** ** poslist: ** Used by sqlite3Fts5IterPoslist() when the poslist needs to be buffered. ** There is no way to tell if this is populated or not.

71181. True if not checked out

71182. **comment:** ** This function is called whenever processing of the doclist for the ** last term on leaf page (pWriter->iBtPage) is completed. ** ** The doclist-index for that term is currently stored in-memory within the ** Fts5SegWriter.aDlidx[] array. If it is large enough, this function ** writes it out to disk. Or, if it is too small to bother with, discards ** it. ** ** Fts5SegWriter.btterm currently contains the first term on page iBtPage.

label: code-design

71183. The ORDER BY clause or NULL

71184. ** Check the RTree node or entry given by pCellData and p against the MATCH ** constraint pConstraint.

71185. ** Allocate an Expr node which joins as many as two subtrees. ** ** One or both of the subtrees can be NULL. Return a pointer to the new ** Expr node. Or, if an OOM error occurs, set pParse->db->mallocFailed, ** free the subtrees and return NULL.

71186. Only one writer allowed at a time. Get the write lock. Return ** SQLITE_BUSY if unable.

71187. **comment:** ** Shutdown the operating system interface. ** ** Some operating systems might need to do some cleanup in this routine, ** to release dynamically allocated objects. But not on unix. ** This routine is a no-op for unix.

label: code-design

71188. ** CAPI3REF: String Globbing * ** ^The [sqlite3_strglob(P,X)] interface returns zero if and only if ** string X matches the [GLOB] pattern P. ** ^The definition of [GLOB] pattern matching used in ** [sqlite3_strglob(P,X)] is the same as for the "X GLOB P" operator in the ** SQL dialect understood by SQLite. ^The [sqlite3_strglob(P,X)] function ** is case sensitive. ** ** Note that this routine returns zero on a match and non-zero if the strings ** do not match, the same as [sqlite3_strcmp()] and [sqlite3_strnicmp()]. ** ** See also: [sqlite3_strlike()].

71189. ** Return coordinate iCoord from cell iCell in node pNode.

71190. ** Invoke this routine to register the "dbstat" virtual table module

71191. Affinity for start of range constraint

71192. Consider functions to be constant if all their arguments are constant ** and either pWalker->eCode==4 or 5 or the function has the ** SQLITE_FUNC_CONST flag.

71193. Allocate space for the cursor, filter and writer objects

71194. If required, populate the overflow page-list cache.

71195. **comment:** ** This function is part of the fts5_decode() debugging function. It is ** only ever used with detail=none tables. ** ** Buffer (pData/nData) contains a doclist in the format used by detail=none ** tables. This function appends a human-readable version of that list to ** buffer pBuf. ** ** If *pRc is other than SQLITE_OK when this function is called, it is a ** no-op. If an OOM or other error occurs within this function, *pRc is ** set to an SQLite error code before returning. The final state of buffer ** pBuf is undefined in this case.

label: code-design

71196. Start of BEFORE trigger programs

71197. Read blob data into this buffer

71198. OP_Destroy stores an in integer r1. If this integer ** is non-zero, then it is the root page number of a table moved to ** location iTable. The following code modifies the sqlite_master table to ** reflect this. ** ** The "#NNN" in the SQL is a special constant that means whatever value ** is in register NNN. See grammar rules associated with the TK_REGISTER ** token for additional information.

71199. xDisconnect - Disconnect from a table

71200. Range of registers holding conflicting PK

71201. xLog

71202. Do not sync the journal if true

71203. context in which to report errors

71204. Construct an expression node for a unary prefix operator

71205. **comment:** ** Journal files begin with the following magic string. The data ** was obtained from /dev/random. It is used only as a sanity check. ** ** Since version 2.8.0, the journal format contains additional sanity ** checking information. If the power fails while the journal is being ** written, semi-random garbage data might appear in the journal ** file after power is restored. If an attempt is then made ** to roll the journal back, the database could be corrupted. The additional ** sanity checking data is an attempt to discover the garbage in the ** journal and ignore it. ** ** The sanity checking information for the new journal format consists ** of a 32-bit checksum on each page of data. The checksum covers both ** the page number and the pPager->pageSize bytes of data for the page. ** This cksum is initialized to a 32-bit random value that appears in the ** journal file right after the header. The random initializer is important, ** because garbage data that appears at the end of a journal is likely ** data that was once in other files that have now been deleted. If the ** garbage data came from an obsolete journal file, the checksums might ** be correct. But by initializing the checksum to random value which ** is different for every journal, we minimize that risk.

label: code-design

71206. ** Resolve all names for all expression in an expression list. This is ** just like sqlite3ResolveExprNames() except that it works for an expression ** list rather than a single expression.

71207. Size of aInput[] in bytes

71208. AUTOINCREMENT

71209. ** The winVfsAppData structure is used for the pAppData member for all of the ** Win32 VFS variants.

71210. Used when flags==MEM_Frame

71211. ** The iterator currently points to a table (not index) of type ** RBU_PK_WITHOUT_ROWID. This function creates the PRIMARY KEY ** declaration for the corresponding imposter table. For example, ** if the iterator points to a table created as: *** ** CREATE TABLE t1(a, b, c, PRIMARY KEY(b, a DESC)) WITHOUT ROWID ** ** this function returns: *** ** PRIMARY KEY("b", "a" DESC)

71212. Identifiers have been resolved

71213. aCell array

71214. The first 8 memory cells are used for the result set. So we will ** commandeer the 9th cell to use as storage for an array of pointers ** to trigger subprograms. The VDBE is guaranteed to have at least 9 ** cells.

71215. EVIDENCE-OF: R-50385-09674 Value is a big-endian 48-bit ** twos-complement integer.

71216. PK array for current table

71217. ** Figure out if we are dealing with Unix, Windows, or some other operating ** system. *** After the following block of preprocess macros, all of SQLITE_OS_UNIX, ** SQLITE_OS_WIN, and SQLITE_OS_OTHER will be defined to either 1 or 0. One of ** the three will be 1. The other two will be 0.

71218. ** Close a statvfs cursor.

71219. 1

71220. between_op ::= NOT BETWEEN

71221. Bx

71222. Destination for coroutine B

71223. ***** Linked List Management *****

71224. Language id used with tokenizer

71225. ** Extract the next token from buffer z (length n) using the tokenizer ** and other information (column names etc.) in pParse. Create an Fts3Expr ** structure of type FTSQUERY_PHRASE containing a phrase consisting of this ** single token and set *ppExpr to point to it. If the end of the buffer is ** reached before a token is found, set *ppExpr to zero. It is the ** responsibility of the caller to eventually deallocate the allocated ** Fts3Expr structure (if any) by passing it to sqlite3_free(). *** Return SQLITE_OK if successful, or SQLITE_NOMEM if a memory allocation ** fails.

71226. _SQLITE3RBU_H

71227. ... also used as column name list in a VIEW

71228. Address of the select-B-exhausted subroutine

71229. NEAR distance. As in "NEAR/nNear".

71230. ** Remove connection db from the blocked connections list. If connection ** db is not currently a part of the list, this function is a no-op.

71231. SQLITE_UTF8, SQLITE_UTF16BE or UTF16LE

71232. Page to add cells to

71233. ** Build a trigger step out of an INSERT statement. Return a pointer ** to the new trigger step. *** The parser calls this routine when it sees an INSERT inside the ** body of a trigger.

71234. Used to find the beginnings of sentences

71235. random NFS retry error, unless during file system support * introspection, in which it actually means what it says

71236. Determine iLower and iUpper using (\$P) only.

71237. ** Compare the term in buffer zLhs (size in bytes nLhs) with that in ** zRhs (size in bytes nRhs) using memcmp. If one term is a prefix of ** the other, it is considered to be smaller than the other. *** Return -ve if zLhs is smaller than zRhs, 0 if it is equal, or +ve ** if it is greater.

71238. IN/OUT: Page content -area pointer

71239. An index on pSrc

71240. ** Implementation of the scalar function icu_load_collation(). *** This scalar function is used to add ICU collation based collation ** types to an SQLite database connection. It is intended to be called ** as follows: *** SELECT icu_load_collation(<locale>, <collation-name>); *** Where <locale> is a string containing an ICU locale identifier (i.e. ** "en_AU", "tr_TR" etc.) and <collation-name> is the name of the ** collation sequence to create.

71241. ** Space for tracking which blocks are checked out and the size ** of each block. One byte per block.

71242. 20

71243. Some kind of disk I/O error occurred

71244. The SELECT statement containing the ORDER BY

71245. There should be at least one opcode.

71246. Result is an aggregate

71247. ***** End of pcache.c *****

71248. **comment:** Database connection malloc context
label: code-design

71249. Maximum JUMP opcode

71250. Pointer to where zName will be stored

71251. Jump here to break out of the loop

71252. pOrderby is really a DISTINCT clause

71253. TK_REGISTER: original value of Expr.op ** TK_COLUMN: the value of p5 for OP_Column ** TK_AGG_FUNCTION: nesting depth

71254. **comment:** ** RowSetEntry objects are allocated in large chunks (instances of the ** following structure) to reduce memory allocation overhead. The ** chunks are kept on a linked list so that they can be deallocated ** when the RowSet is destroyed.
label: code-design

71255. **comment:** The index to be tested for coverage
label: test

71256. cellSizePtr method

71257. Call stat() on path zIn. Set bLink to true if the path is a symbolic ** link, or false otherwise.

71258. Reset and ready to run again

71259. ** This routine implements the OP_Vacuum opcode of the VDBE.

71260. WhereClause currently being scanned

71261. This column is allowed to be NULL

71262. The AFFINITY() function evaluates to a string that describes ** the type affinity of the argument. This is used for testing of ** the SQLite type logic.

71263. 0x05

71264. **comment:** ** This function is called on a MultiSegReader that has been started using ** sqlite3Fts3MsrIncrStart(). One or more calls to MsrIncrNext() may also ** have been made. Calling this function puts the MultiSegReader in such ** a state that if the next two calls are: *** ** sqlite3Fts3SegReaderStart() ** sqlite3Fts3SegReaderStep() ** ** then the entire doclist for the term is available in ** MultiSegReader.aDoclist/nDoclist.
label: code-design

71265. Construct a new Expr object from a single identifier. Use the ** new Expr to populate pOut. Set the span of pOut to be the identifier ** that created the expression.

71266. Unqualified name of the table to create

71267. The average document size, which is required to calculate the cost ** of each doclist, has not yet been determined. Read the required ** data from the %_stat table to calculate it. *** Entry 0 of the %_stat table is a blob containing (nCol+1) FTS3 ** varints, where nCol is the number of columns in the FTS3 table. ** The first varint is the number of documents currently stored in ** the table. The following nCol varints contain the total amount of ** data stored in all rows of each column of the table, from left ** to right.

71268. If pReadr1 contained the smaller value, set aTree[i] to its index. ** Then set pReadr2 to the next PmaReader to compare to pReadr1. In this ** case there is no cache of pReadr2 in pTask->pUnpacked, so set ** pKey2 to point to the record belonging to pReadr2. *** Alternatively, if pReadr2 contains the smaller of the two values, ** set aTree[i] to its index and update pReadr1. If vdbeSorterCompare() ** was actually called above, then pTask->pUnpacked now contains ** a value equivalent to pReadr2. So set pKey2 to NULL to prevent ** vdbeSorterCompare() from decoding pReadr2 again. *** If the two values were equal, then

the value from the oldest ** PMA should be considered smaller. The VdbeSorter.aReadr[] array ** is sorted from oldest to newest, so pReadr1 contains older values ** than pReadr2 iff (pReadr1 < pReadr2).

71269. ShallowCopy only needs to copy the information above

71270. ** The MSVCRT has malloc_usable_size(), but it is called _msize(). The ** use of _msize() is automatic, but can be disabled by compiling with ** -DSQLITE_WITHOUT_MSIZE. Using the _msize() function also requires ** the malloc.h header file.

71271. fullname

71272. Length of string zMaster

71273. pParse->nQueryLoop outside the WHERE loop

71274. Description of the substitution

71275. Various PAGER_* flags

71276. ***** Begin file sqlite3.h *****

71277. ***** Begin file wherecode.c *****

71278. ** Free all components of the Fts3Phrase structure that were allocated by ** the eval module. Specifically, this means to free: ** * * * the contents of pPhrase->doclist, and ** * any Fts3MultiSegReader objects held by phrase tokens.

71279. COLUMN or TABLE.COLUMN if no AS clause and is direct

71280. Index of column this phrase must match

71281. Drop the temporary PENDING lock

71282. aLoop[]s that should be reversed for ORDER BY

71283. Number of elements in aList

71284. ** Memory available for allocation. nPool is the size of the array ** (in Mem3Blocks) pointed to by aPool less 2.

71285. One for each column cache entry

71286. ** Obtain a reference to a memory mapped page object for page number pgno. ** The new object will use the pointer pData, obtained from xFetch(). ** If successful, set *ppPage to point to the new page reference ** and return SQLITE_OK. Otherwise, return an SQLite error code and set ** *ppPage to zero. ** ** Page references obtained by calling this function should be released ** by calling pagerReleaseMapPage().

71287. ** Show the complete content of a WhereClause

71288. 53

71289. The trunk page is required by the caller but it contains ** pointers to free-list leaves. The first leaf becomes a trunk ** page in this case.

71290. tFromCol = OLD.tToCol

71291. Prepared statement being queried

71292. .uTemp =

71293. ** This function is used to write a single varint to a buffer. The varint ** is written to *pp. Before returning, *pp is set to point 1 byte past the ** end of the value written. ** ** If *pbFirst is zero when this function is called, the value written to ** the buffer is that of parameter iVal. ** ** If *pbFirst is non-zero when this function is called, then the value ** written is either (iVal.*piPrev) (if bDescIdx is zero) or (*piPrev-iVal) ** (if bDescIdx is non-zero). ** ** Before returning, this function always sets *pbFirst to 1 and *piPrev ** to the value of parameter iVal.

71294. Must be count()

71295. Array of OP_Once flags

71296. Text of the CREATE TABLE or CREATE VIEW statement

71297. A-overwrites-C

71298. **comment:** Sizes of inserted documents
label: documentation

71299. doubly linked

71300. Amount to pop the stack

71301. Both are UTF16, but with different byte orders

71302. ONEPASS is valid for multiple rows

71303. The memory locations

71304. sqlite3_file methods

71305. 0123456789 123456789 123456789 123

71306. IN/OUT: Mask of tokens to highlight

71307. Buffer containing current poslist

71308. Binding to these vars invalidates VM

71309. **comment:** ** When a cached column is reused, make sure that its register is ** no longer available as a temp register. ticket #3879: that same ** register might be in the cache in multiple places, so be sure to ** get them all.
label: code-design

71310. ** CAPI3REF: Checkpoint a database ** METHOD: sqlite3 *** ** ^{(The sqlite3_wal_checkpoint_v2(D,X,M,L,C) interface runs a checkpoint ** operation on database X of [database connection] D in mode M. Status ** information is written back into integers pointed to by L and C.)}^ ** ^{(The M parameter must be a valid [checkpoint mode];)}^ ** ^<dd> ** ^<dt>SQLITE_CHECKPOINT_PASSIVE<dd> ** ^Checkpoint as many frames as possible without waiting for any database ** readers or writers to finish, then sync the database file if all frames ** in the log were checkpointerd. ^{The [busy-handler callback] ** is never invoked in the SQLITE_CHECKPOINT_PASSIVE mode. ** ^On the other hand, passive mode might leave the checkpoint unfinished ** if there are concurrent readers or writers. ** ** ^<dt>SQLITE_CHECKPOINT_FULL<dd> ** ^This mode blocks (it invokes the ** [sqlite3_busy_handler|busy-handler callback]) until there is no ** database writer and all readers are reading from the most recent database ** snapshot. ^{It then checkpoints all frames in the log file and syncs the ** database file. ^{This mode blocks new database writers while it is pending, ** but new database readers are allowed to continue unimpeded. ** ** ^<dt>SQLITE_CHECKPOINT_RESTART<dd> ** ^This mode works the same way as SQLITE_CHECKPOINT_FULL with the addition ** that after checkpointing the log file it blocks (calls the ** [busy-handler callback]) ** until all readers are reading from the database file only. ^{This ensures ** that the next writer will restart the log file from the beginning. ** ^Like SQLITE_CHECKPOINT_FULL, this mode blocks new ** database writer attempts while it is pending, but does not impede readers. ** ** ^<dt>SQLITE_CHECKPOINT_TRUNCATE<dd> ** ^This mode works the same way as SQLITE_CHECKPOINT_RESTART with the ** addition that it also truncates the log file to zero bytes just prior ** to a successful return. ** ^</dt> ** ** ^{If pnLog is not NULL, then *pnLog is set to the total number of frames in ** the log file or to -1 if the checkpoint could not run because ** of an error or because the database is not in [WAL mode]. ^{If pnCkpt is not ** NULL, then *pnCkpt is set to the total number of checkpointed frames in the ** log file (including any that were already checkpointed before the function ** was called) or to -1 if the checkpoint could not run due to an error or ** because the database is not in WAL mode. ^{Note that upon successful ** completion of an SQLITE_CHECKPOINT_TRUNCATE, the log file will have been ** truncated to zero bytes and so both *pnLog and *pnCkpt will be set to zero. ** ** ^{All calls obtain an exclusive "checkpoint" lock on the database file. ^{If ** any other process is running a checkpoint operation at the same time, the ** lock cannot be obtained and SQLITE_BUSY is returned. ^{Even if there is a ** busy-handler configured, it will not be invoked in this case. ** ** ^{The SQLITE_CHECKPOINT_FULL, RESTART and TRUNCATE modes also obtain the ** exclusive "writer" lock on the database file. ^{If the writer lock cannot be ** obtained immediately, and a busy-handler is configured, it is invoked and ** the writer lock retried until either the busy-handler returns 0 or the lock ** is successfully obtained. ^{The busy-handler is also invoked while waiting for ** database readers as described above. ^{If the busy-handler returns 0 before ** the writer lock is obtained or while waiting for database readers, the ** checkpoint operation proceeds from that point in the same way as ** ^{SQLITE_CHECKPOINT_PASSIVE - checkpointing as many frames as possible ** without blocking any further. ^{SQLITE_BUSY is returned in this case. ** ** ^{If parameter zDb is NULL or points to a zero length string, then the ** specified operation is attempted on all WAL databases [attached] to ** [database connection] db. In this case the ** values written to output parameters *pnLog and *pnCkpt are undefined. ^{If ** an SQLITE_BUSY error is encountered when processing one or more of the ** attached WAL databases, the operation is still attempted on any remaining ** attached databases and SQLITE_BUSY is returned at the end. ^{If any other ** error occurs while processing an attached database, processing is abandoned ** and the error code is returned to the caller immediately. ^{If no error ** (SQLITE_BUSY or otherwise) is encountered while processing the attached ** databases, SQLITE_OK is returned. ** ** ^{If database zDb is the name of an attached database that is not in WAL ** mode, SQLITE_OK is returned and both *pnLog and *pnCkpt set to -1. ^{If ** zDb is not NULL (or a zero length string) and is not the name of any ** attached database, SQLITE_ERROR is returned to the caller. ** ** ^{Unless it returns SQLITE_MISUSE, ** the sqlite3_wal_checkpoint_v2() interface ** sets the error information that is queried by ** [sqlite3_errcode()] and [sqlite3_errmsg()]. ** ** ^{The [PRAGMA wal_checkpoint] command can be used to invoke this interface ** from SQL.}

71311. ***** Begin reduce actions *****

71312. Number of entries in apSub

71313. Check that the sort worked
71314. ** This function is called as part of a SorterRewind() operation on a sorter ** that has already written two or more level-0 PMAs to one or more temp ** files. It builds a tree of MergeEngine/IncrMerger/PmaReader objects that ** can be used to incrementally merge all PMAs on disk. ** ** If successful, SQLITE_OK is returned and *ppOut set to point to the ** MergeEngine object at the root of the tree before returning. Or, if an ** error occurs, an SQLite error code is returned and the final value ** *ppOut is undefined.
71315. Unknown opcode in sqlite3_file_control()
71316. SQLITE_COUNTOFVIEW_OPTIMIZATION
71317. Write error here if one occurs
71318. Value of apPage[0]->intKey
71319. Output pointer
71320. In debug mode, we mark all persistent databases as sharable ** even when they are not. This exercises the locking code and ** gives more opportunity for asserts(sqlite3_mutex_held()) ** statements to find locking problems.
71321. Allocate or grow the PendingList as required.
71322. Next frame to read from wal file
71323. The 3-byte case
71324. 410
71325. Value is a VdbeFrame object
71326. COMMIT
71327. Set defaults for non-supported filesystems
71328. Table handle
71329. ** 2001 September 15 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This is the implementation of the page cache subsystem or "pager".
** ** The pager is used to access a database disk file. It implements ** atomic commit and rollback through the use of a journal file that ** is separate from the database file. The pager also implements file ** locking to prevent two processes from writing the same database ** file simultaneously, or one process from reading the database while ** another is writing.
71330. Specific chunk into which cursor points
71331. Buffer to a append to
71332. ** chng_addr_0: ** regPrev(0) = idx(0) ** chng_addr_1: ** regPrev(1) = idx(1) ** ...
71333. Reference to page 1
71334. Array to write results into (see above)
71335. This branch is taken if an error occurs while trying to open ** or roll back a hot-journal while holding an EXCLUSIVE lock. The ** pager_unlock() routine will be called before returning to unlock ** the file. If the unlock attempt fails, then Pager.eLock must be ** set to UNKNOWN_LOCK (see the comment above the #define for ** UNKNOWN_LOCK above for an explanation). *** ** In order to get pager_unlock() to do this, set Pager.eState to ** PAGER_ERROR now. This is not actually counted as a transition ** to ERROR state in the state diagram at the top of this file, ** since we know that the same call to pager_unlock() will very ** shortly transition the pager object to the OPEN state. Calling ** assert_pager_state() would fail now, as it should not be possible ** to be in ERROR state when there are zero outstanding page ** references.
71336. IN/OUT: True after first int written
71337. Size of zPattern
71338. **comment:** For length() and typeof() functions with a column argument, ** set the P5 parameter to the OP_Column opcode to OPFLAG_LENGTHHARG ** or OPFLAG_TYPEOFARG respectively, to avoid unnecessary data ** loading.
label: code-design
71339. Return an integer that records the current (uncommitted) write ** position in the WAL
71340. Language id to use in tokenizing
71341. ** An "etByte" is an 8-bit unsigned value.
71342. Address of the next freeblock
71343. Name context for pParse->pNewTable
71344. schema
71345. Low-level memory allocation interface
71346. Sanity checking: The schema for the trigger and for the table are ** always defined. The trigger must be in the same schema as the table ** or else it must be a TEMP trigger.
71347. Merge engine containing PmaReaders to compare
71348. Page number of pNew
71349. "?" or "_"
71350. !=. Part of = or ==
71351. P4_STATIC or P4_TRANSIENT
71352. Number of cells on page
71353. Floating or exponential, depending on exponent. %g
71354. Status return flags
71355. Pointer to string from which to read char
71356. ** The Pager.eState variable stores the current 'state' of a pager. A ** pager may be in any one of the seven states shown in the following ** state diagram. *** **
OPEN <-----+---+ ** ||| ** V || ** +-----> READER-----+ | ** ||| ** | V | ** |<-----WRITER_LOCKED-----> ERROR ** || ^ ** | V | ** |<-----
WRITER_CACHEMOD-----> | ** ||| ** | V | ** |<-----WRITER_DBMOD-----> | ** ||| ** | V | ** +<-----WRITER_FINISHED----->+ *** *** List
of state transitions and the C [function] that performs each: *** ** OPEN -> READER [sqlite3PagerSharedLock] ** READER -> OPEN [pager_unlock] *** **
READER -> WRITER_LOCKED [sqlite3PagerBegin] ** WRITER_LOCKED -> WRITER_CACHEMOD [pager_open_journal] ** WRITER_CACHEMOD ->
WRITER_DBMOD [syncJournal] ** WRITER_DBMOD -> WRITER_FINISHED [sqlite3PagerCommitPhaseOne] ** WRITER_*** -> READER
[page_end_transaction] *** ** WRITER_*** -> ERROR [pager_error] ** ERROR -> OPEN [pager_unlock] *** *** OPEN: *** ** The pager starts up in this
state. Nothing is guaranteed in this ** state - the file may or may not be locked and the database size is ** unknown. The database may not be read or written. **
*** ** No read or write transaction is active. *** ** Any lock, or no lock at all, may be held on the database file. *** ** The dbSize, dbOrigSize and dbFileSize variables
may not be trusted. *** ** READER: *** ** In this state all the requirements for reading the database in ** rollback (non-WAL) mode are met. Unless the pager is
(or recently ** was) in exclusive-locking mode, a user-level read transaction is ** open. The database size is known in this state. *** ** A connection running with
locking_mode=normal enters this state when ** it opens a read-transaction on the database and returns to state ** OPEN after the read-transaction is completed.
However a connection ** running in locking_mode=exclusive (including temp databases) remains in ** this state even after the read-transaction is closed. The
only way ** a locking_mode=exclusive connection can transition from READER to OPEN ** is via the ERROR state (see below). *** ** A read transaction may
be active (but a write-transaction cannot). *** ** A SHARED or greater lock is held on the database file. *** ** The dbSize variable may be trusted (even if a user-
level read ** transaction is not active). The dbOrigSize and dbFileSize variables ** may not be trusted at this point. *** ** If the database is a WAL database, then
the WAL connection is open. *** ** Even if a read-transaction is not open, it is guaranteed that ** there is no hot-journal in the file-system. *** **
WRITER_LOCKED: *** ** The pager moves to this state from READER when a write-transaction ** is first opened on the database. In WRITER_LOCKED
state, all locks ** required to start a write-transaction are held, but no actual ** modifications to the cache or database have taken place. ** ** In rollback mode, a
RESERVED or (if the transaction was opened with ** BEGIN EXCLUSIVE) EXCLUSIVE lock is obtained on the database file when ** moving to this state, but
the journal file is not written to or opened ** to in this state. If the transaction is committed or rolled back while ** in WRITER_LOCKED state, all that is required
is to unlock the database ** file. *** ** IN WAL mode, WalBeginWriteTransaction() is called to lock the log file. ** If the connection is running with
locking_mode=exclusive, an attempt ** is made to obtain an EXCLUSIVE lock on the database file. *** ** A write transaction is active. *** ** If the connection is
open in rollback-mode, a RESERVED or greater ** lock is held on the database file. *** ** If the connection is open in WAL-mode, a WAL write transaction ** is
open (i.e. sqlite3WalBeginWriteTransaction() has been successfully ** called). *** ** The dbSize, dbOrigSize and dbFileSize variables are all valid. *** ** The
contents of the pager cache have not been modified. *** ** The journal file may or may not be open. *** ** Nothing (not even the first header) has been written to the
journal. *** ** WRITER_CACHEMOD: *** ** A pager moves from WRITER_LOCKED state to this state when a page is ** first modified by the upper layer. In
rollback mode the journal file ** is opened (if it is not already open) and a header written to the ** start of it. The database file on disk has not been modified. ***

*** A write transaction is active. *** A RESERVED or greater lock is held on the database file. *** The journal file is open and the first header has been written ** to it, but the header has not been synced to disk. *** The contents of the page cache have been modified. *** WRITER_DBMOD: *** The pager transitions from WRITER_CACHEMOD into WRITER_DBMOD state ** when it modifies the contents of the database file. WAL connections ** never enter this state (since they do not modify the database file, ** just the log file). *** A write transaction is active. *** An EXCLUSIVE or greater lock is held on the database file. *** The journal file is open and the first header has been written ** and synced to disk. *** The contents of the page cache have been modified (and possibly ** written to disk). *** WRITER_FINISHED: *** It is not possible for a WAL connection to enter this state. *** A rollback-mode pager changes to WRITER_FINISHED state from WRITER_DBMOD ** state after the entire transaction has been successfully written into the ** database file. In this state the transaction may be committed simply ** by finalizing the journal file. Once in WRITER_FINISHED state, it is ** not possible to modify the database further. At this point, the upper ** layer must either commit or rollback the transaction. *** A write transaction is active. *** An EXCLUSIVE or greater lock is held on the database file. *** All writing and syncing of journal and database data has finished. ** If no error occurred, all that remains is to finalize the journal to ** commit the transaction. If an error did occur, the caller will need ** to rollback the transaction. *** ERROR: *** The ERROR state is entered when an IO or disk-full error (including ** SQLITE_IOERR_NOMEM) occurs at a point in the code that makes it ** difficult to be sure that the in-memory pager state (cache contents, ** db size etc.) are consistent with the contents of the file-system. *** Temporary pager files may enter the ERROR state, but in-memory pagers ** cannot. *** For example, if an IO error occurs while performing a rollback, ** the contents of the page-cache may be left in an inconsistent state. ** At this point it would be dangerous to change back to READER state ** (as usually happens after a rollback). Any subsequent readers might ** report database corruption (due to the inconsistent cache), and if ** they upgrade to writers, they may inadvertently corrupt the database ** file. To avoid this hazard, the pager switches into the ERROR state ** instead of READER following such an error. *** Once it has entered the ERROR state, any attempt to use the pager ** to read or write data returns an error. Eventually, once all ** outstanding transactions have been abandoned, the pager is able to ** transition back to OPEN state, discarding the contents of the ** page-cache and any other in-memory state at the same time. Everything ** is reloaded from disk (and, if necessary, hot-journal rollback performed) ** when a read-transaction is next opened on the pager (transitioning ** the pager into READER state). At that point the system has recovered ** from the error. *** Specifically, the pager jumps into the ERROR state if: *** 1. An error occurs while attempting a rollback. This happens in ** function sqlite3PagerRollback(). *** 2. An error occurs while attempting to finalize a journal file ** following a commit in function sqlite3PagerCommitPhaseTwo(). ** 3. An error occurs while attempting to write to the journal or ** database file in function pagerStress() in order to free up ** memory. *** In other cases, the error is returned to the b-tree layer. The b-tree ** layer then attempts a rollback operation. If the error condition ** persists, the pager enters the ERROR state via condition (1) above. *** Condition (3) is necessary because it can be triggered by a read-only ** statement executed within a transaction. In this case, if the error ** code were simply returned to the user, the b-tree layer would not ** automatically attempt a rollback, as it assumes that an error in a ** read-only statement cannot leave the pager in an internally inconsistent ** state. *** The Pager.errCode variable is set to something other than SQLITE_OK. ** There are one or more outstanding references to pages (after the ** last reference is dropped the pager should move back to OPEN state). *** The pager is not an in-memory pager. *** Notes: *** A pager is never in WRITER_DBMOD or WRITER_FINISHED state if the ** connection is open in WAL mode. A WAL connection is always in one ** of the first four states. *** Normally, a connection open in exclusive mode is never in PAGER_OPEN ** state. There are two exceptions: immediately after exclusive-mode has ** been turned on (and before any read or write transactions are ** executed), and when the pager is leaving the "error state". *** See also: assert_pager_state().

71357. **comment:** ** Return a pointer to a buffer containing the term associated with the ** entry that the iterator currently points to.

label: code-design

71358. Next in list of deferred tokens

71359. xCommit - commit transaction

71360. If there are any write-transactions at all, invoke the commit hook

71361. True if aLeft[] contains PK fields only

71362. Saved LIMIT and OFFSET

71363. Address to jump to for next iteration

71364. **comment:** Docid of most recently inserted document

label: documentation

71365. Number of result columns written here

71366. If we get this far, it means that the reader will want to use ** the WAL to get at content from recent commits. The job now is ** to select one of the aReadMark[] entries that is closest to ** but not exceeding pWal->hdr.mxFrame and lock that entry.

71367. Cursor pointing at record to retrieve.

71368. **comment:** Process the ORDER BY clause for singleton SELECT statements. ** The ORDER BY clause for compounds SELECT statements is handled ** below, after all of the result-sets for all of the elements of ** the compound have been resolved. *** If there is an ORDER BY clause on a term of a compound-select other ** than the right-most term, then that is a syntax error. But the error ** is not detected until much later, and so we need to go ahead and ** resolve those symbols on the incorrect ORDER BY for consistency.

label: code-design

71369. Initial value of pParse->pWith

71370. The page being analyzed

71371. Figure out how much space the key will consume if it is written to ** the current node of layer iLayer. Due to the prefix compression, ** the space required changes depending on which node the key is to ** be added to.

71372. ** Change the lock state for a shared-memory segment.

71373. Vdbe

71374. Priority queue for search points

71375. Search TEMP before MAIN

71376. ** Return true if expression pExpr is a vector, or false otherwise. *** A vector is defined as any expression that results in two or more ** columns of result. Every TK_VECTOR node is an vector because the ** parser will not generate a TK_VECTOR with fewer than two entries. ** But a TK_SELECT might be either a vector or a scalar. It is only ** considered a vector if it has two or more result columns.

71377. ** Return the value of the 'auto-vacuum' property. If auto-vacuum is ** enabled 1 is returned. Otherwise 0.

71378. SQLITE_CORE

71379. # include "sqlite3ext.h"

71380. Range constraints on queries

71381. If the magic name ":memory:" will create an in-memory database, then ** leave the autoVacuum mode at 0 (do not auto-vacuum), even if ** SQLITE_DEFAULT_AUTOVACUUM is true. On the other hand, if ** SQLITE OMIT_MEMORYDB has been defined, then ":memory:" is just a ** regular file-name. In this case the auto-vacuum applies as per normal.

71382. as ::=

71383. Data actually written

71384. ***** Begin file sqlite3rbu.c *****

71385. Output should be DISTINCT

71386. True if an incremental I/O handle

71387. Compute successively longer WherePaths using the previous generation ** of WherePaths as the basis for the next. Keep track of the mxChoice ** best paths at each generation

71388. Number of memory registers for sub-program

71389. **comment:** If iLevel is less than 0 and this is not a scan, include a seg-reader ** for the pending-terms. If this is a scan, then this call must be being ** made by an fts4aux module, not an FTS table. In this case calling ** Fts3SegReaderPending might segfault, as the data structures used by ** fts4aux are not completely populated. So it's easiest to filter these ** calls out here.

label: code-design

71390. ***** Begin file mem0.c *****

71391. HAVE_READLINK && HAVE_LSTAT

71392. ABLEFTHENDEFERRABLEELSEXCEPTRANSACTIONNATURALTERAISEXCLUSIVE

71393. ** The parser calls this routine after the CREATE VIRTUAL TABLE statement ** has been completely parsed.

71394. Return a pointer to the allocated memory.

71395. ***** Begin file btreeInt.h *****

71396. The GROUP BY clause. May be NULL

71397. Do whatever the default action is
71398. SQLITE OMIT CTE
71399. If the cache is already full, delete the least recently used entry
71400. ***** Begin file os_setup.h *****
71401. Root of tree
71402. Advance pointer p until it points to pEnd or an 0x01 byte that is ** not part of a varint. Note that it is not possible for a negative ** or extremely large varint to occur within an uncorrupted position ** list. So the last byte of each varint may be assumed to have a clear ** 0x80 bit.
71403. ACTION
71404. Counting column references
71405. For a cursor with the BTREE SEEK_EQ hint, only the OP_SeekGE and ** OP_SeekLE opcodes are allowed, and these must be immediately followed ** by an OP_IdxGT or OP_IdxLT opcode, respectively, with the same key.
71406. Make any required updates to pointer map entries associated with ** cells stored on sibling pages following the balance operation. Pointer ** map entries associated with divider cells are set by the insertCell() ** routine. The associated pointer map entries are: ** ** a) if the cell contains a reference to an overflow chain, the ** entry associated with the first page in the overflow chain, and ** ** b) if the sibling pages are not leaves, the child page associated ** with the cell. ** ** If the sibling pages are not leaves, then the pointer map entry ** associated with the right-child of each sibling may also need to be ** updated. This happens below, after the sibling pages have been ** populated, not here.
71407. The trunk has no leaves and the list is not being searched. ** So extract the trunk page itself and use it as the newly ** allocated page
71408. Test all NOT NULL constraints.
71409. **comment:** Temporary space to use
 label: code-design
71410. The JSON to search
71411. Pointer p now points at the first byte past the varint we are ** interested in. So, unless the doclist is corrupt, the 0x80 bit is ** clear on character p[-1].
71412. The operand, and output
71413. ** Initialize and deinitialize the mutex subsystem.
71414. ** Return the address of the next instruction to be inserted.
71415. Pointer to node data (or NULL)
71416. Database size after this commit
71417. IN_INDEX_LOOP, _MEMBERSHIP, and/or _NOOP_OK
71418. ** The following object holds the list of automatically loaded ** extensions. ** ** This list is shared across threads. The SQLITE_MUTEX_STATIC_MASTER ** mutex must be held while accessing this list.
71419. ** Input "r" is a numeric quantity which might be a Julian day number, ** or the number of seconds since 1970. If the value of r is within ** range of a Julian day number, install it as such and set validJD. ** If the value is a valid unix timestamp, put it in p->s and set p->rawS.
71420. xUpdate - write data
71421. Text of the SQL statement
71422. Parent context (if any)
71423. 940
71424. transtype ::= IMMEDIATE
71425. ** Added for 3.6.0
71426. If not NULL: boolean for each table and index
71427. Set the BtLock.eLock variable to the maximum of the current lock ** and the requested lock. This means if a write-lock was already held ** and a read-lock requested, we don't incorrectly downgrade the lock.
71428. 65
71429. SQLITE_PERM
71430. 8
71431. Number of bytes actually read from file
71432. Forward declaration
71433. OUT: Result
71434. Normally, if a transaction is rolled back, any backup processes are ** updated as data is copied out of the rollback journal and into the ** database. This is not generally possible with a WAL database, as ** rollback involves simply truncating the log file. Therefore, if one ** or more frames have already been written to the log (and therefore ** also copied into the backup databases) as part of this transaction, ** the backups must be restarted.
71435. nCol*nPhrase values
71436. (279) trans_opt ::= TRANSACTION
71437. VIEW => ID
71438. ** The following set of routines walk through the parse tree and assign ** a specific database to all table references where the database name ** was left unspecified in the original SQL statement. The pFix structure ** must have been initialized by a prior call to sqlite3FixInit(). ** ** These routines are used to make sure that an index, trigger, or ** view in one database does not refer to objects in a different database. ** (Exception: indices, triggers, and views in the TEMP database are ** allowed to refer to anything.) If a reference is explicitly made ** to an object in a different database, an error message is added to ** pParse->zErrMsg and these routines return non-zero. If everything ** checks out, these routines return 0.
71439. Next in hash table chain
71440. Content is text with \ escapes
71441. DELETE operations
71442. Number of times lookaside has been disabled
71443. **comment:** ** The following array of hash tables is used to buffer pending index ** updates during transactions. All pending updates buffered at any one ** time must share a common language-id (see the FTS4 langid= feature). ** The current language id is stored in variable iPrevLangid. ** ** A single FTS4 table may have multiple full-text indexes. For each index ** there is an entry in the aIndex[] array. Index 0 is an index of all the ** terms that appear in the document set. Each subsequent index in aIndex[] ** is an index of prefixes of a specific length. ** ** Variable nPendingData contains an estimate the memory consumed by the ** pending data structures, including hash table overhead, but not including ** malloc overhead. When nPendingData exceeds nMaxPendingData, all hash ** tables are flushed to disk. Variable iPrevDocid is the docid of the most ** recently inserted record.
 label: code-design
71444. ** This function returns non-zero if the specified UTF-8 string buffer ** ends with a directory separator character or one was successfully ** added to it.
71445. Verify that the call to _bytes() does not invalidate the _text() pointer
71446. OUT: *pzToken is the token text
71447. limit_opt ::= LIMIT expr OFFSET expr
71448. If the SQL column is blank it means this is an index that ** was created to be the PRIMARY KEY or to fulfill a UNIQUE ** constraint for a CREATE TABLE. The index should have already ** been created when we processed the CREATE TABLE. All we have ** to do here is record the root page number for that index.
71449. Read-only system table
71450. VM being constructed
71451. ** Open a new Fts5Index handle. If the bCreate argument is true, create ** and initialize the underlying %_data table. ** ** If successful, set *pp to point to the new object and return SQLITE_OK. ** Otherwise, set *pp to NULL and return an SQLite error code.
71452. ** Obtain the mutex p. If some other thread already has the mutex, block ** until it can be obtained.
71453. ***** Begin file hash.c *****
71454. Text encoding - one of the SQLITE_UTF* values
71455. ** Compute a 32-bit checksum on the N-byte buffer. Return the result.
71456. Fixer object
71457. Suspicious pointer arithmetic
71458. The X expression
71459. Opcode: IdxGE P1 P2 P3 P4 P5 ** Synopsis: key=r[P3@P4] ** ** The P4 register values beginning with P3 form an unpacked index ** key that omits the PRIMARY KEY. Compare this key value against the index ** that P1 is currently pointing to, ignoring the PRIMARY KEY or ROWID ** fields at the end. ** **

If the P1 index entry is greater than or equal to the key value ** then jump to P2. Otherwise fall through to the next instruction.

71460. ** Read a single varint from the doclist at *pp and advance *pp to point ** to the first byte past the end of the varint. Add the value of the varint ** to *pVal.

71461. Allocate space for a new sqlite3_backup object... ** EVIDENCE-OF: R-64852-21591 The sqlite3_backup object is created by a ** call to sqlite3_backup_init() and is destroyed by a call to ** sqlite3_backup_finish().

71462. selcollist ::= sclp expr as

71463. Column name

71464. ** This function is called as the second part of each xNext operation when ** iterating through the results of a full-text query. At this point the ** cursor points to a row that matches the query expression, with the ** following caveats: ** ** * Up until this point, "NEAR" operators in the expression have been ** treated as "AND". ** ** * Deferred tokens have not yet been considered. ** ** If *pRc is not SQLITE_OK when this function is called, it immediately ** returns 0. Otherwise, it tests whether or not after considering NEAR ** operators and deferred tokens the current row is still a match for the ** expression. It returns 1 if both of the following are true: ** ** 1. *pRc is SQLITE_OK when this function returns, and ** ** 2. After scanning the current FTS table row for the deferred tokens, ** it is determined that the row does *not* match the query. ** ** Or, if no error occurs and it seems the current row does match the FTS ** query, return 0.

71465. sign * significand * (10 ^ (esign * exponent))

71466. Use a new PGroup for each PCache

71467. Object describing the transient index

71468. Used to verify lookaside not used for schema

71469. Opcode: RowSetTest P1 P2 P3 P4 ** Synopsis: if r[P3] in rowset(P1) goto P2 ** ** Register P3 is assumed to hold a 64-bit integer value. If register P1 ** contains a RowSet object and that RowSet object contains ** the value held in P3, jump to register P2. Otherwise, insert the ** integer in P3 into the RowSet and continue on to the ** next opcode. ** ** The RowSet object is optimized for the case where sets of integers ** are inserted in distinct phases, which each set contains no duplicates. ** Each set is identified by a unique P4 value. The first set ** must have P4==0, the final set must have P4==1, and for all other sets ** must have P4>0. ** ** This allows optimizations: (a) when P4==0 there is no need to test ** the RowSet object for P3, as it is guaranteed not to contain it, ** (b) when P4==1 there is no need to insert the value, as it will ** never be tested for, and (c) when a value that is part of set X is ** inserted, there is no need to search to see if the same value was ** previously inserted as part of set X (only if it was previously ** inserted as part of some other set).

71470. 303

71471. Token identifying trigger

71472. Step 5a

71473. ***** Begin file memjournal.c *****

71474. ** Free an allocated buffer obtained from pcache1Alloc().

71475. Make sure the output register has a buffer large enough to store ** the new record. The output register (pOp->p3) is not allowed to ** be one of the input registers (because the following call to ** sqlite3VdbeMemClearAndResize() could clobber the value before it is used).

71476. ** Release a MemPage. This should be called once for each prior ** call to btreeGetPage.

71477. The affinity to be applied

71478. **comment:** ** Change the page size used by the Pager object. The new page size ** is passed in *pPageSize. ** ** If the pager is in the error state when this function is called, it ** is a no-op. The value returned is the error state error code (i.e. ** one of SQLITE_IOERR, an SQLITE_IOERR_xxx sub-code or SQLITE_FULL). ** ** Otherwise, if all of the following are true: ** ** * the new page size (value of *pPageSize) is valid (a power ** of two between 512 and SQLITE_MAX_PAGE_SIZE, inclusive), and ** ** * there are no outstanding page references, and ** ** * the database is either not an in-memory database or it is ** an in-memory database that currently consists of zero pages. ** ** then the pager object page size is set to *pPageSize. ** ** If the page size is changed, then this function uses sqlite3PagerMalloc() ** to obtain a new Pager.pTmpSpace buffer. If this allocation attempt ** fails, SQLITE_NOMEM is returned and the page size remains unchanged. ** In all other cases, SQLITE_OK is returned. ** ** If the page size is not changed, either because one of the enumerated ** conditions above is not true, the pager was in error state when this ** function was called, or because the memory allocation attempt failed, ** then *pPageSize is set to the old, retained page size before returning.

label: code-design

71479. Ticket #3810. ** Normally, whenever a table is dropped, all associated triggers are ** dropped too. But if a TEMP trigger is created on a non-TEMP table ** and the table is dropped by a different database connection, the ** trigger is not visible to the database connection that does the ** drop so the trigger cannot be dropped. This results in an ** "orphaned trigger" - a trigger whose associated table is missing.

71480. eidlist ::= nm collate sortorder

71481. This table's current rowid value

71482. Append the name of the path for element i to pStr

71483. True if a temp table was used

71484. d8..df

71485. Cursor object

71486. ** Check the internal RTree node given by pCellData against constraint p. ** If this constraint cannot be satisfied by any child within the node, ** set *peWithin to NOT_WITHIN.

71487. ** Take or release a shared-memory lock.

71488. x IS NULL

71489. IMP: R-04460-53386

71490. ** Zero all counters associated with the sqlite3_stmt_scanstatus() data.

71491. The value of the "rank" column.

71492. **comment:** ** Return TRUE if all of the following are true: ** ** (1) X has the same or lower cost than Y ** (2) X is a proper subset of Y ** (3) X skips at least as many columns as Y ** ** By "proper subset" we mean that X uses fewer WHERE clause terms ** than Y and that every WHERE clause term used by X is also used ** by Y. ** ** If X is a proper subset of Y then Y is a better choice and ought ** to have a lower cost. This routine returns TRUE when that cost ** relationship is inverted and needs to be adjusted. The third rule ** was added because if X uses skip-scan less than Y it still might ** deserve a lower cost even if it is a proper subset of Y.

label: code-design

71493. True to use a bg thread for this object

71494. Write the contents of the page out to the database file.

71495. Opcode: VNext P1 P2 * * * * Advance virtual table P1 to the next row in its result set and ** jump to instruction P2. Or, if the virtual table has reached ** the end of its result set, then fall through to the next instruction.

71496. Write a frame or frames to the log.

71497. ** This routine is a no-op if the database schema is already initialized. ** Otherwise, the schema is loaded. An error code is returned.

71498. Cannot insert into a read-only table.

71499. Copy of sixth arg to _apply()

71500. A single database

71501. Neither the RHS or LHS are deferred.

71502. Cursor is pointing at the last entry

71503. "PRAGMA index_list" includes the main PK b-tree

71504. For looping through nTerm phrase terms

71505. Only truncate if it is an in-memory sub-journal.

71506. Never gets this far otherwise

71507. Justification of ALWAYS(): The xConstructor method is required to ** create a valid sqlite3_vtab if it returns SQLITE_OK.

71508. Unpacked version of aRecord[]

71509. **comment:** INSTEAD OF triggers can only appear on views and BEFORE triggers ** cannot appear on views. So we might as well translate every ** INSTEAD OF trigger into a BEFORE trigger. It simplifies code ** elsewhere.

label: code-design

71510. ** Return the number of values available from the current row of the ** currently executing statement pStmt.

71511. "SELECT *" statement on zTbl

71512. 1 for STAT3 or STAT4. 0 otherwise

71513. Number of RBU VFS in the stack

71514. ** Delete any previous value and set the value stored in *pMem to val, ** manifest type INTEGER.
71515. The jrnEnc flag is true if Journal pages should be passed through ** the codec. It is false for pure in-memory journals.
71516. ** Do the work for either sqlite3changeset_start() or start_strm().
71517. file-mapping handle
71518. Skip over the 'height' varint that occurs at the start of every ** interior node. Then load the blockid of the left-child of the b-tree ** node into variable iChild. ***
Even if the data structure on disk is corrupted, this (reading two ** varints from the buffer) does not risk an overread. If zNode is a ** root node, then the buffer comes from a SELECT statement. SQLite does ** not make this guarantee explicitly, but in practice there are always ** either more than 20 bytes of allocated space following the nNode bytes of ** contents, or two zero bytes. Or, if the node is read from the %_segments ** table, then there are always 20 bytes of zeroed padding following the ** nNode bytes of content (see sqlite3Fts3ReadBlock() for details).
71519. TUNING: Minimum for skip-scan
71520. Id of this connection with its winShmNode
71521. ** Set the auxiliary data pointer and delete function, for the iArg'th ** argument to the user-function defined by pCtx. Any previous value is ** deleted by calling the delete function specified when it was set. *** ** The left-most argument is 0. *** Undocumented behavior: If iArg is negative then make the data available ** to all functions within the current prepared statement using iArg as an ** access code.
71522. Opcode: IfNot P1 P2 P3 * * *** Jump to P2 if the value in register P1 is False. The value ** is considered false if it has a numeric value of zero. If the value ** in P1 is NULL then take the jump if and only if P3 is non-zero.
71523. Right-hand Fts5SegIter
71524. 550
71525. tab2 must not be a virtual table
71526. **comment:** *** This function - unixLogErrorAtLine(), is only ever called via the macro ** unixLogError(). *** It is invoked after an error occurs in an OS function and errno has been ** set. It logs a message using sqlite3_log() containing the current value of ** errno and, if possible, the human-readable equivalent from strerror() or ** strerror_r(). *** The first argument passed to the macro should be the error code that ** will be returned to SQLite (e.g. SQLITE_IOERR_DELETE, SQLITE_CANTOPEN). ** The two subsequent arguments should be the name of the OS function that ** failed (e.g. "unlink", "open") and the associated file-system path, ** if any.
label: code-design
71527. sqlite3GetFuncCollSeq() might be called
71528. Make sure at least the first p2+1 entries of the header have been ** parsed and valid information is in aOffset[] and pC->aType[].
71529. ** Macros to determine the number of bytes required by a normal Expr ** struct, an Expr struct with the EP_Reduced flag set in Expr.flags ** and an Expr struct with the EP_TokenOnly flag set.
71530. Want a reserved lock?
71531. 71
71532. ** The near-set object passed as the first argument contains more than ** one phrase. All phrases currently point to the same row. The ** Fts5ExprPhrase.plist buffers are populated accordingly. This function ** tests if the current row contains instances of each phrase sufficiently ** close together to meet the NEAR constraint. Non-zero is returned if it ** does, or zero otherwise. *** If in/out parameter (*pRc) is set to other than SQLITE_OK when this ** function is called, it is a no-op. Or, if an error (e.g. SQLITE_NOMEM) ** occurs within this function (*pRc) is set accordingly before returning. ** The return value is undefined in both these cases. *** If no error occurs and non-zero (a match) is returned, the position-list ** of each phrase object is edited to contain only those entries that ** meet the constraint before returning.
71533. ** Read and return an unsigned 32-bit big-endian integer from the buffer ** passed as the only argument.
71534. Structure object to return
71535. Bytes of valid data so far
71536. ** Advance an iterator created by sqlite3changeset_start() to the next ** change in the changeset. This function may return SQLITE_ROW, SQLITE_DONE ** or SQLITE_CORRUPT. *** This function may not be called on iterators passed to a conflict handler ** callback by changeset_apply().
71537. File descriptor for main journal
71538. ** Create a new VFS file descriptor (stored in memory obtained from ** sqlite3_malloc) and open the file named "path" in the file descriptor. *** ** The caller is responsible not only for closing the file descriptor ** but also for freeing the memory associated with the file descriptor.
71539. rank MATCH ? expression (or NULL)
71540. ** This function is used by test procedures to inspect the internal state ** of the global cache.
71541. && !defined(SQLITE OMIT _SUBQUERY)
71542. **comment:** Mask of FTSS5INDEX_XXX flags
label: code-design
71543. Search for a table and column that appears on one side or the ** other of the == operator in every subterm. That table and column ** will be recorded in iCursor and iColumn. There might not be any ** such table and column. Set okToChngToIN if an appropriate table ** and column is found but leave okToChngToIN false if not found.
71544. 284
71545. the count using a callback.
71546. Values for up to two %d fields in zPfx
71547. * The value used with sqlite3_win32_set_directory() to specify that * the data directory should be changed.
71548. ** Create an sqlite3_backup process to copy the contents of zSrcDb from ** connection handle pSrcDb to zDestDb in pDestDb. If successful, return ** a pointer to the new sqlite3_backup object. *** If an error occurs, NULL is returned and an error code and error message ** stored in database handle pDestDb.
71549. Always analyze if any index lacks statistics
71550. ** Pragma virtual table module xEof method.
71551. **comment:** ** This macro is used by various functions that merge doclists. The two ** arguments are 64-bit docid values. If the value of the stack variable ** bDescDoclist is 0 when this macro is invoked, then it returns (i1-i2). ** Otherwise, (i2-i1). *** Using this makes it easier to write code that can merge doclists that are ** sorted in either ascending or descending order.
label: code-design
71552. Values to return.
71553. ** Advance the iterator passed as the only argument. If the end of the ** doclist-index page is reached, return non-zero.
71554. Name of PK index
71555. Now search for a zipvfs instance lower down in the VFS stack. If ** one is found, this is an error.
71556. ** HAVE_MREMAP defaults to true on Linux and false everywhere else.
71557. Size of each buffer in bytes
71558. Do not use an automatic index if the this loop is expected ** to run less than 2 times.
71559. For looping over expression in a list
71560. SQLITE OMIT _DECLTYPE
71561. If we reach this point, it means that pExpr refers to a table ** that is in the FROM clause of the aggregate query. *** ** Make an entry for the column in pAggInfo->aCol[] if there ** is not an entry there already.
71562. _FTS5_H
71563. **comment:** These registers need to be preserved in case there is an IN operator ** loop. So we could deallocate the registers here (and potentially ** reuse them later) if (pLoop->wsFlags & WHERE_IN_ABLE)==0. But it seems ** simpler and safer to simply not reuse the registers. *** ** sqlite3ReleaseTempRange(pParse, iReg, nConstraint+2);
label: code-design
71564. Estimated rows in table - from sqlite_stat1 table
71565. Column currently being processed
71566. 19
71567. Used by fstat()
71568. ** If parameter iCol is not 0, write an POS_COLUMN (1) byte followed by ** the value of iCol encoded as a varint to *pp. This will start a new ** column list. **
** Set *pp to point to the byte just after the last byte written before ** returning (do not modify it if iCol==0). Return the total number of bytes ** written (0 if iCol==0).

71569. If leaving WAL mode, close the log file. If successful, the call ** to PagerCloseWal() checkpoints and deletes the write-ahead-log ** file. An EXCLUSIVE lock may still be held on the database file ** after a successful return.

71570. 62

71571. The complete WAL information

71572. Refcount for pOuterNC and outer contexts

71573. Need convertCompoundSelectToSubquery()

71574. If there is not already a read-only (or read-write) transaction opened ** on the b-tree database, open one now. If a transaction is opened, it ** will be closed before this function returns.

71575. Next entry is on the next page

71576. If the SF_Converted flags is set, then this Select object was ** was created by the convertCompoundSelectToSubquery() function. ** In this case the ORDER BY clause (p->pOrderBy) should be resolved ** as if it were part of the sub-query, not the parent. This block ** moves the pOrderBy down to the sub-query. It will be moved back ** after the names have been resolved.

71577. ** Finished with one layer of the tree

71578. (298) constlist ::= tcons (OPTIMIZED OUT)

71579. ** The sqlite3_mutex_enter() and sqlite3_mutex_try() routines attempt ** to enter a mutex. If another thread is already within the mutex, ** sqlite3_mutex_enter() will block and sqlite3_mutex_try() will return ** SQLITE_BUSY. The sqlite3_mutex_try() interface returns SQLITE_OK ** upon successful entry. Mutexes created using SQLITE_MUTEX_RECURSIVE can ** be entered multiple times by the same thread. In such cases the, ** mutex must be exited an equal number of times before another thread ** can enter. If the same thread tries to enter any other kind of mutex ** more than once, the behavior is undefined.

71580. a: p4<<29 | p6<<15 | p8 (unmasked)

71581. ** Reset a sorting cursor back to its original empty state.

71582. Return true if value x is found any of the first nCol entries of aiCol[]

71583. Check if the root node now has exactly one child. If so, remove ** it, schedule the contents of the child for reinsertion and ** reduce the tree height by one. *** This is equivalent to copying the contents of the child into ** the root node (the operation that Gutman's paper says to perform ** in this scenario).

71584. This block is really an inlined version of sqlite3VdbeMemRelease() ** that takes advantage of the fact that the memory cell value is ** being set to NULL after releasing any dynamic resources. *** The justification for duplicating code is that according to ** callgrind, this causes a certain test case to hit the CPU 4.7 ** percent less (x86 linux, gcc version 4.1.2, -O6) than if ** sqlite3MemRelease() were called from here. With -O2, this jumps ** to 6.6 percent. The test case is inserting 1000 rows into a table ** with no indexes using a single prepared INSERT statement, bind() ** and reset(). Inserts are grouped into a transaction.

71585. Parse string

71586. ** This function compares the two table rows or index records ** specified by {nKey1, pKey1} and pPKey2. It returns a negative, zero ** or positive integer if key1 is less than, equal to or ** greater than key2. The {nKey1, pKey1} key must be a blob ** created by the OP_MakeRecord opcode of the VDBE. The pPKey2 ** key must be a parsed key such as obtained from ** sqlite3VdbeParseRecord. *** If argument bSkip is non-zero, it is assumed that the caller has already ** determined that the first fields of the keys are equal. *** Key1 and Key2 do not have to contain the same number of fields. If all ** fields that appear in both keys are equal, then pPKey2->default_rc is ** returned. *** If database corruption is discovered, set pPKey2->errCode to ** SQLITE_CORRUPT and return 0. If an OOM error is encountered, ** pPKey2->errCode is set to SQLITE_NOMEM and, if it is not NULL, the ** malloc-failed flag set on database handle (pPKey2->pKeyInfo->db).

71587. Do not create a trigger on a system table

71588. Used to accumulate return value

71589. IMP: R-14450-37597

71590. ifdef SQLITE_DEBUG

71591. **comment:** Ensure adequate test coverage
label: test

71592. defer_subclause ::= DEFERRABLE init_deferred_pred_opt

71593. colset ::= STRING

71594. Schema name of database to vacuum

71595. Delete the Table structure itself.

71596. Busy handler for eMode2

71597. Jump here if the results are unknown due to NULLs

71598. Test variable that can be set to enable WHERE tracing

71599. WITH

71600. xBegin - begin transaction

71601. Table hold the row

71602. idlist_opt ::= LP idlist RP

71603. All output needs to be distinct

71604. Zero-based index of first term.

71605. Number of bytes to write

71606. Number of terms written to node so far

71607. Desired number of arguments. (-1)==any

71608. Next byte to write in WAL file

71609. ***** Test and Debug Logic *****

71610. SQLITE_STMTSTATUS_MEMUSED

71611. The expression parse tree

71612. Raw cell content as appears on disk

71613. If this is not a view, open the table and and all indices

71614. Pointer to first in list of synonyms

71615. ** Append a new table name to the given SrcList. Create a new SrcList if ** need be. A new entry is created in the SrcList even if pTable is NULL. *** A SrcList is returned, or NULL if there is an OOM error. The returned ** SrcList might be the same as the SrcList that was input or it might be ** a new one. If an OOM error does occurs, then the prior value of pList ** that is input to this routine is automatically freed. *** If pDatabase is not null, it means that the table has an optional ** database name prefix. Like this: "database.table". The pDatabase ** points to the table name and the pTable points to the database name. ** The SrcList.a[].zName field is filled with the table name which might ** come from pTable (if pDatabase is NULL) or from pDatabase. ** SrcList.a[].zDatabase is filled with the database name from pTable, ** or with NULL if no database is specified. *** In other words, if call like this: ** ** sqlite3SrcListAppend(D,A,B,0); ** ** Then B is a table name and the database name is unspecified. If called ** like this: ** ** sqlite3SrcListAppend(D,A,B,C); ** ** Then C is the table name and B is the database name. If C is defined ** then so is B. In other words, we never have a case where: ** ** sqlite3SrcListAppend(D,A,0,C); ** ** Both pTable and pDatabase are assumed to be quoted. They are dequoted ** before being added to the SrcList.

71616. Name of the table to add to the FROM clause

71617. True to disable the incremental doclist optimization. This is controled ** by special insert command 'test-no-incr-doclist'.

71618. Extend the Fts5Structure object as required to ensure the output ** segment exists.

71619. Offset of space for 4-byte poslist size

71620. If there is no "=MODE" part of the pragma, do a query for the ** current mode

71621. Check if this phrase descends from an OR expression node. If not, ** return NULL. Otherwise, the entry that corresponds to docid ** pCsr->iPrevId may lie earlier in the doclist buffer. Or, if the ** tree that the node is part of has been marked as EOF, but the node ** itself is not EOF, then it may point to an earlier entry.

71622. **comment:** The "yyy" in the name "xxx.yyy"
label: code-design

71623. aKWOOffset[i] is the index into zKWText[] of the start of ** the text for the i-th keyword.

71624. ** A value for VdbeCursor.cacheStatus that means the cache is always invalid.

71625. Name of the file used for SHM

71626. Scan for a vowel

71627. ** Values that may be OR'd together to form the argument to the ** BTREE_HINT_FLAGS hint for sqlite3BtreeCursorHint(): *** The BTREE_BULKLOAD flag is set on index cursors when the index is going ** to be filled with content that is already in sorted order. *** The BTREE_SEEK_EQ flag is set on cursors

that will get OP_SeekGE or ** OP_SeekLE opcodes for a range search, but where the range of entries ** selected will all have the same key. In other words, the cursor will ** be used only for equality key searches. **

71628. Original value of pNew->nOut

71629. Pages per sector for pTargetFd

71630. The operand

71631. File name for write-ahead log

71632. Was sqlite3_global_recover(), but that function is deprecated

71633. __cplusplus

71634. Fix the database pointer on page iPtrPage that pointed at iDbPage so ** that it points at iFreePage. Also fix the pointer map entry for ** iPtrPage.

71635. ** This routine is called if the collation factory fails to deliver a ** collation function in the best encoding but there may be other versions ** of this collation function (for other text encodings) available. Use one ** of these instead if they exist. Avoid a UTF-8 <-> UTF-16 conversion if ** possible.

71636. Bitmap representation

71637. TK_COLUMN: cursor number of table holding column ** TK_REGISTER: register number ** TK_TRIGGER: 1 -> new, 0 -> old ** EP_Unlikely: 134217728 times likelihood ** TK_SELECT: 1st register of result vector

71638. Now that there is a read-lock on the source database, query the ** source pager for the number of pages in the database.

71639. ** Compare the values currently indicated by the two nodes as follows: ** ** res = (*p1) - (*p2) ** ** Nodes that point to values that come later in the iteration order are ** considered to be larger. Nodes at EOF are the largest of all. ** ** This means that if the iteration order is ASC, then numerically larger ** rowids are considered larger. Or if it is the default DESC, numerically ** smaller rowids are larger.

71640. 237

71641. ** Open an Fts3MultiSegReader to scan the doclist for term zTerm/nTerm. Or, ** if isPrefix is true, to scan the doclist for all terms for which ** zTerm/nTerm is a prefix. If successful, return SQLITE_OK and write ** a pointer to the new Fts3MultiSegReader to *ppSegcsr. Otherwise, return ** an SQLite error code. ** ** It is the responsibility of the caller to free this object by eventually ** passing it to fts3SegReaderCursorFree() ** ** SQLITE_OK is returned if no error occurs, otherwise an SQLite error code. ** Output parameter *ppSegcsr is set to 0 if an error occurs.

71642. Size of the header in the record

71643. Current offset in a[]

71644. Whether the function call succeeded or failed, set the output parameters ** to whatever their local counterparts contain. If an error did occur, ** this has the effect of zeroing all output parameters.

71645. Error code returned by ICU function

71646. The index type

71647. True if uses a statement journal

71648. Write the master journal data to the end of the journal file. If ** an error occurs, return the error code to the caller.

71649. The BTree

71650. Statement requested from fts3SqlStmt()

71651. Number of terms in the result set

71652. Number of new slots to add to pSrc->a[]

71653. ** Change the page number of page p to newPgno.

71654. label B:

71655. The following block stores the meta information that will be returned ** to the caller in local variables zDataType, zCollSeq, notnull, primarykey ** and autoinc. At this point there are two possibilities: ** ** 1. The specified column name was "rowid", "oid" or "_rowid_" ** and there is no explicitly declared IPK column. ** ** 2. The table is not a view and the column name identified an ** explicitly declared column. Copy meta information from *pCol.

71656. ** Interpret the given string as a locking mode value.

71657. A MATCH operator or equivalent

71658. A malloc may have failed inside of the xFunc() call. If this ** is the case, clear the mallocFailed flag and return NULL.

71659. Source b-tree file

71660. One of TK_DELETE, TK_UPDATE, TK_INSERT

71661. A record being sorted

71662. Move a page here to make room for the root-page

71663. Number of bytes to allocate for iterator

71664. OUT: Total number of pages cached

71665. Opcode: Halt P1 P2 * P4 P5 ** ** Exit immediately. All open cursors, etc are closed ** automatically. ** ** P1 is the result code returned by sqlite3_exec(), sqlite3_reset(), ** or sqlite3_finalize(). For a normal halt, this should be SQLITE_OK (0). ** For errors, it can be some other value. If P1!=0 then P2 will determine ** whether or not to rollback the current transaction. Do not rollback ** if P2==OE_Fail. Do the rollback if P2==OE_Rollback. If P2==OE_Abort, ** then back out all changes that have occurred during this execution of the ** VDBE, but do not rollback the transaction. ** ** If P4 is not null then it is an error message string. ** ** P5 is a value between 0 and 4, inclusive, that modifies the P4 string. ** ** 0: (no change) ** 1: NOT NULL constraint failed: P4 ** 2: UNIQUE constraint failed: P4 ** 3: CHECK constraint failed: P4 ** 4: FOREIGN KEY constraint failed: P4 ** ** If P5 is not zero and P4 is NULL, then everything after the ":" is ** omitted. ** ** There is an implied "Halt 0 0 0" instruction inserted at the very end of ** every program. So a jump past the last instruction of the program ** is the same as executing Halt.

71666. Hash table to query

71667. True if constraint checking is deferred till COMMIT

71668. ** Get a page from the pager and initialize it. ** ** If pCur!=0 then the page is being fetched as part of a moveToChild() ** call. Do additional sanity checking on the page in this case. ** And if the fetch fails, this routine must decrement pCur->iPage. ** ** The page is fetched as read-write unless pCur is not NULL and is ** a read-only cursor. ** ** If an error occurs, then *ppPage is undefined. It ** may remain unchanged, or it may be set to an invalid value.

71669. Create a copy of the lock path if the conch is taken

71670. ***** Begin file legacy.c *****

71671. True if the EXPLAIN flag is found on the query

71672. EVIDENCE-OF: R-06866-39125 Freeblocks are always connected in order of ** increasing offset.

71673. ** Increment the schema cookie of the main database opened by p->dbMain. ** ** Or, if this is an RBU vacuum, set the schema cookie of the main db ** opened by p->dbMain to one more than the schema cookie of the main ** db opened by p->dbRbu.

71674. ***** End of mem5.c *****

71675. Skip over any whitespace before checking for a keyword, an open or ** close bracket, or a quoted string.

71676. OUT: Mask of WhereLoops to run in reverse order

71677. Locate the PRIMARY KEY index. Or, if this table was originally ** an INTEGER PRIMARY KEY table, create a new PRIMARY KEY index.

71678. Next element in list

71679. Set the current transaction state to TRANS_NONE and unlock the ** pager if this call closed the only read or write transaction.

71680. Approximate page size used in %_data

71681. Error code returned by the function.

71682. child table cols -> parent key cols

71683. Special processing to add months

71684. ** Add the list of function arguments to the SrcList entry for a ** table-valued-function.

71685. P4 is a vector of 32-bit integers

71686. Store integers here

71687. ***** End of mutex_w32.c *****

71688. Do the commit only if all databases successfully complete phase 1. ** If one of the BtreeCommitPhaseOne() calls fails, this indicates an ** IO error while deleting or truncating a journal file. It is unlikely, ** but could happen. In this case abandon processing and return the error.

71689. If the translation is between UTF-16 little and big endian, then ** all that is required is to swap the byte order. This case is handled ** differently from the others.

71690. '>'. Part of > or >=

71691. ** This structure encapsulates a user-function destructor callback (as ** configured using create_function_v2()) and a reference counter. When ** create_function_v2() is called to create a function with a destructor, ** a single object of this type is allocated. FuncDestructor.nRef is set to ** the number of

FuncDef objects created (either 1 or 3, depending on whether ** or not the specified encoding is SQLITE_ANY). The FuncDef.pDestructor *** member of each of the new FuncDef objects is set to point to the allocated ** FuncDestructor. *** *** Thereafter, when one of the FuncDef objects is deleted, the reference ** count on this object is decremented. When it reaches 0, the destructor ** is invoked and the FuncDestructor structure freed.

71692. Mem.z points to an ephemeral string

71693. **comment:** Database connection (for malloc)

label: code-design

71694. Use the patchset format if true

71695. ** Return true if the expression contains no non-deterministic SQL ** functions. Do not consider non-deterministic SQL functions that are ** part of sub-select statements.

71696. Values between 2097152 and 268435455

71697. Version 3.12.0 and later

71698. Base class used by SQLite core

71699. Index in pInfo->aConstraint[]

71700. **comment:** ** Generate code that will assemble an index key and stores it in register ** regOut. The key with be for index pIdx which is an index on pTab. ** iCur is the index of a cursor open on the pTab table and pointing to ** the entry that needs indexing. If pTab is a WITHOUT ROWID table, then ** iCur must be the cursor of the PRIMARY KEY index. *** ** Return a register number which is the first in a block of ** registers that holds the elements of the index key. The ** block of registers has already been deallocated by the time ** this routine returns. *** ** If *piPartIdxLabel is not NULL, fill it in with a label and jump ** to that label if pIdx is a partial index that should be skipped. ** The label should be resolved using sqlite3ResolvePartIdxLabel(). ** A partial index should be skipped if its WHERE clause evaluates ** to false or null. If pIdx is not a partial index, *piPartIdxLabel ** will be set to zero which is an empty label that is ignored by ** sqlite3ResolvePartIdxLabel(). *** ** The pPrior and regPrior parameters are used to implement a cache to ** avoid unnecessary register loads. If pPrior is not NULL, then it is ** a pointer to a different index for which an index key has just been ** computed into register regPrior. If the current pIdx index is generating ** its key into the same sequence of registers and if pPrior and pIdx share ** a column in common, then the register corresponding to that column already ** holds the correct value and the loading of that register is skipped. ** This optimization is helpful when doing a DELETE or an INTEGRITY_CHECK ** on a table with multiple indices, and especially with the ROWID or ** PRIMARY KEY columns of the index.

label: code-design

71701. ***** Continuing where we left off in os_common.h *****

71702. ** Enter a mutex on a Btree given a cursor owned by that Btree. *** ** These entry points are used by incremental I/O only. Enter() is required ** any time OMIT_SHARED_CACHE is not defined, regardless of whether or not ** the build is threadsafe. Leave() is only required by threadsafe builds.

71703. The schema has been loaded

71704. 121

71705. Page content

71706. Number of queued entries by iLevel

71707. FLOA

71708. At this point, there is a write transaction open on both the ** vacuum database and the main database. Assuming no error occurs, ** both transactions are closed by this block - the main database ** transaction by sqlite3BtreeCopyFile() and the other by an explicit ** call to sqlite3BtreeCommit().

71709. Number of nested calls to the parser/code generator

71710. expr ::= expr in_op LP select RP

71711. ** Helper functions to obtain and relinquish the global mutex. The ** global mutex is used to protect the winLockInfo objects used by ** this file, all of which may be shared by multiple threads. *** ** Function winShmMutexHeld() is used to assert() that the global mutex ** is held when required. This function is only used as part of assert() ** statements. e.g. *** winShmEnterMutex() ** assert(winShmMutexHeld()); ** winShmLeaveMutex()

71712. pSelect is correlated, not constant

71713. **comment:** The correct SQL-92 behavior is for the LIKE operator to ignore ** case. Thus 'a' LIKE 'A' would be true.

label: code-design

71714. Find old rows

71715. Microsoft-style identifiers in [...]

71716. Current read offset

71717. ** This version of the memory allocation is for use by the application. ** First make sure the memory subsystem is initialized, then do the ** allocation.

71718. ** This object becomes the sqlite3_user_data() for the SQL functions ** that are created by sqlite3_rtree_geometry_callback() and ** sqlite3_rtree_query_callback() and which appear on the right of MATCH ** operators in order to constrain a search. *** ** xGeom and xQueryFunc are the callback functions. Exactly one of ** xGeom and xQueryFunc fields is non-NULL, depending on whether the ** SQL function was created using sqlite3_rtree_geometry_callback() or ** sqlite3_rtree_query_callback(). *** ** This object is deleted automatically by the destructor mechanism in ** sqlite3_create_function_v2().

71719. ** This callback is invoked once for each index when reading the ** sqlite_stat1 table. *** ** argv[0] = name of the table ** argv[1] = name of the index (might be NULL) ** argv[2] = results of analysis - on integer for each column ** ** Entries for which argv[1]==NULL simply record the number of rows in ** the table.

71720. ** Generate code that will cause the most recent index analysis to ** be loaded into internal hash tables where is can be used.

71721. True for UPDATE, False for INSERT

71722. Statement handle pointing at new row

71723. **comment:** Number of COLNAME_xxx symbols

label: code-design

71724. single- and double-quoted strings

71725. Exact encoding match

71726. If the call to xBestIndex() with all terms enabled produced a plan ** that does not require any source tables (IOW: a plan with mBest==0), ** then there is no point in making any further calls to xBestIndex() ** since they will all return the same result (if the xBestIndex() ** implementation is sane).

71727. New row data is stored here

71728. ** Check to see if the frame with header in aFrame[] and content ** in aData[] is valid. If it is a valid frame, fill *piPage and ** *pnTruncate and return true. Return if the frame is not valid.

71729. Attempt to compute the RHS. After this step, if anything other than ** IN_INDEX_NOOP is returned, the table opened ith cursor pExpr->iTable ** contains the values that make up the RHS. If IN_INDEX_NOOP is returned, ** the RHS has not yet been coded.

71730. Sorting index number. 0==None

71731. Resize an allocation

71732. Allocate space to hold the change in document sizes

71733. Currently always set to 2

71734. Opcode: SoftNull P1 * * * *** Synopsis: r[P1]=NULL *** Set register P1 to have the value NULL as seen by the OP_MakeRecord ** instruction, but do not free any string or blob memory associated with ** the register, so that if the value was a string or blob that was ** previously copied using OP_SCopy, the copies will continue to be valid.

71735. ** Generate the text of a WHERE expression which can be used to select all ** tables that have foreign key constraints that refer to table pTab (i.e. ** constraints for which pTab is the parent table) from the sqlite_master ** table.

71736. ** Allowed values for the DB.pSchema->flags field. *** ** The DB_SchemaLoaded flag is set after the database schema has been ** read into internal hash tables. *** ** DB_UnresetViews means that one or more views have column names that ** have been filled out. If the schema changes, these column names might ** changes and so the view will need to be reset.

71737. Used size of aTree/aReadr (power of 2)

71738. Language being queried for

71739. Only the rowid is required for a table btree

71740. Bytes of memory required to allocate the new cache

71741. **comment:** *** CAPI3REF: Prepared Statement Status ** METHOD: sqlite3_stmt *** ** ^/(Each prepared statement maintains various ** [SQLITE_STMTSTATUS counters] that measure the number ** of times it has performed specific operations.)^ These counters can ** be used to monitor the performance characteristics of the prepared ** statements. For example, if the number of table steps greatly exceeds ** the number of table searches or result rows, that would tend to indicate ** that the prepared statement is using a full table scan rather than ** an index. *** ** ^/(This interface is used to retrieve and reset counter values from ** a [prepared

statement]. The first argument is the prepared statement ** object to be interrogated. The second argument ** is an integer code for a specific [SQLITE_STMTSTATUS counter] ** to be interrogated.)^ ** ^The current value of the requested counter is returned. ** ^If the resetFlg is true, then the counter is reset to zero after this ** interface call returns. ** ** See also: [sqlite3_status()] and [sqlite3_db_status()].

label: code-design

71742. Figure out whether or not the current contents of memory should be ** flushed to a PMA before continuing. If so, do so. ** ** If using the single large allocation mode (pSorter->aMemory!=0), then ** flush the contents of memory to a new PMA if (a) at least one value is ** already in memory and (b) the new value will not fit in memory. ** ** Or, if using separate allocations for each record, flush the contents ** of memory to a PMA if either of the following are true: ** ** * The total memory allocated for the in-memory list is greater ** than (page-size * cache-size), or ** ** * The total memory allocated for the in-memory list is greater ** than (page-size * 10) and sqlite3HeapNearlyFull() returns true.

71743. from

71744. OUT: Tokenization cursor

71745. Size of zName in bytes, including \0

71746. Buffer to write to

71747. True if docids are descending

71748. ** Initialize the iterator object pIter to iterate through the entries in ** segment pSeg. The iterator is left pointing to the first entry when ** this function returns. ** ** If an error occurs, Fts5Index.rc is set to an appropriate error code. If ** an error has already occurred when this function is called, it is a no-op.

71749. A range of registers to hold previous output

71750. Number of backup operations reading this btree

71751. Write the old.* vector first.

71752. **comment:** ** An instance of this object is used for writing a PMA. ** ** The PMA is written one record at a time. Each record is of an arbitrary ** size. But I/O is more efficient if it occurs in page-sized blocks where ** each block is aligned on a page boundary. This object caches writes to ** the PMA so that aligned, page-size blocks are written.

label: code-design

71753. ** If the Index.aSample variable is not NULL, delete the aSample[] array ** and its contents.

71754. Jumps here in order to return true.

71755. Mask of SQLITE_INDEX_SCAN_* flags

71756. Cursor for the first index

71757. This block advances the phrase iterators until they point to a set of ** entries that together comprise a match.

71758. Verify that the every entry in the mapping region is still reachable ** via the hash table even after the cleanup.

71759. ** Generate code that will erase and refill index *pIdx. This is ** used to initialize a newly created index or to recompute the ** content of an index in response to a REINDEX command. ** ** if memRootPage is not negative, it means that the index is newly ** created. The register specified by memRootPage contains the ** root page number of the index. If memRootPage is negative, then ** the index already exists and must be cleared before being refilled and ** the root page number of the index is taken from pIndex->tNum.

71760. Fts3 table handle

71761. As configured by CursorSetHints()

71762. Top of insert loop. Label "C" in templates 3 and 4

71763. Arguments to currently executing user function

71764. Exclude terms using these operators

71765. Opcode: VDestroy P1 * * P4 * * * * P4 is the name of a virtual table in database P1. Call the xDestroy method ** of that table.

71766. ** The maximum value of a ?nnn wildcard that the parser will accept.

71767. ***** Interface to code in fts5_buffer.c.

71768. Underlying file is readonly

71769. The end of a position list is marked by a zero encoded as an FTS3 ** varint. A single POS_END (0) byte. Except, if the 0 byte is preceded by ** a byte with the 0x80 bit set, then it is not a varint 0, but the tail ** of some other, multi-byte, value. ** ** The following while-loop moves pEnd to point to the first byte that is not ** immediately preceded by a byte with the 0x80 bit set. Then increments ** pEnd once more so that it points to the byte immediately following the ** last byte in the position-list.

71770. 900

71771. aSeg[] index of firsttest iterator

71772. ** An instance of the WhereScan object is used as an iterator for locating ** terms in the WHERE clause that are useful to the query planner.

71773. ** Return true if there have been no changes to monitored tables recorded ** by the session object passed as the only argument.

71774. Query flattening

71775. Preferred text encoding

71776. xFileSize

71777. index internal

71778. The very beginning of the WHERE loop

71779. Pointer to corresponding expr token

71780. Snippet start text - ""

71781. ** Compare the OBJECT label at pNode against zKey,nKey. Return true on ** a match.

71782. Starting offset to score

71783. 2x number of WHEN terms

71784. Query term

71785. tab1 must not have triggers

71786. Hint level

71787. OUT: Copy of zThis

71788. ** Callback used by fts5Bm25GetData() to count the number of rows in the ** table matched by each individual phrase within the query.

71789. If the new Btree uses a sharable pBtShared, then link the new ** Btree into the list of all sharable Btrees for the same connection. ** The list is kept in ascending order by pBt address.

71790. Root page of PK index

71791. Space to write cursor structure

71792. Copy of SubProgram.token

71793. ** Maximum number of prefix indexes on single FTS5 table. This must be ** less than 32. If it is set to anything large than that, an #error ** directive in fts5_index.c will cause the build to fail.

71794. Close all cursors opened via this handle.

71795. ***** Begin %syntax_error code *****

71796. write it out to the temporary break file

71797. Destructor for pUser

71798. Pointer to save as auxdata

71799. Release memory back to the heap

71800. ** Error message for when two or more terms of a compound select have different ** size result sets.

71801. Registered using sqlite3_prepupdate_hook()

71802. ** Return TRUE if the word ends with three letters which ** are consonant-vowel-consonant and where the final consonant ** is not 'w', 'x', or 'y'. ** ** The word is reversed here. So we are really checking the ** first three letters and the first one cannot be in [wxy].

71803. If the function returned an error, throw an exception

71804. Here code is inserted which will be executed whenever the ** parser fails

71805. Implement the main merge loop

71806. First argument for xOutput

71807. %w -> Strings with '\"' doubled

71808. Source line number where error occurred

71809. Number of pages in pPage[]

71810. **comment:** ** Move data out of a btree key or data field and into a Mem structure. ** The data is payload from the entry that pCur is currently pointing ** to. offset and amt determine what portion of the data or key to retrieve. ** The result is written into the pMem element. ** ** The pMem object must have been initialized. This routine will use ** pMem->zMalloc to hold the content from the btree, if possible. New ** pMem->zMalloc space will be allocated if necessary. The calling routine ** is responsible for making sure that the pMem object is eventually ** destroyed. ** ** If this routine fails for any reason (malloc returns NULL or unable ** to read from the disk) then the pMem is left in an inconsistent state.

label: code-design

71811. **comment:** The NEVER() on the second term is because sqlite3FunctionUsesThisSrc() ** is always called before sqlite3ExprAnalyzeAggregates() and so the ** TK_COLUMNS have not yet been converted into TK_AGG_COLUMN. If ** sqlite3FunctionUsesThisSrc() is used differently in the future, the ** NEVER() will need to be removed.

label: code-design

71812. idlist ::= idlist COMMA nm

71813. The complete IN operator

71814. ** An object of the following type is used to read data from a single ** FTS segment node. See the following functions: ** ** nodeReaderInit() ** nodeReaderNext() ** nodeReaderRelease()

71815. Do not use a rollback journal

71816. ID => nothing

71817. Must be element [1]

71818. Remove the page from the files free-list. This is not required ** if bCommit is non-zero. In that case, the free-list will be ** truncated to zero after this function returns, so it doesn't ** matter if it still contains some garbage entries.

71819. ** CAPI3REF: Index Of A Parameter With A Given Name ** METHOD: sqlite3_stmt ** ** ^Return the index of an SQL parameter given its name. ^The ** index value returned is suitable for use as the second ** parameter to [sqlite3_bind_blob|sqlite3_bind()]. ^A zero ** is returned if no matching parameter is found. ^The parameter ** name must be given in UTF-8 even if the original statement ** was prepared from UTF-16 text using [sqlite3_prepare16_v2()] or ** [sqlite3_prepare16_v3()]. ** ** See also: [sqlite3_bind_blob|sqlite3_bind()], ** [sqlite3_bind_parameter_count()], and ** [sqlite3_bind_parameter_name()].

71820. Size of allocation containing aBuf

71821. cmd ::= createkw trigger_decl BEGIN trigger_cmd_list END

71822. The ')' before options in the CREATE TABLE

71823. ROLLBACK

71824. if the conch has data compare the contents

71825. Operations on page references.

71826. If zName is specified, operate on only the one system call ** specified.

71827. Ensure the database schema has been loaded

71828. Limits

71829. Next and previous entry on the min-heap

71830. Types of VDBE cursors

71831. **comment:** mem1.u.i = 0; // not needed, here to silence compiler warning

label: code-design

71832. ** Open a temporary file. ** ** Write the file descriptor into *pFile. Return SQLITE_OK on success ** or some other error code if we fail. The OS will automatically ** delete the temporary file when it is closed. ** ** The flags passed to the VFS layer xOpen() call are those specified ** by parameter vfsFlags ORed with the following: ** ** SQLITE_OPEN_READWRITE ** SQLITE_OPEN_CREATE ** SQLITE_OPEN_EXCLUSIVE ** SQLITE_OPEN_DELETEONCLOSE

71833. **comment:** ** Return the opcode for a given address. If the address is -1, then ** return the most recently inserted opcode. ** ** If a memory allocation error has occurred prior to the calling of this ** routine, then a pointer to a dummy VdbeOp will be returned. That opcode ** is readable but not writable, though it is cast to a writable value. ** The return of a dummy opcode allows the call to continue functioning ** after an OOM fault without having to check to see if the return from ** this routine is a valid pointer. But because the dummy.opcode is 0, ** dummy will never be written to. This is verified by code inspection and ** by running with Valgrind.

label: code-design

71834. Write transactions are not possible on a read-only database

71835. ** Invoke the OP_AggFinalize opcode for every aggregate function ** in the AggInfo structure.

71836. Single result column

71837. EVIDENCE-OF: R-02748-19096 This option sets the threading mode to ** Single-thread.

71838. Create/Connect to the underlying relational database schema. If ** that is successful, call sqlite3_declare_vtab() to configure ** the r-tree table schema.

71839. Size of aRoot[] in bytes

71840. ** 2003 April 6 **** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file contains code used to implement the PRAGMA command.

71841. ** Append text to the HighlightContext output string - p->zOut. Argument ** z points to a buffer containing n bytes of text to append. If n is ** negative, everything up until the first '\0' is appended to the output. ** ** If *pRc is set to any value other than SQLITE_OK when this function is ** called, it is a no-op. If an error (i.e. an OOM condition) is encountered, ** *pRc is set to an error code before returning.

71842. Populate the azTblCol[] and nTblCol variables based on the columns ** of the input table. Ignore any input table columns that begin with ** "rbu_".

71843. ** This is the callback routine for the code that initializes the ** database. See sqlite3Init() below for additional information. ** This routine is also called from the OP_ParseSchema opcode of the VDBE. ** ** Each callback contains the following information: ** ** argv[0] = name of thing being created ** argv[1] = root page number for table or index. 0 for trigger or view. ** argv[2] = SQL text for the CREATE statement. **

71844. 5x

71845. Read the specified cookie value

71846. Create the SELECT statement to read keys in sorted order

71847. #include <stdlib.h>

71848. True if all terms are at the same rowid

71849. The prepared statement being evaluated

71850. All offsets for this column have been gathered.

71851. Add the first rowid field to the hash-entry

71852. If there is already a lock of this type or more restrictive on the ** unixFile, do nothing. Don't use the end_lock: exit path, as ** unixEnterMutex() hasn't been called yet.

71853. Name to be added

71854. ** Generate code to move content from registers iFrom...iFrom+nReg-1 ** over to iTo..iTo+nReg-1. Keep the column cache up-to-date.

71855. Matching column in the left table

71856. Used to iterate through old.* values

71857. Number of nested loop

71858. PRAGMA checkpoint_fullfsync=ON

71859. Added by 3.5.0

71860. **comment:** One of the INCRINIT_XXX constants

label: code-design

71861. The original page number

71862. ** Invoke the xSync method of all virtual tables in the sqlite3.aVTrans ** array. Return the error code for the first error that occurs, or ** SQLITE_OK if all xSync operations are successful. ** ** If an error message is available, leave it in p->zErrMsg.

71863. ** Implementation of the R*-tree variant of SplitNode from Beckman[1990].

71864. Opcode: OpenPseudo P1 P2 P3 *** Synopsis: P3 columns in r[P2] *** Open a new cursor that points to a fake table that contains a single ** row of data. The content of that one row is the content of memory ** register P2. In other words, cursor P1 becomes an alias for the ** MEM_Blob content contained in register P2.

*** A pseudo-table created by this opcode is used to hold a single ** row output from the sorter so that the row can be decomposed into ** individual columns using the OP_Column opcode. The OP_Column opcode ** is the only cursor opcode that works with a pseudo-table. *** P3 is the number of fields in the records that will be stored by ** the pseudo-table.

71865. Pointer map 'type' entry for pDbPage

71866. jump, in3

71867. Buffer containing poslist

71868. sqlite3GetVdbe() has always been previously called

71869. Delete all attached table objects. And the contents of their ** associated hash-tables.

71870. Restrictions on range of iteration

71871. ** This function ensures that the caller has obtained an exclusive ** shared-cache table-lock on the %_segdir table. This is required before ** writing data to the fts3 table. If this lock is not acquired first, then ** the caller may end up attempting to take this lock as part of committing ** a transaction, causing SQLite to return SQLITE_LOCKED or ** LOCKED_SHAREDCACHE to a COMMIT command. *** It is best to avoid this because if FTS3 returns any error when ** committing a transaction, the whole transaction will be rolled back. ** And this is not what users expect when they get SQLITE_LOCKED_SHAREDCACHE. ** It can still happen if the user locks the underlying tables directly ** instead of accessing them via FTS.

71872. Will become the return value of this function

71873. **comment:** The skip-scan logic inside the call to codeAllEqualityConstraints() ** above has already left the cursor sitting on the correct row, ** so no further seeking is needed
label: code-design

71874. ** Return a list of comma separated SQL expressions and a FROM clause that ** could be used in a SELECT statement such as the following: *** ** SELECT <list of expressions> FROM %_content AS x ... *** ** to return the docid, followed by each column of text data in order ** from left to write. If parameter zFunc is not NULL, then instead of ** being returned directly each column of text data is passed to an SQL ** function named zFunc first. For example, if zFunc is "unzip" and the ** table has the three user-defined columns "a", "b", and "c", the following ** string is returned: *** ** "docid, unzip(x.'a'), unzip(x.'b'), unzip(x.'c') FROM %_content AS x" *** ** The pointer returned points to a buffer allocated by sqlite3_malloc(). It ** is the responsibility of the caller to eventually free it. ** ** If *pRc is not SQLITE_OK when this function is called, it is a no-op (and ** a NULL pointer is returned). Otherwise, if an OOM error is encountered ** by this function, NULL is returned and *pRc is set to SQLITE_NOMEM. If ** no error occurs, *pRc is left unmodified.

71875. Result code to return

71876. Newly allocated winShmNode

71877. Integer value

71878. Size of sibling on the left

71879. ***** End of msvc.h *****

71880. The WhereOrSet object holds a set of possible WhereOrCosts that ** correspond to the subquery(s) of OR-clause processing. Only the ** best N_OR_COST elements are retained.

71881. If non-NULL, bulk memory to hold pList

71882. Use SorterOpen instead of OpenEphemeral

71883. Allocate the buffers used by this operation. The allocation is ** relinquished before this function returns.

71884. IMP: R-64479-57858

71885. DELETE statement

71886. The minimum count for any phrase so far.

71887. sqlite3VbeCursorRestore() can only fail if the record has been deleted ** out from under the cursor. That will never happens for an IdxRowid ** or Seek opcode

71888. Next available winShm.id value

71889. 1230

71890. Used when p4type is P4_SUBPROGRAM

71891. Connection is read only

71892. Linked list of records

71893. New segment within pStruct

71894. ** Return true if it OK to factor constant expressions into the initialization ** code. The argument is a Parse object for the code generator.

71895. SQLITE_AFF_TEXT

71896. ** Allocate a new Select structure and return a pointer to that ** structure.

71897. ** Token pTok has appeared in a MATCH expression where the NEAR operator ** is expected. If token pTok does not contain "NEAR", store an error ** in the pParse object.

71898. A parse of the cell we are pointing at

71899. Name of pragma

71900. ** Determine if the argument is a power of two

71901. True if the value to convert is signed

71902. ** Schema of the tokenizer table.

71903. ***** Begin file opcodes.c *****

71904. Value for "level" column of output

71905. Column for entry to add

71906. SQLITE_OK

71907. Virtual table handle

71908. ** Parse context.

71909. ** Return the total number of blocks this module has read from the %_data ** table since it was created.

71910. **comment:** EVIDENCE-OF: R-26747-61719 When the application provides any amount of ** scratch memory using SQLITE_CONFIG_SCRATCH, SQLite avoids unnecessary ** large heap allocations.
label: code-design

71911. Sync the db file

71912. **comment:** ** CAPI3REF: Low-Level Control Of Database Files ** METHOD: sqlite3 *** ** ^The [sqlite3_file_control()] interface makes a direct call to the ** xFileControl method for the [sqlite3_io_methods] object associated ** with a particular database identified by the second argument. ^The ** name of the database is "main" for the main database or "temp" for the ** TEMP database, or the name that appears after the AS keyword for ** databases that are added using the [ATTACH] SQL command. ** ^A NULL pointer can be used in place of "main" to refer to the ** main database file. ** ^The third and fourth parameters to this routine ** are passed directly through to the second and third parameters of ** the xFileControl method. ^The return value of the xFileControl ** method becomes the return value of this routine. ** ** ^The SQLITE_FCNTL_FILE_POINTER value for the op parameter causes ** a pointer to the underlying [sqlite3_file] object to be written into ** the space pointed to by the 4th parameter. ^The SQLITE_FCNTL_FILE_POINTER ** case is a short-circuit path which does not actually invoke the ** underlying sqlite3_io_methods.xFileControl method. ** ** ^If the second parameter (zDbName) does not match the name of any ** open database file, then SQLITE_ERROR is returned. ^This error ** code is not remembered and will not be recalled by [sqlite3_errcode()] ** or [sqlite3_errmsg()]. The underlying xFileControl method might ** also return SQLITE_ERROR. There is no way to distinguish between ** an incorrect zDbName and an SQLITE_ERROR return from the underlying ** xFileControl method. ** ** See also: [SQLITE_FCNTL_LOCKSTATE]
label: code-design

71913. int nByte

71914. ** for(rc = sqlite3Fts5ExprFirst(pExpr, pIdx, bDesc); ** rc==SQLITE_OK && 0==sqlite3Fts5ExprEof(pExpr); ** rc = sqlite3Fts5ExprNext(pExpr) **){ ** // The document with rowid iRowid matches the expression! ** i64 iRowid = sqlite3Fts5ExprRowid(pExpr); ** }

71915. **comment:** ** Change the way data is synced to disk in order to increase or decrease ** how well the database resists damage due to OS crashes and power ** failures. Level 1 is the same as asynchronous (no syncs() occur and ** there is a high probability of damage) Level 2 is the default. There ** is a very low but non-zero probability of damage. Level 3 reduces the ** probability of damage to near zero but with a write performance reduction.
label: code-design

71916. If the pager is in the ERROR state and the call to unlock the database ** file fails, set the current lock to UNKNOWN_LOCK. See the comment ** above the #define for UNKNOWN_LOCK for an explanation of why this ** is necessary.

71917. For ORDER BY, column number in result set

71918. Input JSON
71919. ***** Begin file btmutex.c *****
71920. **comment:** The row-trigger may have deleted the row being updated. In this ** case, jump to the next row. No updates or AFTER triggers are ** required. This behavior - what happens when the row being updated ** is deleted or renamed by a BEFORE trigger - is left undefined in the ** documentation.
label: documentation
71921. Index of the database that is being written
71922. Name of the operator
71923. After reading the first page of the database assuming a page size ** of BtShared.pageSize, we have discovered that the page-size is ** actually pageSize. Unlock the database, leave pBt->pPage1 at ** zero and return SQLITE_OK. The caller will call this function ** again with the correct page-size.
71924. 79
71925. ** Recover as many snapshots as possible from the wal file associated with ** schema zDb of database db.
71926. Advance each of the segments to point to the first docid.
71927. The pointer is always the first 4 bytes of the page in this case.
71928. Mapped memory region
71929. number of free pages
71930. IN/OUT: PendingList structure
71931. If there is an ORDER BY clause and it is not being ignored, set up ** space for the aSortCost[] array. Each element of the aSortCost array ** is either zero - meaning it has not yet been initialized - or the ** cost of sorting nRowEst rows of data where the first X terms of ** the ORDER BY clause are already in order, where X is the array ** index.
71932. Create labels for the "break" and "continue" instructions ** for the current loop. Jump to addrBrk to break out of a loop. ** Jump to cont to go immediately to the next iteration of the ** loop. *** When there is an IN operator, we also have a "addrNxt" label that ** means to continue with the next IN value combination. When ** there are no IN operators in the constraints, the "addrNxt" label ** is the same as "addrBrk".
71933. Number of neighbors on either side of pPage
71934. ** The unixFile structure is subclass of sqlite3_file specific to the unix ** VFS implementations.
71935. the open file object
71936. ** Make fts5yyt testcase() the same as testcase()
71937. old.* database record
71938. **comment:** ** This function must be called before exiting any API function (i.e. ** returning control to the user) that has called sqlite3_malloc or ** sqlite3_realloc. *** The returned value is normally a copy of the second argument to this ** function. However, if a malloc() failure has occurred since the previous ** invocation SQLITE_NOMEM is returned instead. *** If an OOM as occurred, then the connection error-code (the value ** returned by sqlite3_errcode()) is set to SQLITE_NOMEM.
label: code-design
71939. Copy of update mask used with pUpdate
71940. ** CAPI3REF: Number Of SQL Parameters ** METHOD: sqlite3_stmt ** ** ^This routine can be used to find the number of [SQL parameters] ** in a [prepared statement]. SQL parameters are tokens of the ** form "?", "?NNN", ":AAA", "\$AAA", or "@AAA" that serve as ** placeholders for values that are [sqlite3_bind_blob | bound] ** to the parameters at a later time. ** ** ^This routine actually returns the index of the largest (rightmost) ** parameter. For all forms except ?NNN, this will correspond to the ** number of unique parameters. If parameters of the ?NNN form are used, ** there may be gaps in the list.)^ ** See also: [sqlite3_bind_blob|sqlite3_bind()], ** [sqlite3_bind_parameter_name()], and ** [sqlite3_bind_parameter_index()].
71941. Forward references
71942. ERROR: generated size does not match predicted size
71943. Segment term filter configuration
71944. Write the nRec value into the journal file header. If in ** full-synchronous mode, sync the journal first. This ensures that ** all data has really hit the disk before nRec is updated to mark ** it as a candidate for rollback. *** This is not required if the persistent media supports the ** SAFE_APPEND property. Because in this case it is not possible ** for garbage data to be appended to the file, the nRec field ** is populated with 0xFFFFFFFF when the journal header is written ** and never needs to be updated.
71945. ** Forward declarations of structure
71946. Number of matching column names
71947. Check that no sample contains an anEq[] entry with an index of ** p->nMaxEqZero or greater set to zero.
71948. ** Generate an ORDER BY or GROUP BY term out-of-range error.
71949. ** Change the size of an existing memory allocation
71950. Silently convert bound parameters that appear inside of CREATE ** statements into a NULL when parsing the CREATE statement text out ** of the sqlite_master table
71951. ** This function does an "OR" merge of two doclists (output contains all ** positions contained in either argument doclist). If the docids in the ** input doclists are sorted in ascending order, parameter bDescDoclist ** should be false. If they are sorted in ascending order, it should be ** passed a non-zero value. *** If no error occurs, *paOut is set to point at an sqlite3_malloc'd buffer ** containing the output doclist and SQLITE_OK is returned. In this case ** *pnOut is set to the number of bytes in the output doclist. *** If an error occurs, an SQLite error code is returned. The output values ** are undefined in this case.
71952. Iterator used for several purposes
71953. ** Get the suggested cache-size value.
71954. ** The xRollbackTo() method. *** Discard the contents of the pending terms table.
71955. Hash value
71956. 64-bit offset
71957. Move the write position of the WAL back to iFrame. Called in ** response to a ROLLBACK TO command.
71958. Copy of third arg to _filter_table()
71959. IN operator used as a loop
71960. for each column: True==DESC, False==ASC
71961. %r -> 1st, 2nd, 3rd, 4th, etc. English only
71962. getenv("TEMP")
71963. 1 when initialized
71964. Advance pointer p until it points to pEnd or an 0x01 byte that is ** not part of a varint
71965. The database disk image is malformed
71966. ** Implementation of the round() function
71967. This branch is taken when the xShmMap() method returns SQLITE_BUSY. ** We assume this is a transient condition, so return WAL_RETRY. The ** xShmMap() implementation used by the default unix and win32 VFS ** modules may return SQLITE_BUSY due to a race condition in the ** code that determines whether or not the shared-memory region ** must be zeroed before the requested page is returned.
71968. Do we own the heap (i.e. destroy it on shutdown)?
71969. Store allocated handle here
71970. ** This routine generates code that will initialize all of the ** register used by the autoincrement tracker.
71971. The request could not be fulfilled using a freelist slot. Check ** to see if defragmentation is necessary.
71972. Forward references
71973. Forms a linked list of all cursors
71974. synopsis: if P1.nullRow then r[P3]=NULL, goto P2
71975. To many cells for a single page. The page must be corrupt
71976. ** Allowed values for the unixFile.ctrlFlags bitmask:
71977. All terms in pWLoop->aLTerm[] except pEndRange are used to initialize ** the cursor. These terms are not needed as hints for a pure range ** scan (that has no == terms) so omit them.
71978. Make sure there is enough space in p->azResult to hold everything ** we need to remember from this invocation of the callback.
71979. expr ::= expr likeop expr
71980. The associated database connection

71981. (302) selectnowith ::= oneselect (OPTIMIZED OUT)
71982. List of dirty pages to write
71983. ** If the cursor requires seeking (bSeekRequired flag is set), seek it. ** Return SQLITE_OK if no error occurs, or an SQLite error code otherwise. ** ** If argument bErrormsg is true and an error occurs, an error message may ** be left in sqlite3_vtab.zErrMsg.
71984. Parameter passed through to xFilter()
71985. Trigger under construct by a CREATE TRIGGER
71986. ** Cursor iCur is open on an intkey b-tree (a table). Register iRowid contains ** a rowid value just read from cursor iIdxCur, open on index pIdx. This ** function generates code to do a deferred seek of cursor iCur to the ** rowid stored in register iRowid. ** ** Normally, this is just: ** ** OP_DeferredSeek \$iCur \$iRowid
** ** However, if the scan currently being coded is a branch of an OR-loop and ** the statement currently being coded is a SELECT, then P3 of OP_DeferredSeek
** is set to iIdxCur and P4 is set to point to an array of integers ** containing one entry for each column of the table cursor iCur is open ** on. For each table
column, if the column is the i'th column of the ** index, then the corresponding array entry is set to (i+1). If the column ** does not appear in the index at all, the
array entry is set to 0.
71987. ** Define the required Windows SDK version constants if they are not ** already available.
71988. Mapping from columns of pTab to entries in pChanges
71989. We have a match. Do not delete the master journal file.
71990. New level-0 PMA merger
71991. Number of segments not deleted
71992. **comment:** ** CAPI3REF: Obtain old.* Values From A Changeset Iterator *** ** The pIter argument passed to this function may either be an iterator ** passed to a
conflict-handler by [sqlite3changeset_apply()], or an iterator ** created by [sqlite3changeset_start()]. In the latter case, the most recent ** call to
[sqlite3changeset_next()] must have returned SQLITE_ROW. ** Furthermore, it may only be called if the type of change that the iterator ** currently points to is
either [SQLITE_DELETE] or [SQLITE_UPDATE]. Otherwise, ** this function returns [SQLITE_MISUSE] and sets *ppValue to NULL. ** ** Argument iVal
must be greater than or equal to 0, and less than the number ** of columns in the table affected by the current change. Otherwise, ** [SQLITE_RANGE] is
returned and *ppValue is set to NULL. ** ** If successful, this function sets *ppValue to point to a protected ** sqlite3_value object containing the iVal'th value
from the vector of ** original row values stored as part of the UPDATE or DELETE change and ** returns SQLITE_OK. The name of the function comes from
the fact that this ** is similar to the "old.*" columns available to update or delete triggers. ** ** If some other error occurs (e.g. an OOM condition), an SQLite
error code ** is returned and *ppValue is set to NULL.
label: code-design
71993. **comment:** If this is not a threadsafe build (SQLITE_THREADSAFE==0), then use ** the strerror() function to obtain the human-readable error message **
equivalent to errno. Otherwise, use strerror_r().
label: code-design
71994. Page in use, not on the LRU list
71995. sqlite_stat4.nDLt
71996. ** Store an incr-merge hint in the database.
71997. If BTREE_SAVEPOSITION is set, the cursor must already be pointing ** to a row with the same key as the new entry being inserted.
71998. ** Head of a linked list of all sqlite3 objects created by this process ** for which either sqlite3.pBlockingConnection or sqlite3.pUnlockConnection ** is not
NULL. This variable may only accessed while the STATIC_MASTER ** mutex is held.
71999. abNotindexed
72000. ***** End of hash.c *****
72001. Incr-merge parameter A
72002. 96
72003. Name of the system call
72004. ** This is a helper routine for sqlite3PcacheFetchFinish() *** ** In the uncommon case where the page being fetched has not been ** initialized, this routine is
invoked to do the initialization. ** This routine is broken out into a separate function since it ** requires extra stack manipulation that can be avoided in the
common ** case.
72005. Find out which shared locks are already held by sibling connections. ** If any sibling already holds an exclusive lock, go ahead and return ** SQLITE_BUSY.
72006. ** This function populates the pRtree->nRowEst variable with an estimate ** of the number of rows in the virtual table. If possible, this is based ** on sqlite_stat1
data. Otherwise, use RTREE_DEFAULT_ROWEST.
72007. Module destructor function
72008. Link the new savepoint into the database handle's list.
72009. Expression used to recompute the rowid
72010. Pointer to buffer containing doclist
72011. **comment:** ** The interface to the virtual-table mechanism is currently considered ** to be experimental. The interface might change in incompatible ways. ** If
this is a problem for you, do not use the interface at this time. ** ** When the virtual-table mechanism stabilizes, we will declare the ** interface fixed, support it
indefinitely, and remove this comment.
label: code-design
72012. Copy the triggers, views, and virtual tables from the main database ** over to the temporary database. None of these objects has any ** associated storage, so all
we have to do is copy their entries ** from the SQLITE_MASTER table.
72013. Opcode: Gosub P1 P2 * * * * * Write the current address onto register P1 ** and then jump to address P2.
72014. **comment:** ** This function may only be called with an iterator passed to an ** SQLITE_CHANGESET_FOREIGN_KEY conflict handler callback. In this case
** it sets the output variable to the total number of known foreign key ** violations in the destination database and returns SQLITE_OK. ** ** In all other cases
this function returns SQLITE_MISUSE.
label: code-design
72015. Database connection to read from
72016. EVIDENCE-OF: R-37923-12173 The sqlite3_prepare_v2() interface works ** exactly the same as sqlite3_prepare_v3() with a zero prepFlags ** parameter. *** **
Proof in that the 5th parameter to sqlite3LockAndPrepare is 0
72017. IN/OUT: Error code
72018. Column that token must appear in (or -1)
72019. Buffer containing prefix to match
72020. ** This function is a no-op if *pRc is set to other than SQLITE_OK when it ** is called. Otherwise, append a serialized table header (part of the binary **
changeset format) to buffer *pBuf. If an error occurs, set *pRc to an ** SQLite error code before returning.
72021. Change structure
72022. cmd ::= REINDEX nm dbnm
72023. fts3_write.c
72024. ***** Begin file os_win.h *****
72025. File opened that is not a database file
72026. Size of buffer aData[] in bytes
72027. IMMEDIATE => ID
72028. Used when p4type is P4_INT64
72029. This only happens when coding check constraints
72030. Memory space from which to build Expr object
72031. ** Free the configuration object passed as the only argument.
72032. * The value used with sqlite3_win32_set_directory() to specify that * the temporary directory should be changed.
72033. Number of vector elements in comparison
72034. ** The next global variable is incremented each time the OP_Found opcode ** is executed. This is used to test whether or not the foreign key ** operation
implemented using OP_FkIsZero is working. This variable ** has no function other than to help verify the correct operation of the ** library.
72035. This expression is a "*" or a "TABLE.**" and needs to be ** expanded.
72036. If rc is not SQLITE_OK at this point, then either the malloc ** subsystem could not be initialized or the system failed to allocate ** the pInitMutex mutex. Return
an error in either case.

72037. bFullMutex
72038. Query prefix, if any
72039. Number of columns in this key
72040. If the sibling pages are not leaves, ensure that the right-child pointer ** of the right-most new sibling page is set to the value that was ** originally in the same field of the right-most old sibling page.
72041. 1410
72042. Code the trigger program into the sub-vdbe.
72043. ** A minimum allocation is an instance of the following structure. ** Larger allocations are an array of these structures where the ** size of the array is a power of 2. ** ** The size of this object must be a power of two. That fact is ** verified in memsys5Init().
72044. ** Called when a transaction completes (either by COMMIT or ROLLBACK). ** The sqlite3_blob object should be released at this point.
72045. Parsing context for error messages
72046. *****Include os_win.h in the middle of mutex_w32.c *****
72047. ** CAPI3REF: Total Number Of Rows Modified ** METHOD: sqlite3 *** ^This function returns the total number of rows inserted, modified or ** deleted by all [INSERT], [UPDATE] or [DELETE] statements completed ** since the database connection was opened, including those executed as ** part of trigger programs. ^Executing any other type of SQL statement ** does not affect the value returned by sqlite3_total_changes(). ** ** ^Changes made as part of [foreign key actions] are included in the ** count, but those made as part of REPLACE constraint resolution are ** not. ^Changes to a view that are intercepted by INSTEAD OF triggers ** are not counted. ** ** See also the [sqlite3_changes()] interface, the ** [count_changes pragma], and the [total_changes() SQL function]. ** ** If a separate thread makes changes on the same database connection ** while [sqlite3_total_changes()] is running then the value ** returned is unpredictable and not meaningful.
72048. ** Read or write a four-byte big-endian integer value.
72049. ** Return pointers to the hash table and page number array stored on ** page iHash of the wal-index. The wal-index is broken into 32KB pages ** numbered starting from 0. ** ** Set output variable *paHash to point to the start of the hash table ** in the wal-index file. Set *piZero to one less than the frame ** number of the first frame indexed by this hash table. If a ** slot in the hash table is set to N, it refers to frame number ** (*piZero+N) in the log. ** ** Finally, set *paPgno so that *paPgno[1] is the page number of the ** first frame indexed by the hash table, frame (*piZero+1).
72050. Pointer map 'page-no' entry for pDbPage
72051. Pointer to module implementation
72052. Next output slot in pColset
72053. ** Return TRUE if the word ends in a double consonant. ** ** The text is reversed here. So we are really looking at ** the first two characters of z[].
72054. Number of Journal Records
72055. Check some assumptions: ** (a) the cursor is open for writing, ** (b) there is a read/write transaction open, ** (c) the connection holds a write-lock on the table (if required), ** (d) there are no conflicting read-locks, and ** (e) the cursor points at a valid row of an intKey table.
72056. The main btree structure
72057. Now, if the table is not stored in the temp database, reload any temp ** triggers. Don't use IN(...) in case SQLITE_OMIT_SUBQUERY is defined.
72058. The database name
72059. If this is a new rowid, append the 4-byte size field for the previous ** entry, and the new rowid for this entry.
72060. The function calling context
72061. Maximum page size is 64KiB
72062. ***** The next group of routines implement the I/O methods specified ** by the sqlite3_io_methods object. *****
72063. ** Add an entry to one of the pending-terms hash tables.
72064. Return number of characters processed so far. %n
72065. Add the ON clause to the end of the WHERE clause, connected by ** an AND operator.
72066. ** 2008 August 05 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file implements that page cache.
72067. Existing index DESCENDING
72068. sqlite3_test_control(SQLITE_TESTCTRL_OPTIMIZATIONS, sqlite3 *db, int N) ** ** Enable or disable various optimizations for testing purposes. The ** argument N is a bitmask of optimizations to be disabled. For normal ** operation N should be 0. The idea is that a test program (like the ** SQL Logic Test or SLT test module) can run the same SQL multiple times ** with various optimizations disabled to verify that the same answer ** is obtained in every case.
72069. Malloc'd child-journal file descriptor
72070. Used to iterate through hash buckets
72071. Read the schema information out of the schema tables
72072. ** This routine identifies subexpressions in the WHERE clause where ** each subexpression is separated by the AND operator or some other ** operator specified in the op parameter. The WhereClause structure ** is filled with pointers to subexpressions. For example: ** ** WHERE a=='hello' AND coalesce(b,11)<10 AND (c+12!=d OR c==22) ** \ / \ / ** slot[0] slot[1] slot[2] ** ** The original WHERE clause in pExpr is unaltered. All this routine ** does is make slot[] entries point to substructure within pExpr. ** ** In the previous sentence and in the diagram, "slot[]" refers to ** the WhereClause.a[] array. The slot[] array grows as needed to contain ** all terms of the WHERE clause.
72073. Used to be pPager->nOvfl
72074. If it is open, sync the journal file before calling UnlockAndRollback. ** If this is not done, then an unsynced portion of the open journal ** file may be played back into the database. If a power failure occurs ** while this is happening, the database could become corrupt. ** ** If an error occurs while trying to sync the journal, shift the pager ** into the ERROR state. This causes UnlockAndRollback to unlock the ** database and close the journal file without attempting to roll it ** back or finalize it. The next database user will have to do hot-journal ** rollback before accessing the database file.
72075. TRANSACTION => nothing
72076. ** xNext() method for a node of type FTS5_TERM.
72077. X is not a subset of Y
72078. ** Return true if this is an in-memory or temp-file backed pager.
72079. Pointer to current key
72080. RECURSIVE => ID
72081. ** Return the global mutex used by this PCACHE implementation. The ** sqlite3_status() routine needs access to this mutex.
72082. ** Log an error that is an API call on a connection pointer that should ** not have been used. The "type" of connection pointer is given as the ** argument. The zType is a word like "NULL" or "closed" or "invalid".
72083. Number of extra copies of last page
72084. True if statement may modify/insert multiple rows
72085. Copy of pContext passed to s_r_g_c()
72086. ** The following routines are implementations of the MemPage.xParseCell() ** method. ** ** Parse a cell content block and fill in the CellInfo structure. ** ** btreeParseCellPtr() => table btree leaf nodes ** btreeParseCellNoPayload() => table btree internal nodes ** btreeParseCellPtrIndex() => index btree nodes ** ** There is also a wrapper function btreeParseCell() that works for ** all MemPage types and that references the cell by index rather than ** by pointer.
72087. Special case: Add leading zeros if the flag .zeropad is ** set and we are not left justified
72088. Possibly overload the function if the first argument is ** a virtual table column. ** ** For infix functions (LIKE, GLOB, REGEXP, and MATCH) use the ** second argument, not the first, as the argument to test to ** see if it is a column in a virtual table. This is done because ** the left operand of infix functions (the operand we want to ** control overloading) ends up as the second argument to the ** function. The expression "A glob B" is equivalent to ** "glob(B,A)". We want to use the A in "A glob B" to test ** for function overloading. But we use the B term in "glob(B,A)".
72089. Initial checksum value input
72090. Offset to cell content of cell being deleted
72091. A count of rows changed
72092. If there is both an upper and lower bound specified, and the ** comparisons indicate that they are close together, use the fallback ** method (assume that the scan visits 1/64 of the rows) for estimating ** the number of rows visited. Otherwise, estimate the number of rows ** using the method described in the header comment for this function.

72093. ** Destroy a bitmap object. Reclaim all memory used.
72094. Write results into this register
72095. Array of jump instruction addresses
72096. Previous docid stored in doclist
72097. Return Code
72098. Source page object
72099. Phrase object to initialize
72100. Pointer to memory from sqliteMalloc()
72101. 229
72102. An i-node
72103. ** When full-text index nodes are loaded from disk, the buffer that they ** are loaded into has the following number of bytes of padding at the end ** of it. i.e. if a full-text index node is 900 bytes in size, then a buffer ** of 920 bytes is allocated for it. *** This means that if we have a pointer into a buffer containing node data, ** it is always safe to read up to two varints from it without risking an ** overread, even if the node data is corrupted.
72104. 12
72105. ** If this is a WAL database, call sqlite3WalSnapshotRecover(). If this ** is not a WAL database, return an error.
72106. ** A smaller version of VdbeOp used for the VdbeAddOpList() function because ** it takes up less space.
72107. If this is a main database file and the file was opened using a URI ** filename, check for the "modeof" parameter. If present, interpret ** its value as a filename and try to copy the mode, uid and gid from ** that file.
72108. Results
72109. get the host ID via gethostuuid(), pHostID must point to PROXY_HOSTIDLEN ** bytes of writable memory.
72110. ** Close an rbu file.
72111. ** Load an incr-merge hint from the database. The incr-merge hint, if one ** exists, is stored in the rowid==1 row of the %_stat table. *** If successful, populate blob *pHint with the value read from the %_stat ** table and return SQLITE_OK. Otherwise, if an error occurs, return an ** SQLite error code.
72112. Check to verify that it is possible to get a read lock on all ** database schemas. The inability to get a read lock indicates that ** some other database connection is holding a write-lock, which in ** turn means that the other connection has made uncommitted changes ** to the schema. *** Were we to proceed and prepare the statement against the uncommitted ** schema changes and if those schema changes are subsequently rolled ** back and different changes are made in their place, then when this ** prepared statement goes to run the schema cookie would fail to detect ** the schema change. Disaster would follow. *** This thread is currently holding mutexes on all Btrees (because ** of the sqlite3BtreeEnterAll() in sqlite3LockAndPrepare()) so it ** is not possible for another thread to start a new schema change ** while this routine is running. Hence, we do not need to hold ** locks on the schema, we just need to make sure nobody else is ** holding them. *** Note that setting READ_UNCOMMITTED overrides most lock detection, ** but it does *not* override schema lock detection, so this all still ** works even if READ_UNCOMMITTED is set.
72113. For an index scan, make sure referenced columns are actually in ** the index.
72114. The database connection holding this btree
72115. The "Select-ID" for this loop
72116. lp ::= LP
72117. Context object for fts5ParseTokenize
72118. Strings. %s
72119. A 'table' record consists of: *** * A constant 'T' character, *** * Number of columns in said table (a varint), *** * An array of nCol bytes (sPK), *** * A null-terminated table name.
72120. If pShmNode->nRef has reached 0, then close the underlying ** shared-memory file, too
72121. ** Disable lookaside memory allocation for objects that might be ** shared across database connections.
72122. 2-byte (or larger) signed integer
72123. This can occur if there exists an index on a TEMP table which ** has the same name as another index on a permanent index. Since ** the permanent table is hidden by the TEMP table, we can also ** safely ignore the index on the permanent table.
72124. ** Argument aWalData must point to an array of WAL_SAVEPOINT_NDATA u32 ** values. This function populates the array with values required to ** "rollback" the write position of the WAL handle back to the current ** point in the event of a savepoint rollback (via WalSavepointUndo()).
72125. Each a[] describes a term of the WHERE clause
72126. EV: R-33326-45252
72127. Set the value of this object
72128. If we are unable to rollback, quit and return the error ** code. This will cause the pager to enter the error state ** so that no further harm will be done. Perhaps the next ** process to come along will be able to rollback the database.
72129. ** Convert the pStep->zTarget string into a SrcList and return a pointer ** to that SrcList. *** This routine adds a specific database name, if needed, to the target when ** forming the SrcList. This prevents a trigger in one database from ** referring to a target in another database. An exception is when the ** trigger is in TEMP in which case it can refer to any other database it ** wants.
72130. old.* and new.* values
72131. Register holding maximum rowid
72132. Re-insert the contents of any underfull nodes removed from the tree.
72133. Acquire a RESERVED lock
72134. Array of WHEN terms
72135. Phrase to merge pList/nList into
72136. IN/OUT: Previous rowid written (if any)
72137. Mask for four values above
72138. ** Change the value of a Mem to be a string or a BLOB. *** The memory management strategy depends on the value of the xDel ** parameter. If the value passed is SQLITE_TRANSIENT, then the ** string is copied into a (possibly existing) buffer managed by the ** Mem structure. Otherwise, any existing buffer is freed and the ** pointer copied. *** If the string is too large (if it exceeds the SQLITE_LIMIT_LENGTH ** size limit) then no memory allocation occurs. If the string can be ** stored without allocating memory, then it is. If a memory allocation ** is required to store the string, then value of pMem is unchanged. In ** either case, SQLITE_TOOBIG is returned.
72139. **comment:** Now that it is known how many phrases there are, allocate and zero ** the required space using malloc().
label: code-design
72140. Desired text encoding
72141. ** Recover the wal-index by reading the write-ahead log file. *** This routine first tries to establish an exclusive lock on the ** wal-index to prevent other threads/processes from doing anything ** with the WAL or wal-index while recovery is running. The ** WAL_RECOVER_LOCK is also held so that other threads will know ** that this thread is running recovery. If unable to establish ** the necessary locks, this routine returns SQLITE_BUSY.
72142. ** Code an OP_Halt due to UNIQUE or PRIMARY KEY constraint violation.
72143. Pointer to page data (for checksum)
72144. Third parameter
72145. ** Free the expression node object passed as the only argument.
72146. ** Chunk i is a free chunk that has been unlinked. Adjust its ** size parameters for check-out and return a pointer to the ** user portion of the chunk.
72147. ***** Begin file sqlite3session.h *****
72148. for the right-hand
72149. ** Each instance of this object holds a sequence of WhereLoop objects ** that implement some or all of a query plan. *** Think of each WhereLoop object as a node in a graph with arcs;** showing dependencies and costs for travelling between nodes. (That is ** not a completely accurate description because WhereLoop costs are a ** vector, not a scalar, and because dependencies are many-to-one, not ** one-to-one as are graph nodes. But it is a useful visualization aid.) ** Then a WherePath object is a path through the graph that visits some ** or all of the WhereLoop objects once. *** The "solver" works by creating the N best WherePath objects of length ** 1. Then using those as a basis to compute the N best WherePath objects ** of length 2. And so forth until the length of WherePaths equals the ** number of nodes in the FROM clause. The best (lowest cost) WherePath ** at the end is the chosen query plan.
72150. Total space in bytes required on leaf

72151. **comment:** If the client is reading or writing an index and the schema is ** not loaded, then it is too difficult to actually check to see if ** the correct locks are held.
So do not bother - just return true. ** This case does not come up very often anyhow.
label: code-design

72152. Advance at least as far as this

72153. Equality constraint on the langid column

72154. ** End of interface to code in fts5_aux.c. *****

72155. conch file host id length

72156. The pCx->pCursor will be close automatically, if it exists, by ** the call above.

72157. For sizes MX_SMALL+1 and larger

72158. SQLITE_MUTEX_W32

72159. Cursor number for ephemeral table

72160. Next character in the format string

72161. ... change the ending to this (not reversed)

72162. changeset_apply() context

72163. ** Clear the WhereLoop.u union. Leave WhereLoop.pLTerm intact.

72164. Check to see if the new index entry will be unique

72165. ** Return the numeric type for pMem, either MEM_Int or MEM_Real or both or ** none. ** ** Unlike applyNumericAffinity(), this routine does not modify pMem->flags. ** But it does set pMem->u.r and pMem->u.i appropriately.

72166. ** This function is similar to sqlite3ExprDup0, except that if pzBuffer ** is not NULL then *pzBuffer is assumed to point to a buffer large enough ** to store the copy of expression p, the copies of p->u.zToken ** (if applicable), and the copies of the p->pLeft and p->pRight expressions, ** if any. Before returning, *pzBuffer is set to the first byte past the ** portion of the buffer copied into by this function.

72167. ** If the following global variable points to a string which is the ** name of a directory, then that directory will be used to store ** temporary files. ** ** See also the "PRAGMA temp_store_directory" SQL command.

72168. The SELECT statement

72169. ** Flush the current contents of VdbeSorter.list to a new PMA, possibly ** using a background thread.

72170. Array of phrase pointers

72171. BETWEENNOTNULLLIKECASCADEDELETECASECOLLATECREATECURRENT_DATEDETACH

72172. Destructor

72173. TUNING: For simple queries, only the best path is tracked. ** For 2-way joins, the 5 best paths are followed. ** For joins of 3 or more tables, track the 10 best paths

72174. The rollback may have destroyed the pPage1->aData value. So ** call btreeGetPage() on page 1 again to make ** sure pPage1->aData is set correctly.

72175. Temporary buffer

72176. key_opt ::=

72177. A slot in pIdx->aSample[]

72178. Next in BtShared.pLock list

72179. ** Return an Expr object that refers to a memory register corresponding ** to column iCol of table pTab. ** ** regBase is the first of an array of register that contains the data ** for pTab. regBase itself holds the rowid. regBase+1 holds the first ** column. regBase+2 holds the second column, and so forth.

72180. If the schema-cookie from the database file matches the cookie ** stored with the in-memory representation of the schema, do ** not reload the schema from the database file. ** ** If virtual-tables are in use, this is not just an optimization. ** Often, v-tables store their data in other SQLite tables, which ** are queried from within xNext() and other v-table methods using ** prepared queries. If such a query is out-of-date, we do not want to ** discard the database schema, as the user code implementing the ** v-table would have to be ready for the sqlite3_vtab structure itself ** to be invalidated whenever sqlite3_step() is called from within ** a v-table method.

72181. ** If compiled with SQLITE_MUTEX_NOOP, then the no-op mutex implementation ** is used regardless of the run-time threadsafety setting.

72182. ** RBU handle. ** ** nPhaseOneStep: ** If the RBU database contains an rbu_count table, this value is set to ** a running estimate of the number of b-tree operations required to ** finish populating the *-oal file. This allows the sqlite3_bp_progress() ** API to calculate the permyriadage progress of populating the *-oal file ** using the formula: ** ** $\text{permyriadage} = (10000 * \text{nProgress}) / \text{nPhaseOneStep}$ ** ** nPhaseOneStep is initialized to the sum of: ** ** nRow * (nIndex + 1) ** ** for all source tables in the RBU database, where nRow is the number ** of rows in the source table and nIndex the number of indexes on the ** corresponding target database table. ** ** This estimate is accurate if the RBU update consists entirely of ** INSERT operations. However, it is inaccurate if: ** ** ** the RBU update contains any UPDATE operations. If the PK specified ** for an UPDATE operation does not exist in the target table, then ** no b-tree operations are required on index b-trees. Or if the ** specified PK does exist, then (nIndex*2) such operations are ** required (one delete and one insert on each index b-tree). ** ** ** the RBU update contains any DELETE operations for which the specified ** PK does not exist. In this case no operations are required on index ** b-trees. ** ** ** the RBU update contains REPLACE operations. These are similar to ** UPDATE operations. ** ** nPhaseOneStep is updated to account for the conditions above during the ** first pass of each source table. The updated nPhaseOneStep value is ** stored in the rbu_state table if the RBU update is suspended.

72183. The rowid

72184. ** The SQLITE_WITHIN(P,S,E) macro checks to see if pointer P points to ** something between S (inclusive) and E (exclusive). ** ** In other words, S is a buffer and E is a pointer to the first byte after ** the end of buffer S. This macro returns true if P points to something ** contained within the buffer S.

72185. True if RowSet.pEntry is sorted

72186. Second list to merge

72187. ** Enable or disable the extended result codes.

72188. ** Unless an "EXPLAIN QUERY PLAN" command is being processed, this function ** is a no-op. Otherwise, it adds a single row of output to the EQP result, ** where the caption is of the form: ** ** "USE TEMP B-TREE FOR xxx" ** ** where xxx is one of "DISTINCT", "ORDER BY" or "GROUP BY". Exactly which ** is determined by the zUsage argument.

72189. Write new index here

72190. The result of the merge

72191. same as TK_BITNOT, synopsis: r[P1]=~r[P1]

72192. ** Write an entry into the pointer map. ** ** This routine updates the pointer map entry for page number 'key' ** so that it maps to type 'eType' and parent page number 'pgno'. ** ** If *pRC is initially non-zero (non-SQLITE_OK) then this routine is ** a no-op. If an error occurs, the appropriate error code is written ** into *pRC.

72193. Pointer to doclist for this term

72194. **comment:** ** Finalize an iterator allocated with sqlite3changeset_start(). ** ** This function may not be called on iterators passed to a conflict handler ** callback by changeset_apply().
label: code-design

72195. It is possible to have a Blob value that has some non-zero content ** followed by zero content. But that only comes up for Blobs formed ** by the OP_MakeRecord opcode, and such Blobs never get passed into ** sqlite3MemCompare().

72196. Entries in aScan[]

72197. ** Called when a transaction starts.

72198. ** The different query plans.

72199. Same as aData for leaves. aData+4 for interior

72200. Information about aggregates at this level

72201. Space for holding column names

72202. CONFLICT

72203. 6 -> LANGUAGEID

72204. The patch

72205. ** CAPI3REF: Automatically Load Statically Linked Extensions ** ** ^This interface causes the xEntryPoint() function to be invoked for ** each new [database connection] that is created. The idea here is that ** xEntryPoint() is the entry point for a statically linked [SQLite extension] ** that is to be automatically loaded into all new database connections. ** ** ^Even though the function prototype shows that xEntryPoint() takes ** no arguments and returns void, SQLite invokes xEntryPoint() with three ** arguments and expects an integer result as if the signature of the ** entry point where as follows: ** ** <blockquote><pre> **

int xEntryPoint(** &nbs; sqlite3 *db, ** &nbs; const char **pzErrMsg, ** &nbs; const struct sqlite3_api_routines *pThunk ** &nbs;); ** </pre>
</blockquote>^ *** If the xEntryPoint routine encounters an error, it should make *pzErrMsg ** point to an appropriate error message (obtained from [sqlite3_mprintf()]) ** and return an appropriate [error code]. ^SQLite ensures that *pzErrMsg ** is NULL before calling the xEntryPoint(). ^SQLite will invoke ** [sqlite3_free()] on *pzErrMsg after xEntryPoint() returns. ^If any ** xEntryPoint() returns an error, the [sqlite3_open()], [sqlite3_open16()], ** or [sqlite3_open_v2()] call that provoked the xEntryPoint() will fail. ** ** ^Calling sqlite3_auto_extension(X) with an entry point X that is already ** on the list of automatic extensions is a harmless no-op. ^No entry point ** will be called more than once for each database connection that is opened. *** See also: [sqlite3_reset_auto_extension()] ** and [sqlite3_cancel_auto_extension()]

72206. xTruncate

72207. ** Add a new instruction to the list of instructions current in the ** VDBE. Return the address of the new instruction. *** Parameters: ** ** p Pointer to the VDBE ** ** op The opcode for this instruction ** ** p1, p2, p3 Operands ** ** Use the sqlite3VdbeResolveLabel() function to fix an address and ** the sqlite3VdbeChangeP4() function to change the value of the P4 ** operand.

72208. ** This routine checks if there is a RESERVED lock held on the specified ** file by this or any other process. If such a lock is held, return ** non-zero, otherwise zero.

72209. 236

72210. Type of expression node pExpr

72211. Column names to return

72212. If collation sequence pColl was created directly by a call to ** sqlite3_create_collation, and not generated by synthCollSeq(), ** then any copies made by synthCollSeq() need to be invalidated. ** Also, collation destructor - CollSeq.xDel() - function may need ** to be called.

72213. 147

72214. ***** Methods below this point are only available if iVersion>=1.

72215. Soft limit for node size

72216. ***** End of fts3_icu.c *****

72217. 4 if pPage is a leaf. 0 if not

72218. "GROUP" or "ORDER" or ""

72219. OUT: Error message (if rc!=SQLITE_OK)

72220. Lock all btrees used by the statement

72221. if the host id matches and the lock path already exists in the conch ** we'll try to use the path there, if we can't open that path, we'll ** retry with a new auto-generated path

72222. Statement used to read %_segdir entry

72223. Required collating sequence, if not NULL

72224. ** Advance the iterator passed as an argument to the next position. Return ** 1 if the iterator is at EOF or if it now points to the start of the ** position list for the next column.

72225. (323) vtabarglist ::= vtabarg

72226. Obtain or release the WRITER lock.

72227. True (1) if h,m,s are valid

72228. 177

72229. zWal

72230. ** Generate code that will evaluate expression pExpr and store the ** results in register target. The results are guaranteed to appear ** in register target. If the expression is constant, then this routine ** might choose to code the expression at initialization time.

72231. Mask for page offset

72232. ** Add a term to the segment being constructed by the SegmentWriter object ** *ppWriter. When adding the first term to a segment, *ppWriter should ** be passed NULL. This function will allocate a new SegmentWriter object ** and return it via the input/output variable *ppWriter in this case. *** If successful, SQLITE_OK is returned. Otherwise, an SQLite error code.

72233. Number of entries in apShm[] array

72234. ***** Begin file btree.h *****

72235. Path to this element

72236. Do not require OOMs if true

72237. Put the results here

72238. **comment:** ** Transfer error message text from an sqlite3_vtab.zErrMsg (text stored ** in memory obtained from sqlite3_malloc) into a Vdbe.zErrMsg (text stored ** in memory obtained from sqlite3DbMalloc).

label: code-design

72239. ** A list of BtShared objects that are eligible for participation ** in shared cache. This variable has file scope during normal builds, ** but the test harness needs to access it so we make it global for ** test builds. *** Access to this variable is protected by SQLITE_MUTEX_STATIC_MASTER.

72240. General heap allocations

72241. ** The %_data table is completely empty when this function is called. This ** function populates it with the initial structure objects for each index, ** and the initial version of the "averages" record (a zero-byte blob).

72242. Rowset of rows to be updated

72243. Size of this object

72244. Opcode: Gt P1 P2 P3 P4 P5 ** Synopsis: IF r[P3]>r[P1] *** This works just like the Lt opcode except that the jump is taken if ** the content of register P3 is greater than the content of ** register P1. See the Lt opcode for additional information.

72245. Cannot use an IS term from the WHERE clause as an index driver for ** the RHS of a LEFT JOIN. Such a term can only be used if it is from ** the ON clause.

72246. ** Swap all content between two VDBE structures.

72247. If there has been exactly one prior match and this match ** is for the right-hand table of a NATURAL JOIN or is in a ** USING clause, then skip this match.

72248. SQLITE_CANTOPEN

72249. CLOB

72250. IEEE floating point

72251. same as TK_LE, synopsis: IF r[P3]<=r[P1]

72252. ** Initialize zType for the new view or table.

72253. Index of cursor database in db->aDb[] (or -1)

72254. 7

72255. ** Count the number of fullsyncs and normal syncs. This is used to test ** that syncs and fullsyncs are occurring at the right times.

72256. sqlite3_test_control(SQLITE_TESTCTRL_ISINIT); *** Return SQLITE_OK if SQLite has been initialized and SQLITE_ERROR if ** not.

72257. Write number of errors seen to this variable

72258. Start offset of token

72259. If the statement has been run before (and is paused at the OP_ResultRow) ** then back it up to the point where it does the OP_SeekRowid. This could ** have been down with an extra OP_Goto, but simply setting the program ** counter is faster.

72260. val now holds the lower bound of the coordinate pair

72261. ** Print the content of a WhereTerm object

72262. 1 for writing, 0 for read-only

72263. New name of table

72264. 1120

72265. True for a write lock

72266. ***** Begin file os_win.c *****

72267. Replace the backslashes from the filename and lowercase it ** to derive a mutex name.

72268. Number of elements

72269. Data for this leaf

72270. ** Provide hints to the cursor. The particular hint given (and the type ** and number of the varargs parameters) is determined by the eHintType ** parameter. See the definitions of the BTREE_HINT_* macros for details.

72271. **comment:** ** Truncate the in-memory database file image to nPage pages. This ** function does not actually modify the database file on disk. It ** just sets the internal state of the pager object so that the ** truncation will be done when the current transaction is committed. *** This function is only called right before committing a transaction. ** Once this function has been called, the transaction must either be ** rolled back or committed. It is not safe to call this function and ** then continue writing to the database.
label: code-design

72272. ** For an indexes on expression X, locate every instance of expression X in pExpr ** and change that subexpression into a reference to the appropriate column of ** the index.

72273. FTS index object

72274. Write tail of the output list here

72275. oneselect

72276. A semicolon

72277. Verify that all sibling pages are of the same "type" (table-leaf, ** table-interior, index-leaf, or index-interior).

72278. xNext - advance a cursor

72279. If the free-list must be searched for 'nearby'

72280. Destination database name

72281. ON UPDATE

72282. Buffer to accumulate new.* record in

72283. Iteration of constraint generator loop

72284. How aggressive at syncing data to disk

72285. This is the Walker callback from checkConstraintUnchanged(). Set ** bit 0x01 of pWalker->eCode if ** pWalker->eCode to 0 if this expression node references any of the ** columns that are being modified by an UPDATE statement.

72286. TBD need to come up with better match here. Perhaps ** use FILE_ID_BOTH_DIR_INFO Structure.

72287. 168

72288. ** Delete the cell at index iCell of node pNode. After removing the ** cell, adjust the r-tree data structure if required.

72289. Number of columns in one row of the result set

72290. Begin a loop that will extract all source rows in GROUP BY order. ** This might involve two separate loops with an OP_Sort in between, or ** it might be a single loop that uses an index to extract information ** in the right order to begin with.

72291. 107

72292. IN/OUT: First token of snippet

72293. synopsis: r[P3]=func(r[P2@P5])

72294. ** Test to see whether or not the database connection is in autocommit ** mode. Return TRUE if it is and FALSE if not. Autocommit mode is on ** by default. Autocommit is disabled by a BEGIN statement and reenabled ** by the next COMMIT or ROLLBACK.

72295. IMP: R-29538-34987

72296. Copy pages from the log to the database file

72297. The clause this term is part of

72298. The BLOB containing the varints

72299. Set up the array of NodeWriter objects

72300. Array of memory cells for parent frame

72301. 26

72302. *****

72303. Size of aIndex[] array

72304. ** Finish with a read transaction. All this does is release the ** read-lock.

72305. Shared part of btree structure

72306. **comment:** ** Simplify a filename into its canonical form ** by making the following changes: ** ** * removing any trailing and duplicate / ** * convert ./ into just / ** * convert /A/.. where A is any simple name into just / ** ** Changes are made in-place. Return the new name length. ** ** The original filename is in z[0..n-1]. Return the number of ** characters in the simplified name.
label: code-design

72307. Return the header size

72308. ** Return TRUE if the given operator is one of the operators that is ** allowed for an indexable WHERE clause term. The allowed operators are ** "=", "<", ">", "<=", ">=", "IN", "IS", and "IS NULL"

72309. ** Return TRUE if the iCol-th column of index pIdx is NOT NULL

72310. Address of OP_OpenEphemeral opcode for tabTnct

72311. Page p itself has already been visited.

72312. 0x10 .. 0x1F

72313. ** Attempt to set the size of the memory mapping maintained by file ** descriptor pFd to nNew bytes. Any existing mapping is discarded. ** ** If successful, this function sets the following variables: ** ** unixFile.pMapRegion ** unixFile.mmapSize ** unixFile.mmapSizeActual ** ** If unsuccessful, an error message is logged via sqlite3_log() and ** the three variables above are zeroed. In this case SQLite should ** continue accessing the database using the xRead() and xWrite() ** methods.

72314. Destructor for pPtr (or NULL)

72315. Where the SELECT should store results

72316. Iterator to advance to next page

72317. FormatMessage returns 0 on failure. Otherwise it ** returns the number of TCHARs written to the output ** buffer, excluding the terminating null char.

72318. xFetch()'d data for this page

72319. ** Implementation of the TRIM(), LTRIM(), and RTRIM() functions. ** The userdata is 0x1 for left trim, 0x2 for right trim, 0x3 for both.

72320. Set to 1 when TABLE matches

72321. ** Return a sanitized version of the sector-size of OS file pFile. The ** return value is guaranteed to lie between 32 and MAX_SECTOR_SIZE.

72322. Content registers (after the rowid)

72323. Invalid value

72324. The cookie mask contains one bit for each database file open. ** (Bit 0 is for main, bit 1 is for temp, and so forth.) Bits are ** set for each database that is used. Generate code to start a ** transaction on each used database and to verify the schema cookie ** on each used database.

72325. ** Return a pointer to static memory containing an SQL NULL value.

72326. ** Move to the next matching rowid.

72327. ** Free a snapshot handle obtained from sqlite3_snapshot_get().

72328. Index of zColumn in row-record

72329. An attempt to read a column out of a subquery or other ** temporary table.

72330. ** Set *ppModule to a pointer to the sqlite3_tokenizer_module ** structure for the unicode tokenizer.

72331. Context object for new user-function

72332. If nRowEst is zero and there is an ORDER BY clause, ignore it. In this ** case the purpose of this call is to estimate the number of rows returned ** by the overall query. Once this estimate has been obtained, the caller ** will invoke this function a second time, passing the estimate as the ** nRowEst parameter.

72333. List of other sharable Btrees from the same db

72334. **comment:** This is the first term on a leaf that is not the leftmost leaf in ** the segment b-tree. In this case it is necessary to add a term to ** the b-tree hierarchy that is (a) larger than the largest term ** already written to the segment and (b) smaller than or equal to ** this term. In other words, a prefix of (pTerm/nTerm) that is one ** byte longer than the longest prefix (pTerm/nTerm) shares with the ** previous term. ** ** Usually, the previous term is available in pPage->term. The exception ** is if this is the first term written in an incremental-merge step. ** In this case the previous term is not available, so just write a ** copy of (pTerm/nTerm) into the parent node. This is slightly ** inefficient, but still correct.
label: code-design

72335. Assign cursor numbers to the table and all its indices.

72336. Possibly relative input path

72337. List of field names to be indexed
72338. ** Convenience functions for opening and closing files using ** sqlite3_malloc() to obtain space for the file-handle structure.
72339. In this case we would like to delete the journal file. If it is ** not possible, then that is not a problem. Deleting the journal file ** here is an optimization only. **
 ** Before deleting the journal file, obtain a RESERVED lock on the ** database file. This ensures that the journal file is not deleted ** while it is in use by some other client.
72340. ** If the inner loop was generated using a non-null pOrderBy argument, ** then the results were placed in a sorter. After the loop is terminated ** we need to run the sorter and output the results. The following ** routine generates the code needed to do that.
72341. Number of PRIMARY KEY memory cells
72342. Jump here for an error that occurs after successfully allocating ** curMain and calling sqlite3BtreeEnter(). For an error that occurs ** before that point, jump to error_out.
72343. ** Return the SorterCompare function to compare values collected by the ** sorter object passed as the only argument.
72344. ** This function may only be called while the iterator is pointing to an ** SQLITE_UPDATE or SQLITE_DELETE change (see sqlite3changeset_op()). **
 Otherwise, SQLITE_MISUSE is returned. ** ** It sets *ppValue to point to an sqlite3_value structure containing the ** iVal'th value in the old.* record. Or, if that particular value is not ** included in the record (because the change is an UPDATE and the field ** was not modified and is not a PK column), set *ppValue to NULL. ** ** If value iVal is out-of-range, SQLITE_RANGE is returned and *ppValue is ** not modified. Otherwise, SQLITE_OK.
72345. 156
72346. 309
72347. SET clause for UPDATE.
72348. If this is to be the first rowid written to the page, set the ** rowid-pointer in the page-header. Also append a value to the dliidx ** buffer, in case a doclist-index is required.
72349. sqlite3AssertMayAbort() logic
72350. Size of buffer p in bytes
72351. Index in pWC of the next virtual term
72352. The current leaf node is full. Write it out to the database.
72353. 4 billion rows
72354. Pointer to left-most node of this depth
72355. If p!=0, it must match the iOff value.
72356. Grow the output buffer if required.
72357. 13
72358. **comment:** These use local variables, so do them in a separate routine ** to avoid having to move the frame pointer in the common case
 label: code-design
72359. True after xBegin but before xCommit/xRollback
72360. SQLITE_OS_UNIX
72361. EVIDENCE-OF: R-25008-21688 The size of a page is a power of two ** between 512 and 65536 inclusive.
72362. Skip over position list
72363. ** Initialize SQLite. ** ** This routine must be called to initialize the memory allocation, ** VFS, and mutex subsystems prior to doing any serious work with **
 SQLite. But as long as you do not compile with SQLITE_OMIT_AUTOINIT ** this routine will be called automatically by key routines such as **
 sqlite3_open(). ** ** This routine is a no-op except on its very first call for the process, ** or for the first call after a call to sqlite3_shutdown. ** ** The first
 thread to call this routine runs the initialization to ** completion. If subsequent threads call this routine before the first ** thread has finished the initialization
 process, then the subsequent ** threads must block until the first thread finishes with the initialization. ** ** The first thread might call this routine recursively.
 Recursive ** calls to this routine should not block, of course. Otherwise the ** initialization process would never complete. ** ** Let X be the first thread to enter
 this routine. Let Y be some other ** thread. Then while the initial invocation of this routine by X is ** incomplete, it is required that: ** ** * Calls to this routine
 from Y must block until the outer-most ** call by X completes. ** ** * Recursive calls to this routine from thread X return immediately ** without blocking.
72364. MSVC is picky about pulling func ptrs from va lists. ** <http://support.microsoft.com/kb/47961> ** sqlite3GlobalConfig.xTestCallback = va_arg(ap, int(*)(int));
72365. Filename in OS encoding
72366. Point the object iterator at the first object
72367. First byte of cell content area
72368. **comment:** Each unsigned integer in the following array corresponds to a contiguous ** range of unicode codepoints that are not either letters or numbers (i.e. **
 codepoints for which this function should return 0). ** ** The most significant 22 bits in each 32-bit value contain the first ** codepoint in the range. The least
 significant 10 bits are used to store ** the size of the range (always at least 1). In other words, the value ** ((C<<22) + N) represents a range of N codepoints
 starting with codepoint ** C. It is not possible to represent a range larger than 1023 codepoints ** using this format.
 label: code-design
72369. The authorization context
72370. end_block
72371. This is an UPDATE. Foreign key processing is only required if the ** operation modifies one or more child or parent key columns.
72372. ** CAPI3REF: Configure database connections ** METHOD: sqlite3 ** ** The sqlite3_db_config() interface is used to make configuration ** changes to a
 [database connection]. The interface is similar to ** [sqlite3_config()] except that the changes apply to a single ** [database connection] (specified in the first
 argument). ** ** The second argument to sqlite3_db_config(D,V,...) is the ** [SQLITE_DBCONFIG_LOOKASIDE | configuration verb] - an integer code ** that
 indicates what aspect of the [database connection] is being configured. ** Subsequent arguments vary depending on the configuration verb. ** ** ^Calls to
 sqlite3_db_config() return SQLITE_OK if and only if ** the call is considered successful.
72373. Currently in a VACUUM
72374. For an UPDATE, memory cell (p->iNewReg+1+iIdx) contains the required ** value. Make a copy of the cell contents and return a pointer to it. ** It is not safe to
 return a pointer to the memory cell itself as the ** caller may modify the value text encoding.
72375. ** Two routines for printing the content of an sqlite3_index_info ** structure. Used for testing and debugging only. If neither ** SQLITE_TEST or
 SQLITE_DEBUG are defined, then these routines ** are no-ops.
72376. Write a table header
72377. Enable automatic indexes
72378. Statements to read/write/delete a record from xxx_node
72379. Regardless of the strategy selected, FTS can deliver rows in rowid (or ** docid) order. Both ascending and descending are possible.
72380. The conch file contains the header, host id and lock file path
72381. This may fail if SQLite value p contains a utf-16 string that must ** be converted to utf-8 and an OOM error occurs while doing so.
72382. Usually, regResult is the first cell in an array of memory cells ** containing the current result row. In this case regOrig is set to the ** same value. However, if the
 results are being sent to the sorter, the ** values for any expressions that are also part of the sort-key are omitted ** from this array. In this case regOrig is set to
 zero.
72383. ** Deallocate a KeyInfo object
72384. **comment:** ** Implement steps 3, 4, and 5 of the pcache1Fetch() algorithm described ** in the header of the pcache1Fetch() procedure. ** ** This steps are broken
 out into a separate procedure because they are ** usually not needed, and by avoiding the stack initialization required ** for these steps, the main pcache1Fetch()
 procedure can run faster.
 label: code-design
72385. Return True
72386. The index associated with pLoop
72387. Compute data for all columns of the new entry, beginning ** with the first column.
72388. If there is a vector IN term - e.g. "(a, b) IN (SELECT ...)" - create ** a virtual term for each vector component. The expression object ** used by each such virtual
 term is pExpr (the full vector IN(...) ** expression). The WhereTerm.iField variable identifies the index within ** the vector on the LHS that the virtual term
 represents. ** ** This only works if the RHS is a simple SELECT, not a compound
72389. There are pages on the freelist. Reuse one of those pages.
72390. Lhs input changeset
72391. Offset to the cell content area

72392. **comment:** ** Takes an already filled in unix file and alters it so all file locking ** will be performed on the local proxy lock file. The following fields ** are preserved in the locking context so that they can be restored and ** the unix structure properly cleaned up at close time: ** ->lockingContext ** ->pMethod
label: code-design

72393. SQLITE OMIT_COMPLETE

72394. True when processing first term on page

72395. SQL statement 1 (see above)

72396. **comment:** ** When SQLITE OMIT_WSD is defined, it means that the target platform does ** not support Writable Static Data (WSD) such as global and static variables. ** All variables must either be on the stack or dynamically allocated from ** the heap. When WSD is unsupported, the variable declarations scattered ** throughout the SQLite code must become constants instead. The SQLITE_WSD ** macro is used for this purpose. And instead of referencing the variable ** directly, we use its constant as a key to lookup the run-time allocated ** buffer that holds real variable. The constant is also the initializer ** for the run-time allocated buffer. ** In the usual case where WSD is supported, the SQLITE_WSD and GLOBAL ** macros become no-ops and have zero performance impact.
label: code-design

72397. Identifier from "INDEXED BY <zIndex>" clause

72398. Size of aDoclist[] in bytes

72399. blocks are atomic

72400. defer_subclause ::= NOT DEFERRABLE init_deferred_pred_opt

72401. Opcode: MustBeInt P1 P2 * * * * * Force the value in register P1 to be an integer. If the value ** in P1 is not an integer and cannot be converted into an integer ** without data loss, then jump immediately to P2, or if P2==0 ** raise an SQLITE_MISMATCH exception.

72402. ** This function opens a cursor used to read the input data for an ** incremental merge operation. Specifically, it opens a cursor to scan ** the oldest nSeg segments (idx=0 through idx=(nSeg-1)) in absolute ** level iAbsLevel.

72403. Check to see if a unixShmNode object already exists. Reuse an existing ** one if present. Create a new one if necessary.

72404. ** Free all memory allocations in the pParse object

72405. **comment:** ** CAPI3REF: Online Backup API. ** ** The backup API copies the content of one database into another. ** It is useful either for creating backups of databases or ** for copying in-memory databases to or from persistent files. ** ** See Also: [Using the SQLite Online Backup API] ** ** ^SQLite holds a write transaction open on the destination database file ** for the duration of the backup operation. ** ^The source database is read-locked only while it is being read; ** it is not locked continuously for the entire backup operation. ** ^Thus, the backup may be performed on a live source database without ** preventing other database connections from ** reading or writing to the source database while the backup is underway. ** ** ^To perform a backup operation: ** ** sqlite3_backup_init() is called once to initialize the ** backup, ** sqlite3_backup_step() is called one or more times to transfer ** the data between the two databases, and finally ** sqlite3_backup_finish() is called to release all resources ** associated with the backup operation. ** ** There should be exactly one call to sqlite3_backup_finish() for each ** successful call to sqlite3_backup_init(). ** ** [[sqlite3_backup_init()]] sqlite3_backup_init() ** ** ^The D and N arguments to sqlite3_backup_init(D,N,S,M) are the ** [database connection] associated with the destination database ** and the database name, respectively. ** ^The database name is "main" for the main database, "temp" for the ** temporary database, or the name specified after the AS keyword in ** an [ATTACH] statement for an attached database. ** ^The S and M arguments passed to ** sqlite3_backup_init(D,N,S,M) identify the [database connection] ** and database name of the source database, respectively. ** ^The source and destination [database connections] (parameters S and D) ** must be different or else sqlite3_backup_init(D,N,S,M) will fail with ** an error. ** ** ^A call to sqlite3_backup_init() will fail, returning NULL, if ** there is already a read or read-write transaction open on the ** destination database. ** ** ^If an error occurs within sqlite3_backup_init(D,N,S,M), then NULL is ** returned and an error code and error message are stored in the ** destination [database connection] D. ** ^The error code and message for the failed call to sqlite3_backup_init() ** can be retrieved using the [sqlite3_errcode()], [sqlite3_errmsg()], and/or ** [sqlite3_errmsg16()] functions. ** ^A successful call to sqlite3_backup_init() returns a pointer to an ** [sqlite3_backup] object. ** ^The [sqlite3_backup] object may be used with the sqlite3_backup_step() and ** sqlite3_backup_finish() functions to perform the specified backup ** operation. ** ** [[sqlite3_backup_step()]] sqlite3_backup_step() ** ** ^Function sqlite3_backup_step(B,N) will copy up to N pages between ** the source and destination databases specified by [sqlite3_backup] object B. ** ^If N is negative, all remaining source pages are copied. ** ^If sqlite3_backup_step(B,N) successfully copies N pages and there ** are still more pages to be copied, then the function returns [SQLITE_OK]. ** ^If sqlite3_backup_step(B,N) successfully finishes copying all pages ** from source to destination, then it returns [SQLITE_DONE]. ** ^If an error occurs while running sqlite3_backup_step(B,N), ** then an [error code] is returned. ^As well as [SQLITE_OK] and ** [SQLITE_DONE], a call to sqlite3_backup_step() may return [SQLITE_READONLY], ** [SQLITE_NOMEM], [SQLITE_BUSY], [SQLITE_LOCKED], or an ** [SQLITE_IOERR_ACCESS | SQLITE_IOERR_XXX] extended error code. ** ** ^The sqlite3_backup_step() might return [SQLITE_READONLY] if ** ** the destination database was opened read-only, or ** the destination database is using write-ahead-log journaling ** and the destination and source page sizes differ, or ** the destination database is an in-memory database and the ** destination and source page sizes differ. ** ** ^If sqlite3_backup_step() cannot obtain a required file-system lock, then ** the [sqlite3_busy_handler | busy-handler function] ** is invoked (if one is specified). ^If the ** busy-handler returns non-zero before the lock is available, then ** [SQLITE_BUSY] is returned to the caller. ^In this case the call to ** sqlite3_backup_step() can be retried later. ^If the source ** [database connection] ** is being used to write to the source database when sqlite3_backup_step() ** is called, then [SQLITE_LOCKED] is returned immediately. ^Again, in this ** case the call to sqlite3_backup_step() can be retried later on. ^If ** [SQLITE_IOERR_ACCESS | SQLITE_IOERR_XXX], [SQLITE_NOMEM], or ** [SQLITE_READONLY] is returned, then ** there is no point in retrying the call to sqlite3_backup_step(). These ** errors are considered fatal. ^The application must accept ** that the backup operation has failed and pass the backup operation handle ** to the sqlite3_backup_finish() to release associated resources. ** ** ^The first call to sqlite3_backup_step() obtains an exclusive lock ** on the destination file. ^The exclusive lock is not released until either ** sqlite3_backup_finish() is called or the backup operation is complete ** and sqlite3_backup_step() returns [SQLITE_DONE]. ^Every call to ** sqlite3_backup_step() obtains a [shared lock] on the source database that ** lasts for the duration of the sqlite3_backup_step() call. ** ^Because the source database is not locked between calls to ** sqlite3_backup_step(), the source database may be modified mid-way ** through the backup process. ^If the source database is modified by an ** external process or via a database connection other than the one being ** used by the backup operation, then the backup will be automatically ** restarted by the next call to sqlite3_backup_step(). ^If the source ** database is modified by the using the same database connection as is used ** by the backup operation, then the backup database is automatically ** updated at the same time. ** ** [[sqlite3_backup_finish()]] sqlite3_backup_finish() ** ** ^When sqlite3_backup_step() has returned [SQLITE_DONE], or when the ** application wishes to abandon the backup operation, the application ** should destroy the [sqlite3_backup] by passing it to sqlite3_backup_finish(). ** ^The sqlite3_backup_finish() interfaces releases all ** resources associated with the [sqlite3_backup] object. ** ^If sqlite3_backup_step() has not yet returned [SQLITE_DONE], then any ** active write-transaction on the destination database is rolled back. ** The [sqlite3_backup] object is invalid ** and may not be used following a call to sqlite3_backup_finish(). ** ** ^The value returned by sqlite3_backup_finish is [SQLITE_OK] if no ** sqlite3_backup_step() errors occurred, regardless of whether or not ** sqlite3_backup_step() completed. ** ^If an out-of-memory condition or IO error occurred during any prior ** sqlite3_backup_step() call on the same [sqlite3_backup] object, then ** sqlite3_backup_finish() returns the corresponding [error code]. ** ** ^A return of [SQLITE_BUSY] or [SQLITE_LOCKED] from sqlite3_backup_step() ** is not a permanent error and does not affect the return value of ** sqlite3_backup_finish(). ** ** [[sqlite3_backup_remaining()]] [[sqlite3_backup_pagecount()]] ** sqlite3_backup_remaining() and sqlite3_backup_pagecount() ** ** ^The sqlite3_backup_remaining() routine returns the number of pages still ** to be backed up at the conclusion of the most recent sqlite3_backup_step(). ** ^The sqlite3_backup_pagecount() routine returns the total number of pages ** in the source database at the conclusion of the most recent ** sqlite3_backup_step(). ** ^The values returned by these functions are only updated by ** sqlite3_backup_step(). If the source database is modified in a way that ** changes the size of the source database or the number of pages remaining, ** those changes are not reflected in the output of sqlite3_backup_pagecount() ** and sqlite3_backup_remaining() until after the next ** sqlite3_backup_step(). ** ** ^B>Concurrent Usage of Database Handles ** ** ^The source [database connection] may be used by the application for other ** purposes while a backup operation is underway or being initialized. ** ^If SQLite is compiled and configured to support threadsafe database ** connections, then the source database connection may be used concurrently ** from within other threads. ** ** However, the application must guarantee that the destination ** [database connection] is not passed to any other API (by any thread) after ** sqlite3_backup_init() is called and before the corresponding call to ** sqlite3_backup_finish(). SQLite does not currently check to see ** if the application incorrectly accesses the destination [database connection] ** and so no error code is reported, but the operations may malfunction ** nevertheless. Use of the destination database connection while a ** backup is in progress might also cause a mutex deadlock. ** ** If running in [shared cache mode], the application must ** guarantee that the shared cache used by the destination database ** is not accessed while the backup is running. In practice this means ** that the application must guarantee that the disk file being ** backed up to is not accessed by any connection within the process, ** not just the specific connection that was passed to sqlite3_backup_init(). ** ** The [sqlite3_backup] object itself is partially threadsafe. Multiple ** threads may safely make multiple concurrent calls to sqlite3_backup_step(). ** However, the sqlite3_backup_remaining() and sqlite3_backup_pagecount() ** APIs are not strictly speaking threadsafe. If they are invoked at the ** same time as another thread is invoking sqlite3_backup_step() it is ** possible that they return invalid values.
label: code-design

72406. **comment:** ** The SQLITE_THREADSAFE macro must be defined as 0, 1, or 2. ** 0 means mutexes are permanently disable and the library is never ** threadsafe. 1 means the library is serialized which is the highest ** level of threadsafety. 2 means the library is multithreaded - multiple ** threads can use SQLite as long as no two threads try to use the same ** database connection at the same time. ** ** Older versions of SQLite used an optional THREADSAFE macro. ** We support that for legacy. ** ** To ensure that the correct value of "THREADSAFE" is reported when querying ** for compile-time options at runtime (e.g. "PRAGMA compile_options"), this ** logic is partially replicated in ctime.c. If it is updated here, it should ** also be updated there.

label: code-design

72407. Saved value of db->flags

72408. VFS to use for this b-tree

72409. Wrong number of arguments means "no match"

72410. ** Implementation of xSync() method. Flush the contents of the pending-terms ** hash-table to the database.

72411. COLUMNKW => ID

72412. **comment:** Give a better score to a function with a specific number of arguments ** than to function that accepts any number of arguments.

label: code-design

72413. Clear MemPage.isInit to make sure the corruption detection code in ** btreeInitPage() is executed.

72414. An ephemeral table

72415. Assuming the record contains N fields, the record format looks ** like this: ** *-----** | hdr-size | type 0
| type 1 | ... | type N-1 | data0 | ... | data N-1 | ** -----** Data(0) is taken from register P1. Data(1) comes from register P1+1 ** and so forth. ** ** Each type field is a varint representing the serial type of the ** corresponding data element (see sqlite3VdbeSerialType()). The ** hdr-size field is also a varint which is the offset from the beginning ** of the record to data0.

72416. WHERE clause terms

72417. ** A structure to store values read from the rbu_state table in memory.

72418. If the ORDER BY clause contains only columns in the current ** virtual table then allocate space for the aOrderBy part of ** the sqlite3_index_info structure.

72419. ** Powersafe overwrite is on by default. But can be turned off using ** the -DSQLITE_POWERSAFE_OVERWRITE=0 command-line option.

72420. True for an ephemeral table

72421. The new object must be linked on to the end of the list, not ** simply added to the start of it. This is to ensure that the ** tables within the output of sqlite3changegroup_output() are in ** the right order.

72422. Release the page reference.

72423. ** Constants for the largest and smallest possible 64-bit signed integers.

72424. SQL to be evaluated

72425. Either an error, or no such table.

72426. ** Append a new element to the given IdList. Create a new IdList if ** need be. ** ** A new IdList is returned, or NULL if malloc() fails.

72427. ** Register built-in functions used to help implement ALTER TABLE

72428. The new state to shift in

72429. WHERE clause for partial indices

72430. Name of this database. (schema name, not filename)

72431. True for select lists like "count(*)"

72432. IN/OUT: Incr. by number tokens inserted

72433. Parent tokenizer module

72434. xRead

72435. The score for this node. Smallest goes first.

72436. If the destination is DistQueue, then cursor (iParm+1) is open ** on a second ephemeral index that holds all values every previously ** added to the queue.

72437. ** Report an error that an expression is not valid for some set of ** pNC->ncFlags values determined by validMask.

72438. if exponent is present

72439. ** Determine the largest segment index value that exists within absolute ** level iAbsLevel+1. If no error occurs, set *piIdx to this value plus ** one before returning SQLITE_OK. Or, if there are no segments at all ** within level iAbsLevel, set *piIdx to zero. ** ** If an error occurs, return an SQLite error code. The final value of ** *piIdx is undefined in this case.

72440. ** Advance the cursor to the next entry in the database. ** Return value: ** ** SQLITE_OK success ** SQLITE_DONE cursor is already pointing at the last element ** otherwise some kind of error occurred ** ** The main entry point is sqlite3BtreeNext(). That routine is optimized ** for the common case of merely incrementing the cell counter BtCursor.aiIdx ** to the next cell on the current page. The (slower) btreeNext() helper ** routine is called when it is necessary to move to a different page or ** to restore the cursor. ** ** If bit 0x01 of the F argument in sqlite3BtreeNext(C,F) is 1, then the ** cursor corresponds to an SQL index and this routine could have been ** skipped if the SQL index had been a unique index. The F argument ** is a hint to the implement. SQLite btree implementation does not use ** this hint, but COMDB2 does.

72441. Function for logging

72442. Output character index

72443. We cannot be in random rowid mode if this is ** an AUTOINCREMENT table.

72444. True if src db requires unlocking

72445. Current index is unique

72446. Number of input segments

72447. ** CAPI3REF: Enable Or Disable A Session Object ** ** Enable or disable the recording of changes by a session object. When ** enabled, a session object records changes made to the database. When ** disabled - it does not. A newly created session object is enabled. ** Refer to the documentation for [sqlite3session_changeset()] for further ** details regarding how enabling and disabling a session object affects ** the eventual changesets. ** ** Passing zero to this function disables the session. Passing a value ** greater than zero enables it. Passing a value less than zero is a ** no-op, and may be used to query the current state of the session. ** ** The return value indicates the final state of the session object: 0 if ** the session is disabled, or 1 if it is enabled.

72448. ** Acquire a lock on the handle h

72449. **comment:** A memory allocation of a number of bytes which is near the maximum ** signed integer value might cause an integer overflow inside of the ** xMalloc(). Hence we limit the maximum size to 0x7fffff00, giving ** 255 bytes of overhead. SQLite itself will never use anything near ** this amount. The only way to reach the limit is with sqlite3_malloc()

label: code-design

72450. Add pPage to the dirty list

72451. ** Implementation of xOpen method.

72452. ** Arguments aLeft and aRight are pointers to change records for table pTab. ** This function returns true if the two records apply to the same row (i.e. ** have the same values stored in the primary key columns), or false ** otherwise.

72453. Final leaf to traverse

72454. The text of the modifier

72455. Initialize the node to no-match

72456. 27

72457. ** Merge two lists of pages connected by pDirty and in pgno order. ** Do not bother fixing the pDirtyPrev pointers.

72458. 29

72459. Number of references to this node

72460. 6

72461. xConnect - connect to an existing table

72462. ** Change the string value of an sqlite3_value object

72463. Set up the input stream

72464. If bCommit is zero, this loop runs exactly once and page pLastPg ** is swapped with the first free page pulled off the free list. ** ** On the other hand, if bCommit is greater than zero, then keep ** looping until a free-page located within the first nFin pages ** of the file is found.

72465. Get the exclusive locks at the system level. Then if successful ** also mark the local connection as being locked.

72466. Add this many extra columns to the end

72467. Table cursor

72468. ***** End of fts3_hash.h *****

72469. Provide the ability to easily simulate an I/O error during testing

72470. Table whose indices are to be analyzed

72471. This loop runs once for each destination page spanned by the source ** page. For each iteration, variable iOff is set to the byte offset ** of the destination page.

72472. Close any open statement handles.

72473. Set of pages moved to free-list this transaction

72474. Get the conversion type modifier

72475. 88..8f

72476. "stat" column of stat1 table

72477. e0..e7

72478. 50..57 PQRSTUVW

72479. First arg to busy callback

72480. Composite sort order

72481. 1st argument to the access auth function

72482. Max frame that can be backfilled

72483. Output operand

72484. **comment:** TODO: Check that both arguments are non-NULL
label: requirement

72485. On Android, ftruncate() always uses 32-bit offsets, even if ** _FILE_OFFSET_BITS=64 is defined. This means it is unsafe to attempt to ** truncate a file to any size larger than 2GiB. Silently ignore any ** such attempts.

72486. Buffer containing term

72487. Set iBestLvl to the level to read input segments from.

72488. Pointer to application-specific data

72489. The WHERE_* flags defined in sqliteInt.h

72490. If zName is NULL, the upper layer is requesting a temp file.

72491. ** Create a new mask for cursor iCursor. ** ** There is one cursor per table in the FROM clause. The number of ** tables in the FROM clause is limited by a test early in the ** sqlite3WhereBegin() routine. So we know that the pMaskSet->ix[] ** array will never overflow.

72492. Never called with prior errors

72493. Number of cells on this page, local and ovfl

72494. As there exists an unusable MATCH constraint this is an ** unusable plan. Set a prohibitively high cost.

72495. Information for internal btree tables

72496. 3

72497. azColumn

72498. ***** Include sqlite3ext.h in the middle of loadext.c *****

72499. vtabarg ::=

72500. Constant factoring

72501. Information desired. SQLITE_SCANSTAT_*

72502. Generate the expected index checksum based on the contents of the ** %_content table. This block stores the checksum in ctx.cksum.

72503. Open all tables in the pTabList and any indices selected for ** searching those tables.

72504. OUTPUT: True if column is auto-increment

72505. ** Attempt to allocate and return a pointer to a zeroed block of nByte ** bytes. ** ** If an error (i.e. an OOM condition) occurs, return NULL and leave an ** error code in the rbu handle passed as the first argument. Or, if an ** error has already occurred when this function is called, return NULL ** immediately without attempting the allocation or modifying the stored ** error code.

72506. 910

72507. xDisconnect

72508. Name of the database

72509. ** Read a 64-bit variable-length integer from memory starting at p[0]. ** Return the number of bytes read, or 0 on error. ** The value is stored in *v.

72510. Used internally - xBestIndex should ignore

72511. Number of leaf pages yet to be written

72512. Name of database file

72513. ** The following global variable is incremented every time a cursor ** moves, either by the OP_SeekXX, OP_Next, or OP_Prev opcodes. The test ** procedures use this information to make sure that indices are ** working correctly. This variable has no function other than to ** help verify the correct operation of the library.

72514. Register holding rowid of CREATE TABLE entry

72515. We are forced to roll back the active transaction. Before doing ** so, abort any other statements this handle currently has active.

72516. ** Register a function to be invoked when a transaction commits. ** If the invoked function returns non-zero, then the commit becomes a ** rollback.

72517. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, append a blob of data to the buffer. ** ** If an OOM condition is encountered, set *pRc to SQLITE_NOMEM before ** returning.

72518. same as TK_FLOAT, out2

72519. **comment:** ** This is the fallback stemmer used when the porter stemmer is ** inappropriate. The input word is copied into the output with ** US-ASCII case folding. If the input word is too long (more ** than 20 bytes if it contains no digits or more than 6 bytes if ** it contains digits) then word is truncated to 20 or 6 bytes ** by taking 10 or 3 bytes from the beginning and end.
label: code-design

72520. OS functions to use for IO

72521. Pointer to value 1

72522. The expression to evaluate

72523. End of [...]

72524. ** Create a new tokenizer. The values in the argv[] array are the ** arguments passed to the "tokenizer" clause of the CREATE VIRTUAL ** TABLE statement that created the fts3 table. For example, if ** the following SQL is executed: ** ** CREATE .. USING fts3(... , tokenizer <tokenizer-name> arg1 arg2) ** ** then argc is set to 2, and the argv[] array contains pointers ** to the strings "arg1" and "arg2". ** ** This method should return either SQLITE_OK (0), or an SQLite error ** code. If SQLITE_OK is returned, then *ppTokenizer should be set ** to point at the newly created tokenizer structure. The generic ** sqlite3_tokenizer.pModule variable should not be initialized by ** this callback. The caller will do so.

72525. Previous result register. No uniqueness if 0

72526. Index to read from

72527. defined(SQLITE_MUTEX_OMIT)

72528. Bytes of space to allocate at pRet

72529. SQLITE_ENABLE_CURSOR_HINTS

72530. Maximum difference in token positions

72531. ** Set the value returned by the sqlite3_last_insert_rowid() API function.

72532. ** CAPI3REF: Obtain A Composite Changeset From A Changegroup ** ** Obtain a buffer containing a changeset (or patchset) representing the ** current contents of the changegroup. If the inputs to the changegroup ** were themselves changesets, the output is a changeset. Or, if the ** inputs were patchsets, the output is also a patchset. ** ** As with the output of the sqlite3session_changeset() and ** sqlite3session_patchset() functions, all changes related to a single ** table are grouped together in the output of this function. Tables appear ** in the same order as for the very first changeset added to the changegroup. ** If the second or subsequent changesets added to the changegroup contain ** changes for tables that do not appear in the first changeset, they are ** appended onto the end of the output changeset, again in the order in ** which they are first encountered. ** ** If an error occurs, an SQLite error code is returned and the output ** variables (*pnData) and (*ppData) are set to 0. Otherwise, SQLITE_OK ** is returned and the output variables are set to the size of and a ** pointer to the output buffer, respectively. In this case it is the ** responsibility of the caller to eventually free the buffer using a ** call to sqlite3_free().

72533. If iCol is less than zero, then the expression requests the ** rowid of the sub-select or view. This expression is legal (see ** test case misc2.2.2) - it always evaluates to NULL.

72534. expr ::= expr in_op nm dbnm paren_explist

72535. Index of *output* segment in iAbsLevel+1

72536. ** Is the sqlite3ErrName() function needed in the build? Currently, ** it is needed by "mutex_w32.c" (when debugging), "os_win.c" (when ** OSTRACE is enabled), and by several "test*.c" files (which are ** compiled using SQLITE_TEST).

72537. CONFLICTCROSSCURRENT_TIMESTAMPPRIMARYDEFERREDISTINCTDROPFAIL

72538. At this point: ** iFreeBlk: First freeblock after iStart, or zero if none ** iPtr: The address of a pointer to iFreeBlk *** Check to see if iFreeBlk should be coalesced onto the end of iStart.

72539. Next free slot in %_segments

72540. ** Open the database handle and attach the RBU database as "rbu". If an ** error occurs, leave an error code and message in the RBU handle.

72541. Candidate values for BtLock.eLock

72542. If requesting a write-lock, then the Btree must have an open write ** transaction on this file. And, obviously, for this to be so there ** must be an open write transaction on the file itself.

72543. OUT: size of doclist in bytes

72544. **comment:** It used to be that sqlite3PcacheMakeClean(pPg) was called here. But ** that call was dangerous and had no detectable benefit since the cache ** is normally cleaned by sqlite3PcacheCleanAll() after rollback and so ** has been removed.
label: code-design

72545. The global PGroup for mode (2)

72546. Check for the zero-length marker in the %_segments table

72547. ***** Interface to code in fts5_aux.c.

72548. ***** End of lemon-generated parsing tables *****

72549. **comment:** ** CAPI3REF: Create Or Redefine SQL Functions ** KEYWORDS: {function creation routines} ** KEYWORDS: {application-defined SQL function} ** KEYWORDS: {application-defined SQL functions} ** METHOD: sqlite3 *** ^These functions (collectively known as "function creation routines") ** are used to add SQL functions or aggregates or to redefine the behavior ** of existing SQL functions or aggregates. The only differences between ** these routines are the text encoding expected for ** the second parameter (the name of the function being created) ** and the presence or absence of a destructor callback for ** the application data pointer. *** ^The first parameter is the [database connection] to which the SQL ** function is to be added. ^If an application uses more than one database ** connection then application-defined SQL functions must be added ** to each database connection separately. *** ^The second parameter is the name of the SQL function to be created or ** redefined. ^The length of the name is limited to 255 bytes in a UTF-8 ** representation, exclusive of the zero-terminator. ^Note that the name ** length limit is in UTF-8 bytes, not characters nor UTF-16 bytes. ** ^Any attempt to create a function with a longer name ** will result in [SQLITE_MISUSE] being returned. *** ^The third parameter (nArg) ** is the number of arguments that the SQL function or ** aggregate takes. ^If this parameter is -1, then the SQL function or ** aggregate may take any number of arguments between 0 and the limit ** set by [sqlite3_limit] ([SQLITE_LIMIT_FUNCTION_ARG]). If the third ** parameter is less than -1 or greater than 127 then the behavior is ** undefined. *** ^The fourth parameter, eTextRep, specifies what ** [SQLITE_UTF8 | text encoding] this SQL function prefers for ** its parameters. The application should set this parameter to ** [SQLITE_UTF16LE] if the function implementation invokes ** [sqlite3_value_text16le()] on an input, or [SQLITE_UTF16BE] if the ** implementation invokes [sqlite3_value_text16be()] on an input, or ** [SQLITE_UTF16] if [sqlite3_value_text16()] is used, or [SQLITE_UTF8] ** otherwise. ^The same SQL function may be registered multiple times using ** different preferred text encodings, with different implementations for ** each encoding. *** ^When multiple implementations of the same function are available, SQLite ** will pick the one that involves the least amount of data conversion. *** ^The fourth parameter may optionally be ORed with [SQLITE_DETERMINISTIC] ** to signal that the function will always return the same result given ** the same inputs within a single SQL statement. Most SQL functions are ** deterministic. The built-in [random()] SQL function is an example of a ** function that is not deterministic. The SQLite query planner is able to ** perform additional optimizations on deterministic functions, so use ** of the [SQLITE_DETERMINISTIC] flag is recommended where possible. *** ^The fifth parameter is an arbitrary pointer. The implementation of the ** function can gain access to this pointer using [sqlite3_user_data()].^ ** ^The sixth, seventh and eighth parameters, xFunc, xStep and xFinal, are ** pointers to C-language functions that implement the SQL function or ** aggregate. ^A scalar SQL function requires an implementation of the xFunc ** callback only; NULL pointers must be passed as the xStep and xFinal ** parameters. ^An aggregate SQL function requires an implementation of xStep ** and xFinal and NULL pointer must be passed for xFunc. ^To delete an existing ** SQL function or aggregate, pass NULL pointers for all three function ** callbacks. *** ^If the ninth parameter to sqlite3_create_function_v2() is not NULL, ** then it is destructor for the application data pointer. ** The destructor is invoked when the function is deleted, either by being ** overloaded or when the database connection closes.)^ ** ^The destructor is also invoked if the call to ** sqlite3_create_function_v2() fails. ** ^When the destructor callback of the tenth parameter is invoked, it ** is passed a single argument which is a copy of the application data ** pointer which was the fifth parameter to sqlite3_create_function_v2(). ** ^It is permitted to register multiple implementations of the same ** functions with the same name but with either differing numbers of ** arguments or differing preferred text encodings. ^SQLite will use ** the implementation that most closely matches the way in which the ** SQL function is used. ^A function implementation with a non-negative ** nArg parameter is a better match than a function implementation with ** a negative nArg. ^A function where the preferred text encoding ** matches the database encoding is a better ** match than a function where the encoding is different. ** ^A function where the encoding difference is between UTF16le and UTF16be ** is a closer match than a function where the encoding difference is ** between UTF8 and UTF16. *** ^Built-in functions may be overloaded by new application-defined functions. *** ^An application-defined function is permitted to call other ** SQLite interfaces. However, such calls must not ** close the database connection nor finalize or reset the prepared ** statement in which the function is running.
label: code-design

72550. ** Query a blob handle for the size of the data. ** The Incrblob.nByte field is fixed for the lifetime of the Incrblob ** so no mutex is required for access.

72551. EP_xIsSelect and op = IN, EXISTS, SELECT

72552. Result of an sqlite3BtreeLast()

72553. Number of zero bytes at the end of the record

72554. Register allocations

72555. ** Private interfaces - callable only by other where.c routines. *** where.c:

72556. Iterate in descending rowid order

72557. Return address register for reset subroutine

72558. assert(p->rc==SQLITE_OK);

72559. True if iPrev is valid

72560. flag values passed to the btree insert

72561. If this is the first header read from the journal, truncate the ** database file back to its original size.

72562. Sync the header (unless SQLITE_IOCAP_SEQUENTIAL is true or unless ** all syncing is turned off by PRAGMA synchronous=OFF). Otherwise ** an out-of-order write following a WAL restart could result in ** database corruption. See the ticket: *** http://localhost:591/sqlite/info/ff5be73dee

72563. Value to be set

72564. 1-byte header, 16-byte host id, path

72565. ** Add the size of memory allocation "p" to the count in ** db->pnBytesFreed.

72566. The pager calls this method to signal that it has done ** a rollback and that the database is therefore unchanged and ** it hence it is OK for the transaction change counter to be ** unchanged.

72567. Number of outputs for the new entry

72568. Assert the truth of VdbeCoverageAlwaysTaken() and ** VdbeCoverageNeverTaken()

72569. Size of changeset in bytes

72570. Number of arguments. -1 means unlimited

72571. ** Implementation of the sqlite3_pcache.xShrink method. *** Free up as much memory as possible.

72572. ***** No-op Locking *****
***** Of the various locking implementations available, this is by far the ** simplest: locking is ignored. No attempt is made to lock the database ** file for reading or writing. *** This locking mode is appropriate for use on read-only databases ** (ex: databases that are burned into CD-ROM, for example.) It can ** also be used if the application employs some external mechanism to ** prevent simultaneous access of the same database by two or more ** database connections. But there is a serious risk of database ** corruption if this locking mode is used in situations where multiple ** database connections are accessing the same database file at the same ** time and one or more of those connections are writing.

72573. Iterator to read values from

72574. ** Convert the text beginning at *pz into an integer and return ** its value. Advance *pz to point to the first character past ** the integer. *** This function used for parameters to merge= and incrmerge= ** commands.

72575. Second PmaReader to compare

72576. The element to be inserted

72577. Next offset in pgidx

72578. First byte past end of available space

72579. ** Update the maximum rowid for an autoincrement calculation. *** This routine should be called when the regRowid register holds a ** new rowid that is about to be inserted. If that new rowid is ** larger than the maximum rowid in the memId memory cell, then the ** memory cell is updated.

72580. Size of buffer pointed to by zHeader

72581. See if it is possible to write these frames into the start of the ** log file, instead of appending to it at pWal->hdr.mxFrame.

72582. Prevent the single-thread code below

72583. Expressions of the form: ** ** expr1 IN (?1) ** expr1 NOT IN (?2) ** ** with exactly one value on the RHS can be simplified to something ** like this: ** ** expr1 == ?1 ** expr1 <> ?2 ** ** But, the RHS of the == or <> is marked with the EP_Generic flag ** so that it may not contribute to the computation of comparison ** affinity or the collating sequence to use for comparison. Otherwise, ** the semantics would be subtly different from IN or NOT IN.

72584. Auxiliary data storage

72585. cast works around VC++ bug

72586. Array of indexes for this table

72587. LIKE range processing counter register (times 2)

72588. ** Set the suggested cache-size value.

72589. **comment:** ** CAPI3REF: Constants Defining Special Destructor Behavior ** ** These are special values for the destructor that is passed in as the ** final argument to routines like [sqlite3_result_blob()]. ^If the destructor ** argument is SQLITE_STATIC, it means that the content pointer is constant ** and will never change. It does not need to be destroyed. ^The ** SQLITE_TRANSIENT value means that the content will likely change in ** the near future and that SQLite should make its own private copy of ** the content before returning. *** The typedef is necessary to work around problems in certain ** C++ compilers.
label: code-design

72590. Data from the btree layer

72591. Index in aiMap[]

72592. Current value of 'term' column

72593. Operation NULL

72594. If this is a scalar select that is part of an expression, then ** store the results in the appropriate memory cell or array of ** memory cells and break out of the scan loop.

72595. Array of characters to make exceptions

72596. **comment:** ** TODO: Make this more efficient!
label: code-design

72597. Write new cursor here

72598. Child page number

72599. ** Clear the Rtree.pNodeBlob object

72600. ** Pseudo code for loop that calls stat_push(): ** ** Rewind csr ** if eof(csr) goto end_of_scan; ** regChng = 0 ** goto chng_addr_0; ** ** next_row: ** regChng = 0 ** if(idx(0) != regPrev(0)) goto chng_addr_0 ** regChng = 1 ** if(idx(1) != regPrev(1)) goto chng_addr_1 ** ... ** regChng = N ** goto chng_addr_N ** chng_addr_0: ** regPrev(0) = idx(0) ** chng_addr_1: ** regPrev(1) = idx(1) ** ... ** ** endDistinctTest: ** regRowid = idx(rowid) ** stat_push(P, regChng, regRowid) ** Next csr ** if !eof(csr) goto next_row; ** ** end_of_scan:

72601. Step 3. The LHS is now known to be non-NULL. Do the binary search ** of the RHS using the LHS as a probe. If found, the result is ** true.

72602. 520

72603. Leave cursor pointing at NEXT or PREV

72604. 2: (Istart_constraints && startEq && !bRev)

72605. CREATE TABLE x(input, token, start, end, position)

72606. Replacement table for iParent

72607. ** The number of times that a ReadFile(), WriteFile(), and DeleteFile() ** will be retried following a locking error - probably caused by ** antivirus software.
Also the initial delay before the first retry. ** The delay increases linearly with each retry.

72608. Fields below are only available in SQLite 3.10.0 and later

72609. 3 -> UNCOMPRESS

72610. 67

72611. The pager to set safety level for

72612. SQLITE_NOMEM

72613. ** 2007 August 14 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the C functions that implement mutexes. ***
This file contains code that is common across all mutex implementations.

72614. ** An abstract type for a pointer to an IO method finder function:

72615. NULL Function Name

72616. Database containing the trigger

72617. ** The implementation of user-defined scalar function fts5_decode().

72618. Field to extract from pVector

72619. Populate the change object. None of the preupdate_old(), ** preupdate_new() or SerializeValue() calls below may fail as all ** required values and encodings have already been cached in memory. ** It is not possible for an OOM to occur in this block.

72620. The view was modified by some other optimization such as ** pushDownWhereTerms()

72621. Write-ahead log used by "journal_mode=wal"

72622. Size of array apRegion

72623. ** Insert a new FuncDef into a FuncDefHash hash table.

72624. ** CAPI3REF: Overload A Function For A Virtual Table ** METHOD: sqlite3 ** ** ^{(Virtual tables can provide alternative implementations of functions ** using the [xFindFunction] method of the [virtual table module]. ** But global versions of those functions ** must exist in order to be overloaded.)}^ ** ** ^{(This API makes sure a global version of a function with a particular ** name and number of parameters exists. If no such function exists ** before this API is called, a new function is created.)}^ ** The implementation ** of the new function always causes an exception to be thrown. So ** the new function is not good for anything by itself. Its only ** purpose is to be a placeholder function that can be overloaded ** by a [virtual table].

72625. Bytes required to hold all column names

72626. sqlite3_bind_int(p->pIdxWriter, 1, pWriter->iSegid);

72627. (4)

72628. One of TRIGGER_BEFORE, TRIGGER_AFTER

72629. ** Given the name of a column of the form X.Y.Z or Y.Z or just Z, look up ** that name in the set of source tables in pSrcList and make the pExpr ** expression node refer back to that source column. The following changes ** are made to pExpr: ** ** pExpr->iDb Set the index in db->aDb[] of the database X ** (even if X is implied). ** pExpr->iTable Set to the cursor number for the table obtained ** from pSrcList. ** pExpr->pTab Points to the Table structure of X.Y (even if ** X and/or Y are implied). ** pExpr->iColumn Set to the column number within the table. ** pExpr->op Set to TK_COLUMN. ** pExpr->pLeft Any expression this points to is deleted ** pExpr->pRight Any expression this points to is deleted. *** The zDb variable is the name of the database (the "X"). This value may be ** NULL meaning that name is of the form Y.Z or Z. Any available database ** can be used. The zTable variable is the name of the table (the "Y"). This ** value can be NULL if zDb is also NULL. If zTable is NULL it ** means that the form of the name is Z and that columns from any table ** can be used. *** If the name cannot be resolved unambiguously, leave an error message ** in pParse and return WRC_Abort. Return WRC_Prune on success.

72630. xVdbeBranch

72631. 38

72632. True if this cursor is at EOF

72633. **comment:** ** This routine ends a transaction. A transaction is usually ended by ** either a COMMIT or a ROLLBACK operation. This routine may be called ** after rollback of a hot-journal, or if an error occurs while opening ** the journal file or writing the very first journal-header of a ** database transaction. *** ** This routine is never called in PAGER_ERROR state. If it is called ** in PAGER_NONE or PAGER_SHARED state and the lock held is less ** exclusive than a RESERVED lock, it is a no-op. *** ** Otherwise, any active savepoints are released. *** ** If the journal file is open, then it is "finalized". Once a journal ** file has been finalized it is not possible to use it to roll back a ** transaction. Nor will it be considered to be a hot-journal by this ** or any other database connection. Exactly how a journal is finalized ** depends on whether or not the pager is running in exclusive mode and ** the current journal-mode (Pager.journalMode value), as follows: *** ** journalMode==MEMORY ** Journal file descriptor is simply closed. This destroys an ** in-memory journal. *** ** journalMode==TRUNCATE ** Journal file is truncated to zero bytes in size. *** ** journalMode==PERSIST ** The first 28 bytes of the journal file are zeroed. This invalidates ** the first journal header in the file, and hence the entire journal ** file. An invalid journal file cannot be rolled back. *** ** journalMode==DELETE ** The journal file is closed and deleted using sqlite3OsDelete(). *** ** If the pager is running in exclusive mode, this method of finalizing ** the journal file is never used. Instead, if the journalMode is ** DELETE and the pager is in exclusive mode, the method described under ** journalMode==PERSIST is used instead. *** ** After the journal is finalized, the pager moves to PAGER_READER state. ** If running in non-exclusive rollback mode, the lock on the file is ** downgraded to a SHARED_LOCK. *** ** SQLITE_OK is returned if no error occurs. If an error occurs during ** any of the IO operations to finalize the journal file or unlock the ** database then the IO error code is returned to the user. If the ** operation to finalize the journal file fails, then the code still ** tries to unlock the database file if not in exclusive mode. If the ** unlock operation fails as well, then the first error code related ** to the first error encountered (the journal finalization one) is ** returned.

label: code-design

72634. cmd ::= DROP VIEW ifexists fullname

72635. Number of parameters

72636. ** Size of the column cache

72637. ** If it is currently unknown whether or not the FTS table has an %_stat ** table (if p->bHasStat==2), attempt to determine this (set p->bHasStat ** to 0 or 1). Return SQLITE_OK if successful, or an SQLite error code ** if an error occurs.

72638. Call to try make a page clean

72639. ColCache valid when aColCache[].iLevel<=iCacheLevel

72640. IF

72641. Error detected by xRecordCompare (CORRUPT or NOMEM)

72642. True to grab the global stats

72643. Pointer to buffer containing block data

72644. Current size of node in bytes

72645. ** Merge two lists of RowSetEntry objects. Remove duplicates. ** ** The input lists are connected via pRight pointers and are ** assumed to each already be in sorted order.

72646. The phrase object to trim the doclist of

72647. Address of "loops" counter

72648. ** Round up a request size to the next valid allocation size.

72649. ** A WhereTerm with eOperator==WO_OR has its u.pOrInfo pointer set to ** a dynamically allocated instance of the following structure.

72650. Loop through the doclists in the aaOutput[] array. Merge them all ** into a single doclist.

72651. synopsis: r[P2@P3+1]=[P1@P3+1]

72652. Size of new segment at iAbsLevel

72653. Number of tokens seen so far

72654. nPage

72655. ** This routine generates Vdbe code to compute the content of a WITH RECURSIVE ** query of the form: ** ** <recursive-table> AS (<setup-query> UNION [ALL] <recursive-query>) ** _____/_____/** p->pPrior p ** ** ** There is exactly one reference to the recursive-table in the FROM clause ** of recursive-query, marked with the SrcList->a[].fg.isRecursive flag. ** ** The setup-query runs once to generate an initial set of rows that go ** into a Queue table. Rows are extracted from the Queue table one by ** one. Each row extracted from Queue is output to pDest. Then the single ** extracted row (now in the iCurrent table) becomes the content of the ** recursive-table for a recursive-query run. The output of the recursive-query ** is added back into the Queue table. Then another row is extracted from Queue ** and the iteration continues until the Queue table is empty. ** ** If the compound query operator is UNION then no duplicate rows are ever ** inserted into the Queue table. The iDistinct table keeps a copy of all rows ** that have ever been inserted into Queue and causes duplicates to be ** discarded. If the operator is UNION ALL, then duplicates are allowed. ** ** If the query has an ORDER BY, then entries in the Queue table are kept in ** ORDER BY order and the first entry is extracted for each cycle. Without ** an ORDER BY, the Queue table is just a FIFO. ** ** If a LIMIT clause is provided, then the iteration stops after LIMIT rows ** have been output to pDest. A LIMIT of zero means to output no rows and a ** negative LIMIT means to output all rows. If there is also an OFFSET clause ** with a positive value, then the first OFFSET outputs are discarded rather ** than being sent to pDest. The LIMIT count does not begin until after OFFSET ** rows have been skipped.

72656. Most recent error code (SQLITE_*)

72657. left

72658. PCache object

72659. Label for the end of the insertion loop

72660. wqlist

72661. ** Count the number of references to columns.

72662. Database to write to

72663. sqlite_stat4.nLt

72664. Ascii for "J"

72665. ** This is a convenience routine that makes sure that all thread-specific ** data for this thread has been deallocated. ** ** SQLite no longer uses thread-specific data so this routine is now a ** no-op. It is retained for historical compatibility.

72666. Firstest rowid of other than aFirst[1]

72667. ** The following type and function are used to iterate through all opcodes ** in a Vdbe main program and each of the sub-programs (triggers) it may ** invoke directly or indirectly. It should be used as follows: ** ** Op *pOp; ** VdbeOpIter sIter; ** ** memset(&sIter, 0, sizeof(sIter)); ** sIter.v = v; // v is of type Vdbe* ** while((pOp = opIterNext(&sIter))){ ** // Do something with pOp ** } ** sqlite3DbFree(v->db, sIter.apSub); **

72668. Shorthand for pScan->pWC

72669. ** This function is the implementation of both the xConnect and xCreate ** methods of the r-tree virtual table. ** ** argv[0] -> module name ** argv[1] -> database name ** argv[2] -> table name ** argv[...] -> column names...

72670. Reg array containing OLD.* and NEW.* values

72671. Opcode: Delete P1 P2 P3 P4 P5 ** ** Delete the record at which the P1 cursor is currently pointing. ** ** If the OPFLAG_SAVEPOSITION bit of the P5 parameter is set, then ** the cursor will be left pointing at either the next or the previous ** record in the table. If it is left pointing at the next record, then ** the next Next instruction will be a no-op. As a result, in this case ** it is ok to delete a record from within a Next loop. If ** OPFLAG_SAVEPOSITION bit of P5 is clear, then the cursor will be ** left in an undefined state. ** ** If the OPFLAG_AUXDELETE bit is set on P5, that indicates that this ** delete one of several associated with deleting a table row and all its ** associated index entries. Exactly one of those deletes is the "primary" ** delete. The others are all on OPFLAG_FORDELETE cursors or else are ** marked with the AUXDELETE flag. ** ** If the OPFLAG_NCHANGE flag of P2 (NB: P2 not P5) is set, then the row ** change count is incremented (otherwise not). ** ** P1 must not be pseudo-table. It has to be a real table with ** multiple rows. ** ** If P4 is not NULL then it points to a Table object. In this case either ** the update or pre-update hook, or both, may be invoked. The P1 cursor must ** have been positioned using OP_NotFound prior to invoking this opcode in ** this case. Specifically, if one is configured, the pre-update hook is ** invoked if P4 is not NULL. The update-hook is invoked if one is configured, ** P4 is not NULL, and the OPFLAG_NCHANGE flag is set in P2. ** ** If the OPFLAG_ISUPDATE flag is set in P2, then P3 contains the address ** of the memory cell that contains the value that the rowid of the row will ** be set to by the update.

72672. Offset in pointer map page

72673. ** 2002 February 23 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the C-language implementations for many of the SQL ** functions of SQLite. (Some function, and in particular the date and ** time functions, are implemented separately.)

72674. ** current_time() ** ** This function returns the same value as time('now').

72675. Number of references to this KeyInfo object
72676. P4 is a pointer to an sqlite3_vtab structure
72677. Release the shared CKPT lock obtained above.
72678. ** A constraint has failed while inserting a row into an rtree table. ** Assuming no OOM error occurs, this function sets the error message ** (at pRtree->base.zErrMsg) to an appropriate value and returns ** SQLITE_CONSTRAINT. ** ** Parameter iCol is the index of the leftmost column involved in the ** constraint failure. If it is 0, then the constraint that failed is ** the unique constraint on the id column. Otherwise, it is the rtree ** (c1<=c2) constraint on columns iCol and iCol+1 that has failed. ** ** If an OOM occurs, SQLITE_NOMEM is returned instead of SQLITE_CONSTRAINT.
72679. ** Some systems have strcmp(). Others have strcasecmp(). Because ** there is no consistency, we will define our own. ** ** IMPLEMENTATION-OF: R-30243-02494 The sqlite3_strcmp() and ** sqlite3_strnicmp() APIs allow applications and extensions to compare ** the contents of two buffers containing UTF-8 strings in a ** case-independent fashion, using the same definition of "case ** independence" that SQLite uses internally when comparing identifiers.
72680. #include "fts5Int.h"
72681. 285
72682. ***** End of attach.c *****
72683. Append formatted text (not to exceed N bytes) to the JsonString.
72684. True if resolving columns of CREATE INDEX
72685. cmd ::= PRAGMA nm dbnm EQ nmnum
72686. Sum of the nEq values
72687. ** Open a journal file. ** ** The behaviour of the journal file depends on the value of parameter ** nSpill. If nSpill is 0, then the journal file is always create and ** accessed using the underlying VFS. If nSpill is less than zero, then ** all content is always stored in main-memory. Finally, if nSpill is a ** positive value, then the journal file is initially created in-memory ** but may be flushed to disk later on. In this case the journal file is ** flushed to disk either when it grows larger than nSpill bytes in size, ** or when sqlite3JournalCreate() is called.
72688. Remove cells from the start and end of the page
72689. Opcode that opens the sorter
72690. Second part of index name. May be NULL
72691. OR using multiple indices
72692. First token of current snippet
72693. ***** Include fts3_tokenizer.h in the middle of fts3Int.h *****
72694. Update the aBest[] array.
72695. List of VALUES() to be inserted
72696. 204
72697. GetMeta() and UpdateMeta() cannot fail in this context because ** we already have page 1 loaded into cache and marked dirty.
72698. Opcode: SorterOpen P1 P2 P3 P4 * ** This opcode works like OP_OpenEphemeral except that it opens ** a transient index that is specifically designed to sort large ** tables using an external merge-sort algorithm. ** ** If argument P3 is non-zero, then it indicates that the sorter may ** assume that a stable sort considering the first P3 fields of each ** key is sufficient to produce the required results.
72699. How many levels of subquery
72700. ** Formulate a statement to DELETE a row from database db. Assuming a table ** structure like this: ** ** CREATE TABLE x(a, b, c, d, PRIMARY KEY(a, c)); ** ** The DELETE statement looks like this: ** ** DELETE FROM x WHERE a = :1 AND c = :3 AND (:5 OR b IS :2 AND d IS :4) ** ** Variable :5 (nCol+1) is a boolean. It should be set to 0 if we require ** matching b and d values, or 1 otherwise. The second case comes up if the ** conflict handler is invoked with NOTFOUND and returns CHANGESSET_REPLACE. ** ** If successful, SQLITE_OK is returned and SessionApplyCtx.pDelete is left ** pointing to the prepared version of the SQL statement.
72701. Authorize the subquery
72702. List of user defined columns
72703. 201
72704. resolvetype ::= IGNORE
72705. Remove the entry in the parent cell.
72706. **comment:** ** This function is used to add VdbeComment() annotations to a VDBE ** program. It is not used in production code, only for debugging.
label: documentation
72707. if pLimit is null, pOffset will always be null as well.
72708. RTREE_GT or RTREE_GE, or fallback of RTREE_EQ
72709. Precompiled statements used by the implementation. Each of these ** statements is run and reset within a single virtual table API call.
72710. ** An instance of the following structure stores a database schema. ** ** Most Schema objects are associated with a Btree. The exception is ** the Schema for the TEMP databases (sqlite3.aDb[1]) which is free-standing. ** In shared cache mode, a single Schema object can be shared by multiple ** Btrees that refer to the same underlying BtShared object. ** ** Schema objects are automatically deallocated when the last Btree that ** references them is destroyed. The TEMP Schema is manually freed by ** sqlite3_close(). ** ** A thread must be holding a mutex on the corresponding Btree in order ** to access Schema content. This implies that the thread must also be ** holding a mutex on the sqlite3 connection pointer that owns the Btree. ** For a TEMP Schema, only the connection mutex is required.
72711. **comment:** content=xxx option, or NULL
label: code-design
72712. ** This routine is invoked once per CTE by the parser while parsing a ** WITH clause.
72713. The column of the index to be loaded
72714. File descriptor to close
72715. Invoke the callback function if required
72716. ** The first character of the string pointed to by argument z is guaranteed ** to be an open-quote character (see function fts5_isopenquote()). ** ** This function searches for the corresponding close-quote character within ** the string and, if found, dequotes the string in place and adds a new ** nul-terminator byte. ** ** If the close-quote is found, the value returned is the byte offset of ** the character immediately following it. Or, if the close-quote is not ** found, -1 is returned. If -1 is returned, the buffer is left in an ** undefined state.
72717. ** Parse a "special" CREATE VIRTUAL TABLE directive and update ** configuration object pConfig as appropriate. ** ** If successful, object pConfig is updated and SQLITE_OK returned. If ** an error occurs, an SQLite error code is returned and an error message ** may be left in *pzErr. It is the responsibility of the caller to ** eventually free any such error message using sqlite3_free().
72718. Buffer containing root node
72719. Number of entries in this segment
72720. Sub-program to execute
72721. ** For testing purposes, we sometimes want to preserve the state of ** PRNG and restore the PRNG to its saved state at a later time, or ** to reset the PRNG to its initial state. These routines accomplish ** those tasks. ** ** The sqlite3_test_control() interface calls these routines to ** control the PRNG.
72722. Allocate memory for the Pager structure, PCache object, the ** three file descriptors, the database file name and the journal ** file name. The layout in memory is as follows: ** ** Pager object (sizeof(Pager) bytes) ** PCache object (sqlite3PcacheSize() bytes) ** Database file handle (pVfs->szOsFile bytes) ** Sub-journal file handle (journalFileSize bytes) ** Main journal file handle (journalFileSize bytes) ** Database file name (nPathname+1 bytes) ** Journal file name (nPathname+8+1 bytes)
72723. Schema of the expression
72724. The VFS containing this xAccess method
72725. OUT: Array of indexes for this table
72726. Allocate a new WhereLoop to add to the end of the list
72727. ** Set *ppStmt to a statement handle that may be used to iterate through ** all rows in the %_segdir table, from oldest to newest. If successful, ** return SQLITE_OK. If an error occurs while preparing the statement, ** return an SQLite error code. ** ** There is only ever one instance of this SQL statement compiled for ** each FTS3 table. ** ** The statement returns the following columns from the %_segdir table: ** ** 0: idx ** 1: start_block ** 2: leaves_end_block ** 3: end_block ** 4: root
72728. ** CAPI3REF: Recover snapshots from a wal file ** EXPERIMENTAL ** ** If all connections disconnect from a database file but do not perform ** a checkpoint, the existing wal file is opened along with the database ** file the next time the database is opened. At this point it is only ** possible to successfully

call sqlite3_snapshot_open() to open the most ** recent snapshot of the database (the one at the head of the wal file), ** even though the wal file may contain other valid snapshots for which ** clients have sqlite3_snapshot handles. *** This function attempts to scan the wal file associated with database zDb ** of database handle db and make all valid snapshots available to ** sqlite3_snapshot_open(). It is an error if there is already a read ** transaction open on the database, or if the database is not a wal mode ** database. *** SQLITE_OK is returned if successful, or an SQLite error code otherwise.

72729. Finalize the journal file.

72730. The columns to change in the UPDATE statement

72731. ** Set the pIdxInfo->estimatedRows variable to nRow. Unless this ** extension is currently being used by a version of SQLite too old to ** support estimatedRows. In that case this function is a no-op.

72732. pNext

72733. ***** Continuing where we left off in main.c *****

72734. If no match is found, search the built-in functions. *** If the SQLITE_PreferBuiltin flag is set, then search the built-in ** functions even if a prior app-defined function was found. And give ** priority to built-in functions. *** Except, if createFlag is true, that means that we are trying to ** install a new function. Whatever FuncDef structure is returned it will ** have fields overwritten with new information appropriate for the ** new function. But the FuncDefs for built-in functions are read-only. ** So we must not search for built-ins when creating a new function.

72735. Typecast byte array

72736. Significant digits after the decimal point

72737. Mallored memory used by some conversion

72738. ** Close all cursors in the current frame.

72739. 46

72740. The soft heap limit

72741. Number of virtual tables to lock

72742. Scan visits at most 1 row

72743. Case 3.

72744. OUT: Mapped memory

72745. Memory usable by this allocator

72746. The Z in X.Y.Z cannot be NULL

72747. List of table triggers, if required

72748. ** This macro is used to calculate hash key values for data structures. In ** order to use this macro, the entire data structure must be represented ** as a series of unsigned integers. In order to calculate a hash-key value ** for a data structure represented as three such integers, the macro may ** then be used as follows: ***
int hash_key_value; ** hash_key_value = HASH_APPEND(0, <value 1>); ** hash_key_value = HASH_APPEND(hash_key_value, <value 2>); **
hash_key_value = HASH_APPEND(hash_key_value, <value 3>); *** In practice, the data structures this macro is used for are the primary ** key values of modified rows.

72749. Phrase expression

72750. ** This function may only be called while the iterator is pointing to an ** SQLITE_UPDATE or SQLITE_INSERT change (see sqlite3changeset_op()). ** Otherwise, SQLITE_MISUSE is returned. *** It sets *ppValue to point to an sqlite3_value structure containing the ** iVal'th value in the new.* record. Or, if that particular value is not ** included in the record (because the change is an UPDATE and the field ** was not modified), set *ppValue to NULL. *** If value iVal is out-of-range, SQLITE_RANGE is returned and *ppValue is ** not modified. Otherwise, SQLITE_OK.

72751. ** This routine installs a default busy handler that waits for the ** specified number of milliseconds before returning 0.

72752. ***** Begin file fts5.h *****

72753. The WhereLoop object

72754. 1420

72755. Flags that may be passed as the third argument to xTokenize()

72756. The SRT_ operation to apply to prior selects

72757. LOOKUP_DOCSIZE

72758. void* int int

72759. ** Add MEM_Str to the set of representations for the given Mem. Numbers ** are converted using sqlite3_snprintf(). Converting a BLOB to a string ** is a no-op. *** Existing representations MEM_Int and MEM_Real are invalidated if ** bForce is true but are retained if bForce is false. *** A MEM_Null value will never be passed to this function. This function is ** used for converting values to text for returning to the user (i.e. via ** sqlite3_value_text()), or for ensuring that values to be used as btree ** keys are strings. In the former case a NULL pointer is returned the ** user and the latter is an internal programming error.

72760. ** In this version of BtreeMoveto, pKey is a packed index record ** such as is generated by the OP_MakeRecord opcode. Unpack the ** record and then call BtreeMovetoUnpacked() to do the work.

72761. Operands are not equal

72762. **comment:** ** CAPI3REF: Advance A Changeset Iterator ** ** This function may only be used with iterators created by function ** [sqlite3changeset_start()]. If it is called on an iterator passed to ** a conflict-handler callback by [sqlite3changeset_apply()], SQLITE_MISUSE ** is returned and the call has no effect. *** Immediately after an iterator is created by sqlite3changeset_start(), it ** does not point to any change in the changeset. Assuming the changeset ** is not empty, the first call to this function advances the iterator to ** point to the first change in the changeset. Each subsequent call advances ** the iterator to point to the next change in the changeset (if any). If ** no error occurs and the iterator points to a valid change after a call ** to sqlite3changeset_next() has advanced it, SQLITE_ROW is returned. ** Otherwise, if all changes in the changeset have already been visited, ** SQLITE_DONE is returned. *** If an error occurs, an SQLite error code is returned. Possible error ** codes include SQLITE_CORRUPT (if the changeset buffer is corrupt) or ** SQLITE_NOMEM.

label: code-design

72763. Suppress the first OFFSET entries if there is an OFFSET clause

72764. True to enable full mutexing

72765. Left hand child expression

72766. Transient list of dirty sorted by pgno

72767. OUT: End-of-file flag

72768. 12-14

72769. ** Give a listing of the program in the virtual machine. *** The interface is the same as sqlite3VdbeExec(). But instead of ** running the code, it invokes the callback once for each instruction. ** This feature is used to implement "EXPLAIN". *** When p->explain==1, each instruction is listed. When ** p->explain==2, only OP_Explain instructions are listed and these ** are shown in a different format. p->explain==2 is used to implement ** EXPLAIN QUERY PLAN. *** When p->explain==1, first the main program is listed, then each of ** the trigger subprograms are listed one by one.

72770. Tell the wal layer that an EXCLUSIVE lock has been obtained (or released) ** by the pager layer on the database file.

72771. Number of segments to merge

72772. Vdbe being configured

72773. True to use prefix-indexes

72774. Create a list of user columns for the content table

72775. ***** Continuing where we left off in loadext.c *****

72776. Minimum docid to return

72777. ** Return true if pTab is a contentless table.

72778. Next input in pColset

72779. ** Pragma virtual table module xRowid method.

72780. Packed key if the btree is an index

72781. The index to be used for a inequality constraint

72782. Append the position-list data to the output

72783. **comment:** ** If the path name starts with a forward slash or a backslash, it is either ** a legal UNC name, a volume relative path, or an absolute path name in the ** "Unix" format on Windows. There is no easy way to differentiate between ** the final two cases; therefore, we return the safer return value of TRUE ** so that callers of this function will simply use it verbatim.

label: code-design

72784. **comment:** ** Change the comment on the most recently coded instruction. Or ** insert a No-op and add the comment to that new instruction. This ** makes the code easier to read during debugging. None of this happens ** in a production build.

label: code-design

72785. ** Insert a record into the %_segments table.

72786. If there is already a lock of this type or more restrictive on the ** unixFile, do nothing. Don't use the afp_end_lock: exit path, as ** unixEnterMutex() hasn't been called yet.

72787. rowid >= ?

72788. Allowed equality operators

72789. ** Invoke the xRollback method of all virtual tables in the ** sqlite3.aVTrans array. Then clear the array itself.

72790. IMPLEMENTATION-OF: R-46656-45156 The sqlite3_version[] string constant ** contains the text of SQLITE_VERSION macro.

72791. Maximum size of regions mapped by sorter

72792. ** Register a VFS with the system. It is harmless to register the same ** VFS multiple times. The new VFS becomes the default if makeDflt is ** true.

72793. Valid operators for constraints

72794. ***** Begin file whereInt.h *****

72795. the GROUP BY clause

72796. The pragma statement to run

72797. Matchinfo context

72798. Register holding block-output return address

72799. Either this is the cheapest token in the entire query, or it is ** part of a multi-token phrase. Either way, the entire doclist will ** (eventually) be loaded into memory. It may as well be now.

72800. EVIDENCE-OF: R-26900-09176 A value of 13 (0x0d) means the page is a ** leaf table b-tree page.

72801. idlist

72802. Token for <value>, or NULL

72803. Not required due to the previous to assert() statements

72804. Number of leaf pages flushed

72805. ** Attempt to set the page size of the destination to match the page size ** of the source.

72806. Pointer to data to append to buffer

72807. Table to append

72808. ** The implementation of user-defined scalar functions fts5_expr() (bTcl==0) ** and fts5_expr_tcl() (bTcl!=0).

72809. Current offset within aNode[]

72810. Grab the write lock on the log file. If successful, upgrade to ** PAGER_RESERVED state. Otherwise, return an error code to the caller. ** The busy-handler is not invoked if another connection already ** holds the write-lock. If possible, the upper layer will call it.

72811. ** The threadid macro resolves to the thread-id or to 0. Used for ** testing and debugging only.

72812. zNum is greater than 9223372036854775808 so it overflows

72813. get memory map allocation granularity

72814. ** An instance of this structure represents a set of one or more CTEs ** (common table expressions) created by a single WITH clause.

72815. Name of the transformation

72816. Current number of samples

72817. Offset to beginning of page header

72818. IMPLEMENTATION-OF: R-34391-24921 The sqlite3_release_memory() routine ** is a no-op returning zero if SQLite is not compiled with ** SQLITE_ENABLE_MEMORY_MANAGEMENT.

72819. The SELECT statement containing the clause

72820. Pointer to right-sibling

72821. 9

72822. **comment:** ** If *pArg is initially negative then this is a query. Set *pArg to ** 1 or 0 depending on whether or not bit mask of pFile->ctrlFlags is set. *** If *pArg is 0 or 1, then clear or set the mask bit of pFile->ctrlFlags.

label: code-design

72823. ** This function is called to generate code executed when a row is deleted ** from the parent table of foreign key constraint pFKey and, if pFKey is ** deferred, when a row is inserted into the same table. When generating ** code for an SQL UPDATE operation, this function may be called twice - ** once to "delete" the old row and once to "insert" the new row. *** ** Parameter nIncr is passed -1 when inserting a row (as this may decrease ** the number of FK violations in the db) or +1 when deleting one (as this ** may increase the number of FK constraint problems). *** ** The code generated by this function scans through the rows in the child ** table that correspond to the parent table row being deleted or inserted. ** For each child row found, one of the following actions is taken: *** ** Operation | FK type | Action taken ** ----- ** DELETE immediate Increment the "immediate constraint counter". ** Or, if the ON (UPDATE|DELETE) action is RESTRICT, ** throw a "FOREIGN KEY constraint failed" exception. *** ** INSERT immediate Decrement the "immediate constraint counter". ** DELETE deferred Increment the "deferred constraint counter". ** Or, if the ON (UPDATE|DELETE) action is RESTRICT, ** throw a "FOREIGN KEY constraint failed" exception. *** ** INSERT deferred Decrement the "deferred constraint counter". ** *** These operations are identified in the comment at the top of this file ** (fkey.c) as "I.2" and "D.2".

72824. xShmLock

72825. ** Create an RBU VFS named zName that accesses the underlying file-system ** via existing VFS zParent. Or, if the zParent parameter is passed NULL, ** then the new RBU VFS uses the default system VFS to access the file-system. ** The new object is registered as a non-default VFS with SQLite before ** returning. ** ** Part of the RBU implementation uses a custom VFS object. Usually, this ** object is created and deleted automatically by RBU. *** ** The exception is for applications that also use zipvfs. In this case, ** the custom VFS must be explicitly created by the user before the RBU ** handle is opened. The RBU VFS should be installed so that the zipvfs ** VFS uses the RBU VFS, which in turn uses any other VFS layers in use ** (for example multiplexor) to access the file-system. For example, ** to assemble an RBU enabled VFS stack that uses both zipvfs and ** multiplexor (error checking omitted): *** // Create a VFS named "multiplex" (not the default). ** sqlite3_multiplex_initialize(0, 0); *** // Create an rbu VFS named "rbu" that uses multiplexor. If the ** // second argument were replaced with NULL, the "rbu" VFS would ** // access the file-system via the system default VFS, bypassing the ** // multiplexor. ** sqlite3rbu_create_vfs("rbu", "multiplex"); *** // Create a zipvfs VFS named "zipvfs" that uses rbu. ** zipvfs_create_vfs_v3("zipvfs", "rbu", 0, xCompressorAlgorithmDetector); *** // Make zipvfs the default VFS. ** sqlite3_vfs_register(sqlite3_vfs_find("zipvfs"), 1); *** Because the default VFS created above includes a RBU functionality, it ** may be used by RBU clients. Attempting to use RBU with a zipvfs VFS stack ** that does not include the RBU layer results in an error. ** ** The overhead of adding the "rbu" VFS to the system is negligible for ** non-RBU users. There is no harm in an application accessing the ** file-system via "rbu" all the time, even if it only uses RBU functionality ** occasionally.

72826. The binary record

72827. ** Rtree virtual table module xNext method.

72828. Type of token

72829. Size of buffer zPrev in bytes

72830. Name of the column.

72831. In a join with a USING clause, omit columns in the ** using clause from the table on the right.

72832. ** Convert the given register into a string if it isn't one ** already. Return non-zero if a malloc() fails.

72833. If the database supports auto-vacuum, make sure no tables contain ** references to pointer-map pages.

72834. The cell index from which to extract the ID

72835. ** The xRename method for rtree module virtual tables.

72836. Left hand side of comparison

72837. Flags

72838. ** Return the length (in bytes) of the token that begins at z[0]. ** Store the token type in *tokenType before returning.

72839. Verify property (2)

72840. Opcode: FkCounter P1 P2 * * * * Synopsis: fkctr[P1]+=P2 ** ** Increment a "constraint counter" by P2 (P2 may be negative or positive). ** If P1 is non-zero, the database constraint counter is incremented ** (deferred foreign key constraints). Otherwise, if P1 is zero, the ** statement counter is incremented (immediate foreign key constraints).

72841. The entire doclist will not fit on this leaf. The following ** loop iterates through the poslists that make up the current ** doclist.
72842. Substitute function - never called
72843. Expression to pass to authorization callback
72844. The database connection that owns this statement
72845. Force xDisconnect calls on all virtual tables
72846. Result code from this function
72847. **** Begin AFP Locking
***** AFP is the Apple Filing Protocol. AFP is a network filesystem found ** on Apple Macintosh computers - both OS9 and OSX. *** Third-party implementations of AFP are available. But this code here ** only works on OSX.
72848. If there is a WAL file in the file-system, open this database in WAL ** mode. Otherwise, the following function call is a no-op.
72849. Bind the rbu_rowid value to column _rowid_
72850. Apply locks to this open shared-memory segment
72851. Remove terms smaller than this
72852. ** Object for iterating through a single segment, visiting each term/rowid ** pair in the segment. *** pSeg: ** The segment to iterate through. *** iLeafPgno:
** Current leaf page number within segment. *** iLeafOffset: ** Byte offset within the current leaf that is the first byte of the ** position list data (one byte passed the position-list size field). ** rowid field of the current entry. Usually this is the size field of the ** position list data. The exception is if the rowid for the current entry ** is the last thing on the leaf page. *** pLeaf: ** Buffer containing current leaf page data. Set to NULL at EOF. *** iTermLeafPgno, iTermLeafOffset: ** Leaf page number containing the last term read from the segment. And ** the offset immediately following the term data. *** flags: ** Mask of FTS5_SEGITER_XXX values. Interpreted as follows: *** FTS5_SEGITER_ONETERM: ** If set, set the iterator to point to EOF after the current doclist ** has been exhausted. Do not proceed to the next term in the segment. *** FTS5_SEGITER_REVERSE: ** This flag is only ever set if FTS5_SEGITER_ONETERM is also set. If ** it is set, iterate through rowid in descending order instead of the ** default ascending order. *** iRowidOffset/nRowidOffset/aRowidOffset: ** These are used if the FTS5_SEGITER_REVERSE flag is set. *** For each rowid on the page corresponding to the current term, the ** corresponding aRowidOffset[] entry is set to the byte offset of the ** start of the "position-list-size" field within the page. *** iTermIdx: ** Index of current term on iTermLeafPgno.
72853. Primary error code from p->rc
72854. pPager->nRef = 0;
72855. Tiebreaker hash
72856. SQLITE_SYSTEM_MALLOC
72857. Terms that may be omitted
72858. Release memory from the SQLITE_CONFIG_SCRATCH allocation
72859. True if the insert is likely to be an append
72860. Append the varints
72861. ** If zNum represents an integer that will fit in 32-bits, then set ** *pValue to that integer and return true. Otherwise return false. *** This routine accepts both decimal and hexadecimal notation for integers. *** Any non-numeric characters that follow zNum are ignored. ** This is different from sqlite3Atoi64() which requires the ** input number to be zero-terminated.
72862. ** Parse context structure pFrom has just been used to create a sub-vdbe ** (trigger program). If an error has occurred, transfer error information ** from pFrom to pTo.
72863. ** Allocate nByte bytes of memory using sqlite3_malloc(). If successful, ** zero the memory before returning a pointer to it. If unsuccessful, ** return NULL.
72864. ** Decode a blob of varints into N integers
72865. Current rowid
72866. ** Close a blob handle that was previously created using ** sqlite3_blob_open().
72867. ***** End of pager.h *****
72868. Current token position
72869. Index that owns this iterator
72870. **comment:** Read the master journal name from the journal, if it is present. ** If a master journal file name is specified, but the file is not ** present on disk, then the journal is not hot and does not need to be ** played back. *** TODO: Technically the following is an error because it assumes that ** buffer Pager.pTmpSpace is (mxPathname+1) bytes or larger. i.e. that ** (pPager->pageSize >= pPager->pVfs->mxPathname+1). Using os_unix.c, ** mxPathname is 512, which is the same as the minimum allowable value ** for pageSize.
label: code-design
72871. ** Return a pointer to the right-most SELECT statement in a compound.
72872. right
72873. Discard the results. This is used for SELECT statements inside ** the body of a TRIGGER. The purpose of such selects is to call ** user-defined functions that have side effects. We do not care ** about the actual results of the select.
72874. The following function deletes the "minor type" or semantic value ** associated with a symbol. The symbol can be either a terminal ** or nonterminal. "yymajor" is the symbol code, and "yypminor" is ** a pointer to the value to be deleted. The code used to do the ** deletions is derived from the %destructor and/or %token_destructor ** directives of the input grammar.
72875. ** If SQLITE_OMIT_FLOATING_POINT is defined, then none of the floating point ** conversions will work.
72876. 1:save off p0<<21 | p1<<14 | p2<<7 | p3 (masked)
72877. ** CAPI3REF: Retrieving Statement SQL ** METHOD: sqlite3_stmt ** ** ^The sqlite3_sql(P) interface returns a pointer to a copy of the UTF-8 ** SQL text used to create [prepared statement] P if P was ** created by [sqlite3_prepare_v2()], [sqlite3_prepare_v3()], ** [sqlite3_prepare16_v2()], or [sqlite3_prepare16_v3()]. ** ^The sqlite3_expanded_sql(P) interface returns a pointer to a UTF-8 ** string containing the SQL text of prepared statement P with ** [bound parameters] expanded. *** ^For example, if a prepared statement is created using the SQL ** text "SELECT \$abc,xyz" and if parameter \$abc is bound to integer 2345 ** and parameter :xyz is unbound, then sqlite3_sql() will return ** the original string, "SELECT \$abc,xyz" but sqlite3_expanded_sql() ** will return "SELECT 2345,NULL".\n\n^The sqlite3_expanded_sql() interface returns NULL if insufficient memory ** is available to hold the result, or if the result would exceed the ** the maximum string length determined by the [SQLITE_LIMIT_LENGTH]. *** ^The [SQLITE_TRACE_SIZE_LIMIT] compile-time option limits the size of ** bound parameter expansions. ^The [SQLITE_OMIT_TRACE] compile-time ** option causes sqlite3_expanded_sql() to always return NULL. *** ^The string returned by sqlite3_sql(P) is managed by SQLite and is ** automatically freed when the prepared statement is finalized. *** ^The string returned by sqlite3_expanded_sql(P), on the other hand, ** is obtained from [sqlite3_malloc()] and must be free by the application ** by passing it to [sqlite3_free()].
72878. Index of new segment
72879. ** Return a 64-bit integer value for a query parameter.
72880. cmd ::= createkw temp VIEW ifnotexists nm dbnm eidlist_opt AS select
72881. EV: R-46199-30249
72882. Search for the end of the position list within the current page.
72883. table_options ::= WITHOUT nm
72884. ***** End of pragma.c *****
72885. ***** Begin file fts3_icu.c *****
72886. ** Estimate the average size of a row for an index.
72887. Pointer to current poslist
72888. Attempt to locate an element of the hash table pH with a key ** that matches pKey. Return the data for this element if it is ** found, or NULL if there is no match.
72889. Fall through to the next case, OP_String
72890. Lookup by rowid on %_content table
72891. Number of bytes in buffer pB
72892. Populate the cell.aCoord[] array. The first coordinate is azData[3]. *** NB: nData can only be less than nDim*2+3 if the rtree is mis-declared ** with "column" that are interpreted as table constraints. ** Example: CREATE VIRTUAL TABLE bad USING rtree(x,y,CHECK(y>5)); ** This problem was discovered after years of use, so we silently ignore ** these kinds of misdeclared tables to avoid breaking any legacy.
72893. Database page size in bytes. 1==64K
72894. Vdbe to generate code within

72895. ** Construct a new expression node for a function with multiple ** arguments.
 72896. ** Details: *** The %_data table managed by this module, *** CREATE TABLE %_data(id INTEGER PRIMARY KEY, block BLOB); *** , contains the following 5 types of records. See the comments surrounding ** the FTS5_*_ROWID macros below for a description of how %_data rowids are ** assigned to each fo them. *** 1. Structure Records: *** The set of segments that make up an index - the index structure - are ** recorded in a single record within the %_data table. The record consists ** of a single 32-bit configuration cookie value followed by a list of ** SQLite varints. If the FTS table features more than one index (because ** there are one or more prefix indexes), it is guaranteed that all share ** the same cookie value. *** Immediately following the configuration cookie, the record begins with ** three varints: ** + number of levels, ** + total number of segments on all levels, ** + value of write counter. *** Then, for each level from 0 to nMax: ** + number of input segments in ongoing merge. ** + total number of segments in level. ** + for each segment from oldest to newest: ** + segment id (always > 0) ** + first leaf page number (often 1, always greater than 0) ** + final leaf page number *** 2. The Averages Record: *** A single record within the %_data table. The data is a list of varints. ** The first value is the number of rows in the index. Then, for each column ** from left to right, the total number of tokens in the column for all ** rows of the table. *** 3. Segment leaves: *** TERM/DOCLIST FORMAT: *** Most of each segment leaf is taken up by term/doclist data. The ** general format of term/doclist, starting with the first term ** on the leaf page, is: ** varint : size of first term ** blob: first term data ** doclist: first doclist ** zero-or-more { ** varint: number of bytes in common with previous term ** varint: number of bytes of new term data (nNew) ** blob: nNew bytes of new term data ** doclist: next doclist ** } *** doclist format: ** varint: first rowid ** poslist: first poslist ** zero-or-more { ** varint: rowid delta (always > 0) ** poslist: next poslist ** } *** poslist format: ** varint: size of poslist in bytes multiplied by 2, not including ** this field. Plus 1 if this entry carries the "delete" flag. ** collist: collist for column 0 ** zero-or-more { ** 0x01 byte ** varint: column number (I) ** collist: collist for column I ** } *** collist format: ** varint: first offset + 2 ** zero-or-more { ** varint: offset delta + 2 ** } *** PAGE FORMAT *** Each leaf page begins with a 4-byte header containing 2 16-bit ** unsigned integer fields in big-endian format. They are: *** ** The byte offset of the first rowid on the page, if it exists ** and occurs before the first term (otherwise 0). *** ** The byte offset of the start of the page footer. If the page ** footer is 0 bytes in size, then this field is the same as the ** size of the leaf page in bytes. *** ** The page footer consists of a single varint for each term located ** on the page. Each varint is the byte offset of the current term ** within the page, delta-compressed against the previous value. In ** other words, the first varint in the footer is the byte offset of ** the first term, the second is the byte offset of the second less that ** of the first, and so on. *** ** The term/doclist format described above is accurate if the entire ** term/doclist data fits on a single leaf page. If this is not the case, ** the format is changed in two ways: ** + if the first rowid on a page occurs before the first term, it ** is stored as a literal value: ** varint: first rowid ** + the first term on each page is stored in the same way as the ** very first term of the segment: ** varint : size of first term ** blob: first term data *** 5. Segment doclist indexes: *** Doclist indexes are themselves b-trees, however they usually consist of ** a single leaf record only. The format of each doclist index leaf page ** is: *** ** Flags byte. Bits are: ** 0x01: Clear if leaf is also the root page, otherwise set. *** ** Page number of fts index leaf page. As a varint. *** ** First rowid on page indicated by previous field. As a varint. *** ** A list of varints, one for each subsequent termless page. A ** positive delta if the termless page contains at least one rowid, ** or an 0x00 byte otherwise. *** ** Internal doclist index nodes are: *** ** Flags byte. Bits are: ** 0x01: Clear for root page, otherwise set. *** ** Page number of first child page. As a varint. *** ** Copy of first rowid on page indicated by previous field. As a varint. *** ** A list of delta-encoded varints - the first rowid on each subsequent ** child page. ***
 72897. ** json_group_obj(NAME,VALUE)*** Return a JSON object composed of all names and values in the aggregate.
 72898. True if "PRAGMA synchronous=N" has been run
 72899. (304) sclp ::= selcollist COMMA
 72900. transform the db path to a unique cache name
 72901. Attempt the UPDATE. In the case of a NOTFOUND or DATA conflict, ** the result will be SQLITE_OK with 0 rows modified.
 72902. conch file header length
 72903. The file descriptor to be filled in
 72904. Next offset to write to
 72905. This case when there exist aggregate functions or a GROUP BY clause ** or both
 72906. **comment:** ** The sqlite3_mutex_alloc() routine allocates a new ** mutex and returns a pointer to it. If it returns NULL ** that means that a mutex could not be allocated. SQLite ** will unwind its stack and return an error. The argument ** to sqlite3_mutex_alloc() is one of these integer constants: *** ** SQLITE_MUTEX_FAST ** SQLITE_MUTEX_RECURSIVE ** SQLITE_MUTEX_STATIC_MASTER ** SQLITE_MUTEX_STATIC_MEM ** SQLITE_MUTEX_STATIC_OPEN ** SQLITE_MUTEX_STATIC_PRNG ** SQLITE_MUTEX_STATIC_LRU ** SQLITE_MUTEX_STATIC_PMEM ** SQLITE_MUTEX_STATIC_APP1 ** SQLITE_MUTEX_STATIC_APP2 ** SQLITE_MUTEX_STATIC_APP3 ** SQLITE_MUTEX_STATIC_VFS1 ** SQLITE_MUTEX_STATIC_VFS2 ** SQLITE_MUTEX_STATIC_VFS3 ** *** The first two constants cause sqlite3_mutex_alloc() to create ** a new mutex. The new mutex is recursive when SQLITE_MUTEX_RECURSIVE ** is used but not necessarily so when SQLITE_MUTEX_FAST is used. ** The mutex implementation does not need to make a distinction ** between SQLITE_MUTEX_RECURSIVE and SQLITE_MUTEX_FAST if it does ** not want to. But SQLite will only request a recursive mutex in ** cases where it really needs one. If a faster non-recursive mutex ** implementation is available on the host platform, the mutex subsystem ** might return such a mutex in response to SQLITE_MUTEX_FAST. ** ** The other allowed parameters to sqlite3_mutex_alloc() each return ** a pointer to a static preexisting mutex. Six static mutexes are ** used by the current version of SQLite. Future versions of SQLite ** may add additional static mutexes. Static mutexes are for internal ** use by SQLite only. Applications that use SQLite mutexes should ** use only the dynamic mutexes returned by SQLITE_MUTEX_FAST or ** SQLITE_MUTEX_RECURSIVE. ** ** Note that if one of the dynamic mutex parameters (SQLITE_MUTEX_FAST ** or SQLITE_MUTEX_RECURSIVE) is used then sqlite3_mutex_alloc() ** returns a different mutex on every call. But for the static ** mutex types, the same mutex is returned on every call that has ** the same type number.
label: code-design
 72907. Number of terms in subquery FROM clause
 72908. Create a list of user columns for the virtual table
 72909. cnearset
 72910. Virtual table of this cursor
 72911. ** Allocate and return a pointer to an expression to load the column iCol ** from datasource iSrc in SrcList pSrc.
 72912. Number of def. imm constraints when stmt started
 72913. If the journal has been truncated, simply stop reading and ** processing the journal. This might happen if the journal was ** not completely written and synced prior to a crash. In that ** case, the database should have never been written in the ** first place so it is OK to simply abandon the rollback.
 72914. ** op1=INSERT, op2=INSERT -> Unsupported. Discard op2. ** op1=INSERT, op2=UPDATE -> INSERT. ** op1=INSERT, op2=DELETE -> (none) *** ** op1=UPDATE, op2=INSERT -> Unsupported. Discard op2. ** op1=UPDATE, op2=UPDATE -> UPDATE. ** op1=UPDATE, op2=DELETE -> DELETE. *** ** op1=DELETE, op2=INSERT -> UPDATE. ** op1=DELETE, op2=UPDATE -> Unsupported. Discard op2. ** op1=DELETE, op2=DELETE -> Unsupported. Discard op2.
 72915. SQLITE OMIT LIKE OPTIMIZATION
 72916. Number of slots used in aPoint[]
 72917. **comment:** ** Like malloc(), but remember the size of the allocation ** so that we can find it later using sqlite3MemSize(). ** ** For this low-level routine, we are guaranteed that nByte>0 because ** cases of nByte<=0 will be intercepted and dealt with by higher level ** routines.
label: code-design
 72918. ** This function is called to free superfluous dynamically allocated memory ** held by the pager system. Memory in use by any SQLite pager allocated ** by the current thread may be sqlite3_free(jed). *** nReq is the number of bytes of memory required. Once this much has ** been released, the function returns. The return value is the total number ** of bytes of memory released.
 72919. Current leaf block (or 0)
 72920. array of column indexes
 72921. isError!=0 or pVdbe->pAuxData modified
 72922. Fts5 table object
 72923. ** Set the current error code to err_code and clear any prior error message. ** Also set iSysErrno (by calling sqlite3System) if the err_code indicates ** that would be appropriate.
 72924. memset(pPager->aHash, 0, sizeof(pPager->aHash));
 72925. ** Return a pointer to the next prepared statement after pStmt associated ** with database connection pDb. If pStmt is NULL, return the first ** prepared statement for the database connection. Return NULL if there ** are no more.
 72926. Forward references to internal structures
 72927. Size of every page

72928. ** CAPI3REF: SQLite Runtime Status *** ^These interfaces are used to retrieve runtime status information ** about the performance of SQLite, and optionally to reset various ** highwater marks. ^The first argument is an integer code for ** the specific parameter to measure. ^{Recognized integer codes ** are of the form [status parameters | SQLITE_STATUS_...].}^ ** ^The current value of the parameter is returned into *pCurrent. ** ^The highest recorded value is returned in *pHighwater. ^If the ** resetFlag is true, then the highest record value is reset after ** *pHighwater is written. ^{Some parameters do not record the highest ** value. For those parameters ** nothing is written into *pHighwater and the resetFlag is ignored.}^ ** ^{Other parameters record only the highwater mark and not the current ** value. For these latter parameters nothing is written into *pCurrent.}^ *** ^{The sqlite3_status() and sqlite3_status64() routines return ** SQLITE_OK on success and a non-zero [error code] on failure.}^ *** If either the current value or the highwater mark is too large to ** be represented by a 32-bit integer, then the values returned by ** sqlite3_status() are undefined. *** See also: [sqlite3_db_status()]

72929. Table structure - used by update and pre-update hooks

72930. The ORDER BY clause is ignored for all of the above

72931. Sanity checking on the page. This is more important than I originally ** thought. If a power failure occurs while the journal is being written, ** it could cause invalid data to be written into the journal. We need to ** detect this invalid data (with high probability) and ignore it.

72932. The query strategy is to look for an equality constraint on the json ** column. Without such a constraint, the table cannot operate. idxNum is ** 1 if the constraint is found, 3 if the constraint and zRoot are found, ** and 0 otherwise.

72933. Current 'end' value

72934. ** The return value of winGetLastErrorMsg ** is zero if the error message fits in the buffer, or non-zero ** otherwise (if the message was truncated).

72935. Triggers for aAction[] actions

72936. Snippet end text - ""

72937. ** Helper subroutine for PRAGMA integrity_check: ** ** Generate code to output a single-column result row with the result ** held in register regResult. Decrement the result count and halt if ** the maximum number of result rows have been issued.

72938. 2-byte unsigned integer

72939. Free temporary memory and return success

72940. Flags for sqlite3WhereBegin()

72941. We need to prevent a random number of 0x8000000000000000 ** (or -9223372036854775808) since when you do abs() of that ** number of you get the same value back again. To do this ** in a way that is testable, mask the sign bit off of negative ** values, resulting in a positive value. Then take the ** 2s complement of that positive value. The end result can ** therefore be no less than -9223372036854775807.

72942. Hash table containing tokenizers

72943. ** Convert a BLOB literal of the form "x'hhhhh'" into its binary ** value. Return a pointer to its binary value. Space to hold the ** binary value has been obtained from malloc and must be freed by ** the calling routine.

72944. **comment:** Malloc may fail when setting the page-size, as there is an internal ** buffer that the pager module resizes using sqlite3_realloc().
label: code-design

72945. Statement for INSERT operations

72946. shared can just be set because we always have an exclusive

72947. Pointer to the data to be bound

72948. Start the view context.

72949. Unary operator

72950. Calculate the 'docid' delta value to write into the merged ** doclist.

72951. Changeset to apply

72952. DESC => ID

72953. ** Retry flock() calls that fail with EINTR

72954. SQLITE_STATUS_PAGECACHE_OVERFLOW

72955. Second part of [schema].jid field, or NULL

72956. One of TK_BEFORE, TK_AFTER, TK_INSTEAD

72957. Determine how many bytes of data to read.

72958. **comment:** ** Undo the hack that converts floating point types to integer for ** builds on processors without floating point support.
label: code-design

72959. Extra data for callback

72960. xDeviceCharacteristics

72961. True if there are one or more Incrblob cursors

72962. ** Perform a single step of an incremental-vacuum. If successful, return ** SQLITE_OK. If there is no work to do (and therefore no point in ** calling this function again), return SQLITE_DONE. Or, if an error ** occurs, return some other error code. ** ** More specifically, this function attempts to re-organize the database so ** that the last page of the file currently in use is no longer in use. ** ** Parameter nFin is the number of pages that this database would contain ** were this function called until it returns SQLITE_DONE. ** ** If the bCommit parameter is non-zero, this function assumes that the ** caller will keep calling incrVacuumStep() until it returns SQLITE_DONE ** or an error. bCommit is passed true for an auto-vacuum-on-commit ** operation, or false for an incremental vacuum.

72963. ** Use the zone allocator available on apple products unless the ** SQLITE_WITHOUT_ZONEMALLOC symbol is defined.

72964. The tokenizer

72965. Note that because the size of the allocation for p->a[] is not ** necessarily a power of two, sqlite3IdListAppend() may not be called ** on the duplicate created by this function.

72966. Name of table into which we are inserting

72967. Virtual machine being coded

72968. IMP: R-53341-35419

72969. Used to iterate through segments

72970. 590

72971. ** Move a statvfs cursor to the next entry in the file.

72972. Generate code to read the child key values into registers ** regRow..regRow+n. If any of the child key values are NULL, this ** row cannot cause an FK violation. Jump directly to addrOk in ** this case.

72973. ** The methods above are in version 1 of the sqlite_vfs object ** definition. Those that follow are added in version 2 or later

72974. GLOB

72975. Constant expression is reusable

72976. ** The maximum length of a single SQL statement in bytes. ** ** It used to be the case that setting this value to zero would ** turn the limit off. That is no longer true. It is not possible ** to turn this limit off.

72977. Name of savepoint

72978. True to flush contents of memory to PMA

72979. nearest ::= STRING LP nearphrases neardist_opt RP

72980. Doclist is assembled here

72981. Jump here for next iteration of skip-scan

72982. Size of apChange[] array

72983. Number of WAL frames backfilled into DB

72984. Prefix length (0 for main terms index)

72985. ** Add code to implement the OFFSET

72986. Used to build up list of table cols

72987. Jump here when skipping the initialization

72988. between_op ::= BETWEEN

72989. ** This function is called as part of an sqlite3VdbeSorterRewind() operation ** on a sorter that has written two or more PMAs to temporary files. It sets ** up either VdbeSorter.pMerger (for single threaded sorters) or pReader ** (for multi-threaded sorters) so that it can be used to iterate through ** all records stored in the sorter. ** ** SQLITE_OK is returned if successful, or an SQLite error code otherwise.

72990. Result code from fcntl()

72991. ** Recursively walk the expressions of a SELECT statement and generate ** a bitmask indicating which tables are used in that expression ** tree.
72992. ** Return TRUE if there is a vowel anywhere within z[0..n-1]
72993. All triggers indexed by name
72994. The 1-byte case. Overwhelmingly the most common. Handled inline ** by the getVarin32() macro
72995. Current index in aRight
72996. **comment:** First unused byte of aOvflSpace[]
 label: code-design
72997. If *pPgno refers to a pointer-map page, allocate two new pages ** at the end of the file instead of one. The first allocated page ** becomes a new pointer-map page, the second is used by the caller.
72998. ** A convenience macro that returns the number of elements in ** an array.
72999. **comment:** Unused parameter
 label: code-design
73000. ** The following structure contains pointers to all SQLite API routines. ** A pointer to this structure is passed into extensions when they are ** loaded so that the extension can make calls back into the SQLite ** library. ** ** When adding new APIs, add them to the bottom of this structure ** in order to preserve backwards compatibility. ** ** Extensions that use newer APIs should first call the ** sqlite3_libversion_number() to make sure that the API they ** intend to use is supported by the library. Extensions should ** also check to make sure that the pointer to the function is ** not NULL before calling it.
73001. ***** End makeheaders token definitions *****
73002. Result of sqlite3Strlen30(zTab)
73003. ***** End of auth.c *****
73004. ** Release the STATIC_MASTER mutex.
73005. Number of PMAs currently in file
73006. ** Compute the absolute value of a 32-bit signed integer, or possible. Or ** if the integer has a value of -2147483648, return +2147483647
73007. ** Print an IOTRACE message showing SQL content.
73008. True to cause the name to be dequoted
73009. Used by: stats
73010. Size of db file in bytes
73011. ** End of interface to code in fts5_buffer.c. *****
73012. ** The following asserts make sure that structures used by the btree are ** the right size. This is to guard against size changes that result ** when compiling on a different architecture.
73013. Device sector size
73014. This is the database that must be used
73015. **comment:** If the file was truncated to a size smaller than the currently ** mapped region, reduce the effective mapping size as well. SQLite will ** use read() and write() to access data beyond this point from now on.
 label: code-design
73016. PGHDR flags defined below
73017. Write the specified cookie value
73018. Retrive the text stored in column iCol. If an SQL NULL is stored ** in column iCol, jump immediately to the next iteration of the loop. ** If an OOM occurs while retrieving the data (this can happen if SQLite ** needs to transform the data from utf-16 to utf-8), return SQLITE_NOMEM ** to the caller.
73019. True if the delete flag is set
73020. memcmp() result
73021. ** Initialize a node-reader object to read the node in buffer aNode/nNode. ** ** If successful, SQLITE_OK is returned and the NodeReader object set to ** point to the first entry on the node (if any). Otherwise, an SQLite ** error code is returned.
73022. unixFile created and returned by ref
73023. Fragment number
73024. **comment:** Bytes of unused memory
 label: code-design
73025. An optimization. If the database was not actually modified during ** this transaction, the pager is running in exclusive-mode and is ** using persistent journals, then this function is a no-op. ** ** The start of the journal file currently contains a single journal ** header with the nRec field set to 0. If such a journal is used as ** a hot-journal during hot-journal rollback, 0 changes will be made ** to the database file. So there is no need to zero the journal ** header. Since the pager is in exclusive mode, there is no need ** to drop any locks either.
73026. **comment:** Next unused scratch buffer
 label: code-design
73027. all other styles use the locking context to store the db file path
73028. Size of aSlot[] array
73029. Finder function name
73030. The expression to check authorization on
73031. First byte of the cell content area
73032. Ideally, the start of the snippet should be pushed forward in the ** document nDesired tokens. This block checks if there are actually ** nDesired tokens to the right of the snippet. If so, *piPos and ** *pHiMask are updated to shift the snippet nDesired tokens to the ** right. Otherwise, the snippet is shifted by the number of tokens ** available.
73033. The "wsdPrg" macro will resolve to the pseudo-random number generator ** state vector. If writable static data is unsupported on the target, ** we have to locate the state vector at run-time. In the more common ** case where writable static data is supported, wsdPrg can refer directly ** to the "sqlite3Prng" state vector declared above.
73034. ** An object of type Fts5SegWriter is used to write to segments.
73035. ** Given a btree page and a cell index (0 means the first cell on ** the page, 1 means the second cell, and so forth) return a pointer ** to the cell content. ** ** findCellPastPtr() does the same except it skips past the initial ** 4-byte child pointer found on interior pages, if there is one. ** ** This routine works only for pages that do not contain overflow cells.
73036. SQLITE_OMIT_PAGER_PRAGMAS && !SQLITE_OMIT_DEPRECATED
73037. The HAVING clause. May be NULL
73038. The code generator maintains a stack of active WITH clauses ** with the inner-most WITH clause being at the top of the stack. ** ** This routine pushes the WITH clause passed as the second argument ** onto the top of the stack. If argument bFree is true, then this ** WITH clause will never be popped from the stack. In this case it ** should be freed along with the Parse object. In other cases, when ** bFree==0, the With object will be freed along with the SELECT ** statement with which it is associated.
73039. ** Encode value iVal as an SQLite varint and append it to the buffer object ** pBuf. If an OOM error occurs, set the error code in p.
73040. FOREIGN
73041. The cell to write
73042. A term of the ORDER BY clause
73043. Both values must be blobs. Compare using memcmp().
73044. DEFAULT
73045. True for "ORDER BY rowid DESC" queries
73046. result variable
73047. 'usermerge' setting
73048. Initial bulk allocation size
73049. Value being transferred
73050. ifnotexists ::=
73051. If using a docid=? or rowid=? strategy, set the UNIQUE flag.
73052. ** Record 0 of the %_stat table contains a blob consisting of N varints, ** where N is the number of user defined columns in the fts3 table plus ** two. If nCol is the number of user defined columns, then values of the ** varints are set as follows: ** ** Varint 0: Total number of rows in the table. ** ** Varint 1..nCol: For

each column, the total number of tokens stored in ** the column for all rows of the table. *** Varint 1+nCol: The total size, in bytes, of all text values in all ** columns of all rows of the table. **

73053. synopsis: if (->[P1])==0 goto P2

73054. The associated pager. Also accessible by pBt->pPager

73055. Space allocated for zExtra[]

73056. NULL NULL

73057. ** CAPI3REF: Deprecated Soft Heap Limit Interface ** DEPRECATED *** This is a deprecated version of the [sqlite3_soft_heap_limit64()] ** interface. This routine is provided for historical compatibility ** only. All new applications should use the ** [sqlite3_soft_heap_limit64()] interface rather than this one.

73058. If zKey is non-NULL, then this foreign key was declared to ** map to an explicit list of columns in table pParent. Check if this ** index matches those columns. Also, check that the index uses ** the default collation sequences for each column.

73059. A bound parameter in a CREATE statement that originates from ** sqlite3_prepare() causes an error

73060. We should already hold a lock on the database connection

73061. Assert that the caller has been consistent. If this cursor was opened ** expecting an index b-tree, then the caller should be inserting blob ** keys with no associated data. If the cursor was opened expecting an ** intkey table, the caller should be inserting integer keys with a ** blob of associated data.

73062. 0x70 .. 0x7F

73063. ABORT

73064. ** CAPI3REF: Compile-Time Library Version Numbers *** ^The [SQLITE_VERSION] C preprocessor macro in the sqlite3.h header ** evaluates to a string literal that is the SQLite version in the ** format "X.Y.Z" where X is the major version number (always 3 for ** SQLite3) and Y is the minor version number and Z is the release number.)^ ** ^The [SQLITE_VERSION_NUMBER] C preprocessor macro resolves to an integer ** with the value (X*1000000 + Y*1000 + Z) where X, Y, and Z are the same ** numbers used in [SQLITE_VERSION].)^ ** The SQLITE_VERSION_NUMBER for any given release of SQLite will also ** be larger than the release from which it is derived. Either Y will ** be held constant and Z will be incremented or else Y will be incremented ** and Z will be reset to zero. *** Since [version 3.6.18] ([dateof:3.6.18]), ** SQLite source code has been stored in the ** Fossil configuration management ** system. ^The SQLITE_SOURCE_ID macro evaluates to ** a string which identifies a particular check-in of SQLite ** within its configuration management system. ^The SQLITE_SOURCE_ID ** string contains the date and time of the check-in (UTC) and a SHA1 ** or SHA3-256 hash of the entire source tree. *** See also: [sqlite3_libversion()], ** [sqlite3_libversion_number()], [sqlite3_sourceid()], ** [sqlite_version()] and [sqlite_source_id()].

73065. ** An instance of this structure (in the form of a BLOB) is returned by ** the SQL functions that sqlite3_rtree_geometry_callback() and ** sqlite3_rtree_query_callback() create, and is read as the right-hand ** operand to the MATCH operator of an R-Tree.

73066. Destination for SELECT on rhs of INSERT

73067. Not a compound SELECT

73068. In the amalgamation, the os_unix.c and os_win.c source files come before ** this source file. Verify that the #defines of the locking byte offsets ** in os_unix.c and os_win.c agree with the WALINDEX_LOCK_OFFSET value. ** For that matter, if the lock offset ever changes from its initial design ** value of 120, we need to know that so there is an assert() to check it.

73069. Check that the column is not part of an FK child key definition. It ** is not necessary to check if it is part of a parent key, as parent ** key columns must be indexed. The check below will pick up this ** case.

73070. Wal iterator context

73071. turn off high bit in final byte

73072. ** Indicate that the statement currently under construction might write ** more than one entry (example: deleting one row then inserting another, ** inserting multiple rows in a table, or inserting a row and index entries.) ** If an abort occurs after some of these writes have completed, then it will ** be necessary to undo the completed writes.

73073. If a tokenizer= option was successfully parsed, the tokenizer has ** already been allocated. Otherwise, allocate an instance of the default ** tokenizer (unicode61) now.

73074. Handle describing change and conflict

73075. Make sure no sibling connections hold locks that will block this ** lock. If any do, return SQLITE_BUSY right away.

73076. # include <sys/types.h>

73077. Value of pPage->aData[0]

73078. Expression that pNear is a part of

73079. Region to retrieve

73080. PRAGMA index_xinfo = <pk-index>

73081. Unsigned for byte-order conversions

73082. NOTINDEXED

73083. The memcpy() statement assumes that the wildcard characters are ** the first three statements in the compareInfo structure. The ** asserts() that follow verify that assumption

73084. Table being updated or deleted from

73085. Next hash entry with same hash-key

73086. **comment:** *** 2008 November 05 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** ***** This file implements the default page cache implementation (the ** sqlite3_pcachef interface). It also contains part of the implementation ** of the SQLITE_CONFIG_PAGECACHE and sqlite3_release_memory() features. ** If the default page cache implementation is overridden, then neither of ** these two features are available. *** A Page cache line looks like this: *** -----

----- | database page content | PgHdr1 | MemPage | PgHdr | ** *** The database page content is up front (so that buffer overreads tend to ** flow harmlessly into the PgHdr1, MemPage, and PgHdr extensions). MemPage ** is the extension added by the btree.c module containing information such ** as the database page number and how that database page is used. PgHdr ** is added by the pcache.c layer and contains information used to keep track ** of which pages are "dirty". PgHdr1 is an extension added by this ** module (pcache1.c). The PgHdr1 header is a subclass of sqlite3_pcachef_page. ** PgHdr1 contains information needed to look up a page by its page number. ** The superclass sqlite3_pcachef_page.pBuf points to the start of the ** database page content and sqlite3_pcachef_page.pExtra points to PgHdr. ** ** The size of the extension (MemPage+PgHdr+PgHdr1) can be determined at ** runtime using sqlite3_config(SQLITE_CONFIG_PCACHE_HDRSZ, &size). The ** sizes of the extensions sum to 272 bytes on x64 for 3.8.10, but this ** size can vary according to architecture, compile-time options, and ** SQLite library version number. *** If SQLITE_PCACHE_SEPARATE_HEADER is defined, then the extension is obtained ** using a separate memory allocation from the database page content. This ** seeks to overcome the "clownshoe" problem (also called "internal ** fragmentation" in academic literature) of allocating a few bytes more ** than a power of two with the memory allocator rounding up to the next ** power of two, and leaving the rounded-up space unused. *** This module tracks pointers to PgHdr1 objects. Only pcache.c communicates ** with this module. Information is passed back and forth as PgHdr1 pointers. ** ** The pcache.c and pager.c modules deal pointers to PgHdr objects. ** The btree.c module deals with pointers to MemPage objects. *** SOURCE OF PAGE CACHE MEMORY: *** Memory for a page might come from any of three sources: *** (1) The general-purpose memory allocator - sqlite3Malloc() ** (2) Global page-cache memory provided using sqlite3_config() with ** SQLITE_CONFIG_PAGECACHE. ** (3) PCache-local bulk allocation. *** The third case is a chunk of heap memory (defaulting to 100 pages worth) ** that is allocated when the page cache is created. The size of the local ** bulk allocation can be adjusted using *** *** sqlite3_config(SQLITE_CONFIG_PAGECACHE, (void*)0, 0, N). ** ** If N is positive, then N pages worth of memory are allocated using a single ** sqlite3Malloc() call and that memory is used for the first N pages allocated. ** Or if N is negative, then -1024*N bytes of memory are allocated and used ** for as many pages as can be accommodated. *** Only one of (2) or (3) can be used. Once the memory available to (2) or ** (3) is exhausted, subsequent allocations fail over to the general-purpose ** memory allocator (1). *** Earlier versions of SQLite used only methods (1) and (2). But experiments ** show that method (3) with N==100 provides about a 5% performance boost for ** common workloads.

label: code-design

73087. Bottom of the rowid change constraint check

73088. ** Unlock an rbuVfs-file.

73089. ** If the user has not set the safety-level for this database connection ** using "PRAGMA synchronous", and if the safety-level is not already ** set to the value passed to this function as the second parameter, ** set it so.

73090. ** Return TRUE if the page given in the argument was previously passed ** to sqlite3PagerWrite(). In other words, return TRUE if it is ok ** to change the content of the page.

73091. RESTRICT => ID

73092. SQLITE_PRINTF flags below
73093. 316
73094. The P1 operand
73095. From ANALYZE: Est. rows selected by each column
73096. **comment:** Function context - put error messages here
 label: code-design
73097. A regular table
73098. ERROR: copy exceeds output file size
73099. True if the outermost savepoint is a TS
73100. AS => nothing
73101. Parameters passed to SQL geom function
73102. Set up the Stat4Accum.a[] and aBest[] arrays
73103. ** Generate a Token object from a string
73104. Name context for parent SELECT statement
73105. Result of a Windows lock call
73106. ***** sqlite3_bind_ ***** Routines used to attach values to wildcards in a compiled SQL statement.
73107. 293
73108. IN
73109. 196
73110. If the leaf in question has already been trimmed from the segment, ** ignore this b-tree entry. Otherwise, load it into memory.
73111. Both X and Y have COLLATE operators. Make sure X is always ** used by clearing the EP_Collate flag from Y.
73112. Register holding rowid
73113. The BTREE cursor
73114. True for an infix function: LIKE, GLOB, etc
73115. ***** Begin file malloc.c *****
73116. ** Used only when SQLITE_NO_SYNC is not defined.
73117. 169
73118. Segment term filter condition
73119. Last frame in list
73120. Must be of the form INSERT INTO ... SELECT ...
73121. ** Argument pWith (which may be NULL) points to a linked list of nested ** WITH contexts, from inner to outermost. If the table identified by ** FROM clause element pItem is really a common-table-expression (CTE) ** then return a pointer to the CTE definition for that table. Otherwise ** return NULL. ** * If a non-NULL value is returned, set *ppContext to point to the With ** object that the returned CTE belongs to.
73122. ABC in prefix=ABC parameter to parse
73123. Must make sure nOverflow is reset to zero even if the balance() ** fails. Internal data structure corruption will result otherwise. ** Also, set the cursor state to invalid. This stops saveCursorPosition() ** from trying to save the current position of the cursor.
73124. "x IN (value, value, ...)"
73125. **comment:** ** The ALWAYS and NEVER macros surround boolean expressions which ** are intended to always be true or false, respectively. Such ** expressions could be omitted from the code completely. But they ** are included in a few cases in order to enhance the resilience ** of SQLite to unexpected behavior - to make the code "self-healing" ** or "ductile" rather than being "brittle" and crashing at the first ** hint of unplanned behavior. ** * In other words, ALWAYS and NEVER are added for defensive code. ** * When doing coverage testing ALWAYS and NEVER are hard-coded to ** be true and false so that the unreachable code they specify will ** not be counted as untested code.
 label: code-design
73126. SQLITE_THREADSAFE>0 above. SQLITE_THREADS=0 below
73127. ** Disconnect all the virtual table objects in the sqlite3.pDisconnect list. ** * This function may only be called when the mutexes associated with all ** shared b-tree databases opened using connection db are held by the ** caller. This is done to protect the sqlite3.pDisconnect list. The ** sqlite3.pDisconnect list is accessed only as follows: ** * 1) By this function. In this case, all BtShared mutexes and the mutex ** associated with the database handle itself must be held. ** * 2) By function vtabDisconnectAll(), when it adds a VTable entry to ** the sqlite3.pDisconnect list. In this case either the BtShared mutex ** associated with the database the virtual table is stored in is held ** or, if the virtual table is stored in a non-sharable database, then ** the database handle mutex is held. ** * As a result, a sqlite3.pDisconnect cannot be accessed simultaneously ** by multiple threads. It is thread-safe.
73128. ** 2003 April 6 ** * The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** * May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
 ***** This file contains code used to implement the VACUUM command.
 ** * Most of the code in this file may be omitted by defining the ** SQLITE OMIT_VACUUM macro.
73129. Offset into aDigits[] of the digits string
73130. ** An instance of the following structure is used to store the busy-handler ** callback for a given sqlite handle. ** * The sqlite.busyHandler member of the sqlite struct contains the busy ** callback for the database handle. Each pager opened via the sqlite ** handle is passed a pointer to sqlite.busyHandler. The busy-handler ** callback is currently invoked only from within pager.c.
73131. Opcode: Pagecount P1 P2 * * * * * Write the current number of pages in database P1 to memory cell P2.
73132. ** Implementation of the sqlite3_pcache.xUnpin method. ** * Mark a page as unpinned (eligible for asynchronous recycling).
73133. Number of tokens in query
73134. 44
73135. IMP: R-38091-32352
73136. The optimizer is able to deliver rows in group by order so ** we do not have to sort. The OP_OpenEphemeral table will be ** cancelled later because we still need to use the pKeyInfo
73137. 123
73138. OUT: Total number of frames checkpointed
73139. 258
73140. The result code
73141. ***** End of hwtime.h *****
73142. Build and compile a statement to execute:
73143. Store pointer to the allocated page here
73144. Allocate space for the aMSI[] array of each FTSQUERY_PHRASE node
73145. ORDER BY correctly sorts the inner loop
73146. Number of outstanding xFetch references
73147. This token started out using characters that can appear in keywords, ** but z[i] is a character not allowed within keywords, so this must ** be an identifier instead
73148. docid>=?
73149. If zKey is NULL, then this foreign key is implicitly mapped to ** the PRIMARY KEY of table pParent. The PRIMARY KEY index may be ** identified by the test.
73150. If the BtCursor.aOverflow[] has not been allocated, allocate it now. ** * The aOverflow[] array is sized at one entry for each overflow page ** in the overflow chain. The page number of the first overflow page is ** stored in aOverflow[0], etc. A value of 0 in the aOverflow[] array ** means "not yet known" (the cache is lazily populated).
73151. Top and bottom range constraints
73152. An element of aTo[] that we are working on
73153. Computed column name as a token
73154. HAVING
73155. ** Convert a multi-byte character string to UTF-8. ** * Space to hold the returned string is obtained from sqlite3_malloc().

73156. If not building as part of the core, include sqlite3ext.h.
73157. mnHeap, mxHeap
73158. ** Merge two position-lists as required by the NEAR operator. The argument ** position lists correspond to the left and right phrases of an expression ** like: ** "phrase 1" NEAR "phrase number 2" ** ** Position list *pp1 corresponds to the left-hand side of the NEAR ** expression and *pp2 to the right. As usual, the indexes in the position ** lists are the offsets of the last token in each phrase (tokens "1" and "2" ** in the example above). ** ** The output position list - written to *pp - is a copy of *pp2 with those ** entries that are not sufficiently NEAR entries in *pp1 removed.
73159. This is the descendent of an OR node. In this case we cannot use ** an incremental phrase. Load the entire doclist for the phrase ** into memory in this case.
73160. ** Return the N-dimensional column of the cell stored in *p.
73161. ** Implementation of the sqlite_version() function. The result is the version ** of the SQLite library that is running.
73162. Reassembly the compound query so that it will be freed correctly ** by the calling function
73163. List of in-memory records
73164. Find the memory cell that will be used to store the blob of memory ** required for this VdbeCursor structure. It is convenient to use a ** vdbe memory cell to manage the memory allocation required for a ** VdbeCursor structure for the following reasons: ** ** * Sometimes cursor numbers are used for a couple of different ** purposes in a vdbe program. The different uses might require ** different sized allocations. Memory cells provide growable ** allocations. ** ** * When using ENABLE_MEMORY_MANAGEMENT, memory cell buffers can ** be freed lazily via the sqlite3_release_memory() API. This ** minimizes the number of malloc calls made by the system. ** ** The memory cell for cursor 0 is aMem[0]. The rest are allocated from ** the top of the register space. Cursor 1 is at Mem[p->nMem-1]. ** Cursor 2 is at Mem[p->nMem-2]. And so forth.
73165. Generate a subroutine that will fill an ephemeral table with ** the content of this subquery. pItem->addrFillSub will point ** to the address of the generated subroutine. pItem->regReturn ** is a register allocated to hold the subroutine return address
73166. Buffer containing page data
73167. **comment:** sqlite3_malloc()
label: code-design
73168. ** Return the P5 value that should be used for a binary comparison ** opcode (OP_Eq, OP_Ge etc.) used to compare pExpr1 and pExpr2.
73169. P3 Value for OP_VFilter
73170. Not all WHERE terms resolved by outer loop
73171. Used internally by sqlite3Fts3SegReaderXXX() calls
73172. File handle to write to
73173. Trigger being finished
73174. First token of best snippet
73175. ** CAPI3REF: Run-time Limits ** METHOD: sqlite3 ** ** ^ (This interface allows the size of various constructs to be limited ** on a connection by connection basis. The first parameter is the ** [database connection] whose limit is to be set or queried. The ** second parameter is one of the [limit categories] that define a ** class of constructs to be size limited. The third parameter is the ** new limit for that construct.)^ ** ** ^ If the new limit is a negative number, the limit is unchanged. ** ^ (For each limit category SQLITE_LIMIT_<i>NAME</i> there is a ** [limits | hard upper bound] ** set at compile-time by a C preprocessor macro called ** [limits | SQLITE_MAX_<i>NAME</i>]. ** (The ".LIMIT_" in the name is changed to ".MAX_".))^ ** ^ Attempts to increase a limit above its hard upper bound are ** silently truncated to the hard upper bound. ** ** ^ Regardless of whether or not the limit was changed, the ** [sqlite3_limit()] interface returns the prior value of the limit. ** ^ Hence, to find the current value of a limit without changing it, ** simply invoke this interface with the third parameter set to -1. ** ** Run-time limits are intended for use in applications that manage ** both their own internal database and also databases that are controlled ** by untrusted external sources. An example application might be a ** web browser that has its own databases for storing history and ** separate databases controlled by JavaScript applications downloaded ** off the Internet. The internal databases can be given the ** large, default limits. Databases managed by external sources can ** be given much smaller limits designed to prevent a denial of service ** attack. Developers might also want to use the [sqlite3_set_authorizer()] ** interface to further control untrusted SQL. The size of the database ** created by an untrusted script can be contained using the ** [max_page_count] [PRAGMA]. ** ** New run-time limit categories may be added in future releases.
73176. **comment:** ** The following two macros are used within the PAGERTRACE() macros above ** to print out file-descriptors. ** ** PAGERID() takes a pointer to a Pager struct as its argument. The ** associated file-descriptor is returned. FILEHANDLEID() takes an sqlite3_file ** struct as its argument.
label: code-design
73177. IMP: R-11967-56545
73178. ** PRAGMA [schema.]cache_spill ** PRAGMA cache_spill=BOOLEAN ** PRAGMA [schema.]cache_spill=N ** ** The first form reports the current local setting for the ** page cache spill size. The second form turns cache spill on ** or off. When turning cache spill on, the size is set to the ** current cache_size. The third form sets a spill size that ** may be different from the cache size. ** If N is positive then that is the ** number of pages in the cache. If N is negative, then the ** number of pages is adjusted so that the cache uses -N kilobytes ** of memory. ** ** If the number of cache_spill pages is less than the number of ** cache_size pages, no spilling occurs until the page count exceeds ** the number of cache_size pages. ** ** The cache_spill=BOOLEAN setting applies to all attached schemas, ** not just the schema specified.
73179. target database handle
73180. Size of the rendering buffer
73181. Search for prior reference to this subquery
73182. True to use background threads
73183. At this point (pTargetFd->iCookie) contains the value of the ** change-counter cookie (the thing that gets incremented when a ** transaction is committed in rollback mode) currently stored on ** page 1 of the database file.
73184. Buffer of at least *pnList items to use
73185. Write out the page data.
73186. ** Implementation of the length() function
73187. ** Extract the iCol-th column from the nRec-byte record in pRec. Write ** the column value into *ppVal. If *ppVal is initially NULL then a new ** sqlite3_value object is allocated. ** ** If *ppVal is initially NULL then the caller is responsible for ** ensuring that the value written into *ppVal is eventually freed.
73188. ** This function is used to obtain an SQLite prepared statement handle ** for the statement identified by the second argument. If successful, ** *pp is set to the requested statement handle and SQLITE_OK returned. ** Otherwise, an SQLite error code is returned and *pp is set to 0. ** ** If argument apVal is not NULL, then it must point to an array with ** at least as many entries as the requested statement has bound ** parameters. The values are bound to the statements parameters before ** returning.
73189. **comment:** ** This function is called to estimate the number of rows visited by a ** range-scan on a skip-scan index. For example: ** ** CREATE INDEX i1 ON t1(a, b, c); ** SELECT * FROM t1 WHERE a=? AND c BETWEEN ? AND ?; ** ** Value pLoop->nOut is currently set to the estimated number of rows ** visited for scanning (a=? AND b=?). This function reduces that estimate ** by some factor to account for the (c BETWEEN ? AND ?) expression based ** on the stat4 data for the index. this scan will be performed multiple ** times (once for each (a,b) combination that matches a=? is dealt with ** by the caller. ** ** It does this by scanning through all stat4 samples, comparing values ** extracted from pLower and pUpper with the corresponding column in each ** sample. If L and U are the number of samples found to be less than or ** equal to the values extracted from pLower and pUpper respectively, and ** N is the total number of samples, the pLoop->nOut value is adjusted ** as follows: ** ** nOut = nOut * (min(U - L, 1) / N) ** ** If pLower is NULL, or a value cannot be extracted from the term, L is ** set to zero. If pUpper is NULL, or a value cannot be extracted from it, ** U is set to N. ** ** Normally, this function sets *pbDone to 1 before returning. However, ** if no value can be extracted from either pLower or pUpper (and so the ** estimate of the number of rows delivered remains unchanged), *pbDone ** is left as is. ** ** If an error occurs, an SQLite error code is returned. Otherwise, ** SQLITE_OK.
label: code-design
73190. ** Internally, the vdbe manipulates nearly all SQL values as Mem ** structures. Each Mem struct may cache multiple representations (string, ** integer etc.) of the same value.
73191. Bitmask of other loops that must run first
73192. True if there exists a MATCH constraint
73193. For storing the record being decoded
73194. Never called on page 1
73195. TUNING: Cost of a unique index lookup is 15
73196. ** Macros for looping over all elements of a hash table. The idiom is ** like this: ** ** Hash h; ** HashElem *p; ** ... ** for(p=sqliteHashFirst(&h); p; p=sqliteHashNext(p)){ ** SomeStructure *pData = sqliteHashData(p); ** // do something with pData ** }

73197. ** Attempt to advance the node-reader object passed as the first argument to ** the next entry on the node. ** ** Return an error code if an error occurs (SQLITE_NOMEM is possible). ** Otherwise return SQLITE_OK. If there is no next entry on the node ** (e.g. because the current entry is the last) set NodeReader->aNode to ** NULL to indicate EOF. Otherwise, populate the NodeReader structure output ** variables for the new entry.

73198. The WHERE clause of the outer query

73199. expr ::= JOIN_KW

73200. Append N bytes from zIn onto the end of the JsonString string.

73201. This is VxWorks. Set up things specially for that OS

73202. Imposter table name

73203. Array of shared library handles

73204. 138

73205. Left table being joined

73206. The group the new page cache will belong to

73207. First token of snippet

73208. Result code

73209. Turn bulk memory into a hash table object by initializing the ** fields of the Hash structure. *** "pNew" is a pointer to the hash table that is to be initialized.

73210. **comment:** todo
label: requirement

73211. Index Name Table Name

73212. 1 = unlock, 0 = lock

73213. ** Check if the contents of the FTS index match the current contents of the ** content table. If no error occurs and the contents do match, set *pbOk ** to true and return SQLITE_OK. Or if the contents do not match, set *pbOk ** to false before returning. *** If an error occurs (e.g. an OOM or IO error), return an SQLite error ** code. The final value of *pbOk is undefined in this case.

73214. **comment:** If a pre-update hook is registered and this is a write cursor, ** invoke it here. *** TODO: The preupdate-hook is passed SQLITE_DELETE, even though this ** operation should really be an SQLITE_UPDATE. This is probably ** incorrect, but is convenient because at this point the new.* values ** are not easily obtainable. And for the sessions module, an ** SQLITE_UPDATE where the PK columns do not change is handled in the ** same way as an SQLITE_DELETE (the SQLITE_DELETE code is actually ** slightly more efficient). Since you cannot write to a PK column ** using the incremental-blob API, this works. For the sessions module ** anyhow.
label: code-design

73215. ***** Include hash.h in the middle of sqliteInt.h *****

73216. !defined(SQLITE_CORE) || defined(SQLITE_ENABLE_JSON1)

73217. The point in pPage->aCellIdx[] where no cell inserted

73218. ** Compare strings z1 and z2, returning 0 if they are identical, or non-zero ** otherwise. Either or both argument may be NULL. Two NULL values are ** considered equal, and NULL is considered distinct from all other values.

73219. ** A single Vdbe is an opaque structure named "Vdbe". Only routines ** in the source file sqliteVdbe.c are allowed to see the insides ** of this structure.

73220. nStmtSpill

73221. Suppress error messages if VIEW already exists

73222. MemTranslate() may return SQLITE_OK or SQLITE_NOMEM. If NOMEM is returned, ** then the encoding of the value may not have changed.

73223. IMP: R-59593-21810

73224. Read the rowid of the current row of the WHERE scan. In ONEPASS_OFF ** mode, write the rowid into the FIFO. In either of the one-pass modes, ** leave it in register regOldRowid.

73225. List of WHEN terms

73226. This branch is taken if the set of sibling pages somehow contains ** duplicate entries. This can happen if the database is corrupt. ** It would be simpler to detect this as part of the loop below, but ** we do the detection here in order to avoid populating the pager ** cache with two separate objects associated with the same ** page number.

73227. Saved value of p->iLimit

73228. Value to serialize

73229. ** Free the list of sorted records starting at pRecord.

73230. **comment:** ** CAPI3REF: Count The Number Of Rows Modified ** METHOD: sqlite3 *** ^This function returns the number of rows modified, inserted or ** deleted by the most recently completed INSERT, UPDATE or DELETE ** statement on the database connection specified by the only parameter. ** ^Executing any other type of SQL statement does not modify the value ** returned by this function. *** ^Only changes made directly by the INSERT, UPDATE or DELETE statement are ** considered - auxiliary changes caused by [CREATE TRIGGER | triggers], ** [foreign key actions] or [REPLACE] constraint resolution are not counted. *** Changes to a view that are intercepted by ** [INSTEAD OF trigger | INSTEAD OF triggers] are not counted. ^The value ** returned by sqlite3_changes() immediately after an INSERT, UPDATE or ** DELETE statement run on a view is always zero. Only changes made to real ** tables are counted. *** Things are more complicated if the sqlite3_changes() function is ** executed while a trigger program is running. This may happen if the ** program uses the [changes] SQL function, or if some other callback ** function invokes sqlite3_changes() directly. Essentially: *** ** ^Before entering a trigger program the value returned by ** sqlite3_changes() function is saved. After the trigger program ** has finished, the original value is restored.)^ *** ^Within a trigger program each INSERT, UPDATE and DELETE ** statement sets the value returned by sqlite3_changes() ** upon completion as normal. Of course, this value will not include ** any changes performed by sub-triggers, as the sqlite3_changes() ** value will be saved and restored after each sub-trigger has run.)^ *** ** ^This means that if the changes() SQL function (or similar) is used ** by the first INSERT, UPDATE or DELETE statement within a trigger, it ** returns the value as set when the calling statement began executing. ** ^If it is used by the second or subsequent such statement within a trigger ** program, the value returned reflects the number of rows modified by the ** previous INSERT, UPDATE or DELETE statement within the same trigger. *** See also the [sqlite3_total_changes()] interface, the ** [count_changes pragma], and the [changes() SQL function]. *** If a separate thread makes changes on the same database connection ** while [sqlite3_changes()] is running then the value returned ** is unpredictable and not meaningful.
label: code-design

73231. Vdbe is ready to execute

73232. ... is correct.

73233. Number of messages written to zErrMsg so far

73234. ** Convert an SQL-style quoted string into a normal string by removing ** the quote characters. The conversion is done in-place. If the ** input does not begin with a quote character, then this routine ** is a no-op. *** Examples: *** "abc" becomes abc ** 'xyz' becomes xyz ** [pqr] becomes pqr ** `mno` becomes mno **

73235. Database index for <database>

73236. P4 is a pointer to a FuncDef structure

73237. loop counter

73238. ** Parameter eStat must be either SQLITE_DBSTATUS_CACHE_HIT or ** SQLITE_DBSTATUS_CACHE_MISS. Before returning, *pnVal is incremented by the ** current cache hit or miss count, according to the value of eStat. If the ** reset parameter is non-zero, the cache hit or miss count is zeroed before ** returning.

73239. ** Create a new bitmap object able to handle bits between 0 and iSize, ** inclusive. Return a pointer to the new object. Return NULL if ** malloc fails.

73240. Segment reader object

73241. What do we want to learn about the zPath file?

73242. The operand token for setting the span

73243. ** Register a callback to be invoked each time a transaction is written ** into the write-ahead-log by this database connection.

73244. ** An instance of this structure is used by the parser to record both ** the parse tree for an expression and the span of input text for an ** expression.

73245. Step 4. If the RHS is known to be non-NULL and we did not find ** an match on the search above, then the result must be FALSE.

73246. List of locks held on this shared-btree struct

73247. List of free mmap page headers (pDirty)

73248. Name of database (e.g. "main")

73249. If expensive assert() statements are available, do a linear search ** of the wal-index file content. Make sure the results agree with the ** result obtained using the hash indexes above.

73250. 78..7f xyz{}~.

73251. Pointer to buffer containing record

73252. Start of memory holding current results

73253. Rowid of first leaf block to traverse

73254. ** Maximum supported path-length.

73255. ** Return a block of memory of at least nBytes in size. ** Return NULL if unable. *** This function assumes that the necessary mutexes, if any, are ** already held by the caller. Hence "Unsafe".

73256. full bitset of atomics from max sector size and smaller

73257. ** Set the name of a Select object

73258. ** CAPI3REF: Prepared Statement Object ** KEYWORDS: {prepared statement} {prepared statements} *** An instance of this object represents a single SQL statement that ** has been compiled into binary form and is ready to be evaluated. *** Think of each SQL statement as a separate computer program. The ** original SQL text is source code. A prepared statement object ** is the compiled object code. All SQL must be converted into a ** prepared statement before it can be run. *** The life-cycle of a prepared statement object usually goes like this: *** ** Create the prepared statement object using [sqlite3_prepare_v2()]. ** Bind values to [parameters] using the sqlite3_bind_*() ** interfaces. ** Run the SQL by calling [sqlite3_step()] one or more times. ** Reset the prepared statement using [sqlite3_reset()] then go back ** to step 2. Do this zero or more times. ** Destroy the object using [sqlite3_finalize()]. **

73259. True if we can share pBt with another db

73260. ** Return a JsonNode and all its descendants as a JSON string.

73261. **comment:** ** Return the system page size. *** This function should not be called directly by other code in this file. ** Instead, it should be called via macro osGetpagesize().

label: code-design

73262. The pager that owns pPg

73263. ** This is the xRowid method. The SQLite core calls this routine to ** retrieve the rowid for the current row of the result set. The ** rowid should be written to *pRowid.

73264. nbr of 1st byte locked if successful

73265. Copy of initial value of pIn3->flags

73266. Enable load_extension

73267. Number of pages in apOld[]

73268. 1

73269. Special case: If this is an INSERT statement that will insert exactly ** one row into the table, raise a constraint immediately instead of ** incrementing a counter. This is necessary as the VM code is being ** generated for will not open a statement transaction.

73270. **comment:** Malloc must have failed

label: code-design

73271. View Name NULL

73272. OUT: size of *apIndex[] array

73273. Built-in count(*) aggregate

73274. Documents in table

73275. Register that records whether NULLs exist in RHS

73276. sqlite3_test_control(SQLITE_TESTCTRL_RESERVED, sqlite3 *db, int N) *** Set the nReserve size to N for the main database on the database ** connection db.

73277. The pattern string B

73278. OFFSET

73279. RTREE_LE and RTREE_LT end here

73280. ** Allocate a new rowid. This is used for "external content" tables when ** a NULL value is inserted into the rowid column. The new rowid is allocated ** by inserting a dummy row into the %_docszie table. The dummy will be ** overwritten later. *** If the %_docszie table does not exist, SQLITE_MISMATCH is returned. In ** this case the user is required to provide a rowid explicitly.

73281. same as TK_GT, jump, in1, in3

73282. Infinity

73283. Original text of the expression

73284. ** Within this division (the proxying locking implementation) the procedures ** above this point are all utilities. The lock-related methods of the ** proxy-locking sqlite3_io_method object follow.

73285. Saved trace settings

73286. Tokenizer for this cursor.

73287. xGetSystemCall

73288. The page being overwritten.

73289. ***** Include wal.h in the middle of pager.c *****

73290. ** This routine is called after all opcodes have been inserted. It loops ** through all the opcodes and fixes up some details. *** (1) For each jump instruction with a negative P2 value (a label) ** resolve the P2 value to an actual address. *** (2) Compute the maximum number of arguments used by any SQL function ** and store that value in *pMaxFuncArgs. *** (3) Update the Vdbe.readOnly and Vdbe.blsReader flags to accurately ** indicate what the prepared statement actually does. *** (4) Initialize the p4.xAdvance pointer on opcodes that use it. *** (5) Reclaim the memory allocated for storing labels. *** This routine will only function correctly if the mkopcodeh.tcl generator ** script numbers the opcodes correctly. Changes to this routine must be ** coordinated with changes to mkopcodeh.tcl.

73291. Output success/failure

73292. WHERE rowid IN (select)

73293. Number of bytes of cell payload

73294. IN/OUT: Preallocated output buffer

73295. ** Elements of sqlite3Stat[] are protected by either the memory allocator ** mutex, or by the pcache1 mutex. The following array determines which.

73296. Rowid for record being decoded

73297. If the checksum doesn't add up, then one or more of the disk sectors ** containing the master journal filename is corrupted. This means ** definitely roll back, so just return SQLITE_OK and report a (nul) ** master-journal filename.

73298. Wal-index version

73299. Error message written here

73300. onconf ::= ON CONFLICT resolvetype

73301. ** Define various macros that are missing from some systems.

73302. ** If we are not using shared cache, then there is no need to ** use mutexes to access the BtShared structures. So make the ** Enter and Leave procedures no-ops.

73303. Current depth of the r-tree structure

73304. ** Maximum length of a varint encoded integer. The varint format is different ** from that used by SQLite, so the maximum length is 10, not 9.

73305. SQLITE_NULLEQ is clear and at least one operand is NULL, ** then the result is always NULL. ** The jump is taken if the SQLITE_JUMPIFNULL bit is set.

73306. Left doclist

73307. Cells stored on pPg

73308. The matching pSrcList item

73309. ** Delete a VdbeFrame object and its contents. VdbeFrame objects are ** allocated by the OP_Program opcode in sqlite3VdbeExec().

73310. EVIDENCE-OF: R-37497-42412 The size of the reserved region is ** determined by the one-byte unsigned integer found at an offset of 20 ** into the database file header.

73311. First memory cell containing the PRIMARY KEY

73312. Set the limiter.

73313. uncompress=? parameter (or NULL)

73314. out2

73315. Current expected value of %_config table 'version' field
73316. The reference count on pShmNode has already been incremented under ** the cover of the winShmEnterMutex() mutex and the pointer from the ** new (struct winShm) object to the pShmNode has been set. All that is ** left to do is to link the new object into the linked list starting ** at pShmNode->pFirst. This must be done while holding the pShmNode->mutex ** mutex.
73317. Reverse the order of IN operators
73318. mxStrlen
73319. ** CAPI3REF: Reset Automatic Extension Loading *** ^This interface disables all automatic extensions previously ** registered using [sqlite3_auto_extension()].
73320. ** The write transaction open on pPager is being committed (bCommit==1) ** or rolled back (bCommit==0). ** ** Return TRUE if and only if all dirty pages should be flushed to disk. *** Rules: *** * For non-TEMP databases, always sync to disk. This is necessary ** for transactions to be durable. *** * Sync TEMP database only on a COMMIT (not a ROLLBACK) when the backing ** file has been created already (via a spill on pagerStress()) and ** when the number of dirty pages in memory exceeds 25% of the total ** cache size.
73321. ** Convert a UTF-8 string to Microsoft Unicode. ** ** Space to hold the returned string is obtained from sqlite3_malloc().
73322. ** The code generator calls this routine if it discovers that it is ** possible to abort a statement prior to completion. In order to ** perform this abort without corrupting the database, we need to make ** sure that the statement is protected by a statement transaction. *** Technically, we only need to set the mayAbort flag if the ** isMultiWrite flag was previously set. There is a time dependency ** such that the abort must occur after the multiwrite. This makes ** some statements involving the REPLACE conflict resolution algorithm ** go a little faster. But taking advantage of this time dependency ** makes it more difficult to prove that the code is correct (in ** particular, it prevents us from writing an effective ** implementation of sqlite3AssertMayAbort()) and so we have chosen ** to take the safe route and skip the optimization.
73323. If no YMD specified, assume 2000-Jan-01
73324. SQLITE_ENABLE_LOCKING_STYLE
73325. FTS5 global context for db handle
73326. First key to write
73327. If the index uses a collation sequence that is different from ** the default collation sequence for the column, this index is ** unusable. Bail out early in this case.
73328. ***** Begin file alter.c *****
73329. FAIL
73330. Case 1: Reset the single schema identified by iDb
73331. 302
73332. Default behavior: Report an error if the argument to VACUUM is ** not recognized
73333. OUT: expression
73334. EVIDENCE-OF: R-22710-53328 The third and fourth bytes of each ** freeblock form a big-endian integer which is the size of the freeblock ** in bytes, including the 4-byte header.
73335. Closing the handle. Fourth parameter is passed the value 2.
73336. Put any active cursors into REQUIRE_SEEK state.
73337. 0x0a
73338. ** These routines walk (recursively) an expression tree and generate ** a bitmask indicating which tables are used in that expression ** tree.
73339. ** This function implements the ChooseLeaf algorithm from Gutman[84]. ** ChooseSubTree in r*tree terminology.
73340. refact ::= SET DEFAULT
73341. ** CAPI3REF: Function Flags *** ** These constants may be ORed together with the ** [SQLITE_UTF8 | preferred text encoding] as the fourth argument ** to [sqlite3_create_function()], [sqlite3_create_function16()], or ** [sqlite3_create_function_v2()].
73342. same as TK_BITOR, synopsis: r[P3]=r[P1]|r[P2]
73343. 223
73344. sizeof(MemPage)+sizeof(PgHdr)
73345. Select statement to read idx values
73346. True if entry is not a match
73347. ** Advance the position list iterator specified by the first two ** arguments so that it points to the first element with a value greater ** than or equal to parameter iNext.
73348. a: p0 (unmasked)
73349. Compare the term we are searching for with the term just loaded from ** the interior node. If the specified term is greater than or equal ** to the term from the interior node, then all terms on the sub-tree ** headed by node iChild are smaller than zTerm. No need to search ** iChild. *** If the interior node term is larger than the specified term, then ** the tree headed by iChild may contain the specified term.
73350. ** Translate from TK_xx operator to WO_xx bitmask.
73351. Cannot fail given valid arguments
73352. xCachesize
73353. Loop through all the foreign key constraints for which pTab is the ** child table (the table that the foreign key definition is part of).
73354. 260
73355. Pull the requested columns.
73356. ** Process time function arguments. argv[0] is a date-time stamp. ** argv[1] and following are modifiers. Parse them all and write ** the resulting time into the DateTime structure p. Return 0 ** on success and 1 if there are any errors. *** If there are zero parameters (if even argv[0] is undefined) ** then assume a default value of "now" for argv[0].
73357. Pending terms hash table to add entry to
73358. eidlist_opt ::= LP eidlist RP
73359. Thread that is within this mutex
73360. ***** Start of unicode61 tokenizer implementation.
73361. 318
73362. ** PRAGMA shrink_memory *** ** IMPLEMENTATION-OF: R-23445-46109 This pragma causes the database ** connection on which it is invoked to free up as much memory as it ** can, by calling sqlite3_db_release_memory().
73363. Database holding the shared memory
73364. ** Return true if the BtShared mutex is held on the btree, or if the ** B-Tree is not marked as sharable. *** ** This routine is used only from within assert() statements.
73365. **comment:** Index of the equality term within this level
 label: code-design
73366. ** Allocate and return a buffer at least nByte bytes in size. ** ** If an OOM error is encountered, return NULL and set the error code in ** the Fts5Index handle passed as the first argument.
73367. Check that one of SQLITE_SYNC_NORMAL or FULL was passed
73368. When multiple overflows occur, they are always sequential and in ** sorted order. This invariants arise because multiple overflows can ** only occur when inserting divider cells into the parent page during ** balancing, and the dividers are adjacent and sorted.
73369. The loop that contains the LIKE operator
73370. **comment:** ** Try to adjust the cost of WhereLoop pTemplate upwards or downwards so ** that: ** ** (1) pTemplate costs less than any other WhereLoops that are a proper ** subset of pTemplate ** ** (2) pTemplate costs more than any other WhereLoops for which pTemplate ** is a proper subset. *** To say "WhereLoop X is a proper subset of Y" means that X uses fewer ** WHERE clause terms than Y and that every WHERE clause term used by X is ** also used by Y.
 label: code-design
73371. 36
73372. ** Invoke the authorization callback for permission to read column zCol from ** table zTab in database zDb. This function assumes that an authorization ** callback has been registered (i.e. that sqlite3.xAuth is not NULL). *** If SQLITE_IGNORE is returned and pExpr is not NULL, then pExpr is changed ** to an SQL NULL expression. Otherwise, if pExpr is NULL, then SQLITE_IGNORE ** is treated as SQLITE_DENY. In this case an error is left in pParse.

73373. **comment:** ** File Locking Notes: (Mostly about windows but also some info for Unix) *** ** We cannot use LockFileEx() or UnlockFileEx() on Win95/98/ME because ** those functions are not available. So we use only LockFile() and ** UnlockFile(). *** ** LockFile() prevents not just writing but also reading by other processes. ** A SHARED_LOCK is obtained by locking a single randomly-chosen ** byte out of a specific range of bytes. The lock byte is obtained at ** random so two separate readers can probably access the file at the ** same time, unless they are unlucky and choose the same lock byte. ** An EXCLUSIVE_LOCK is obtained by locking all bytes in the range. ** There can only be one writer. A RESERVED_LOCK is obtained by locking ** a single byte of the file that is designated as the reserved lock byte. ** A PENDING_LOCK is obtained by locking a designated byte different from ** the RESERVED_LOCK byte. *** ** On WinNT/2K/XP systems, LockFileEx() and UnlockFileEx() are available, ** which means we can use reader/writer locks. When reader/writer locks ** are used, the lock is placed on the same range of bytes that is used ** for probabilistic locking in Win95/98/ME. Hence, the locking scheme ** will support two or more Win95 readers or two or more WinNT readers. ** But a single Win95 reader will lock out all WinNT readers and a single ** WinNT reader will lock out all other Win95 readers. *** ** The following #defines specify the range of bytes used for locking. ** SHARED_SIZE is the number of bytes available in the pool from which ** a random byte is selected for a shared lock. The pool of bytes for ** shared locks begins at SHARED_FIRST. *** ** The same locking strategy and ** byte ranges are used for Unix. This leaves open the possibility of having ** clients on win95, winNT, and unix all talking to the same shared file ** and all locking correctly. To do so would require that samba (or whatever ** tool is being used for file sharing) implements locks correctly between ** windows and unix. I'm guessing that isn't likely to happen, but by ** using the same locking range we are at least open to the possibility. *** ** Locking in windows is mandatory. For this reason, we cannot store ** actual data in the bytes used for locking. The pager never allocates ** the pages involved in locking therefore. SHARED_SIZE is selected so ** that all locks will fit on a single page even at the minimum page size. ** PENDING_BYTE defines the beginning of the locks. By default PENDING_BYTE ** is set high so that we don't have to allocate an unused page except ** for very large databases. But one should test the page skipping logic ** by setting PENDING_BYTE low and running the entire regression suite. *** ** Changing the value of PENDING_BYTE results in a subtly incompatible ** file format. Depending on how it is changed, you might not notice ** the incompatibility right away, even running a full regression test. ** The default location of PENDING_BYTE is the first byte past the ** 1GB boundary. **

label: code-design

73374. Select id (left-most output column)

73375. 319

73376. SQLITE_USER_AUTHENTICATION

73377. ** Include the header file for the Windows VFS.

73378. Column to filter on

73379. Size of write buffer in bytes

73380. ORDER BY processing for max() func

73381. distinct ::=

73382. **comment:** ** This routine deallocates a previously ** allocated mutex. SQLite is careful to deallocate every ** mutex that it allocates.

label: code-design

73383. Argument to pass to xNotify

73384. segdir.leaves_end_block

73385. mmap() space per open file

73386. The VDBE cursor used by this IN operator

73387. ** Some API routines are omitted when various features are ** excluded from a build of SQLite. Substitute a NULL pointer ** for any missing APIs.

73388. Delete entries for this table or index

73389. ** Return the number of columns in the result set for the statement pStmt.

73390. **comment:** Mask of tables that must be used.

label: code-design

73391. Methods above are valid for version 3

73392. Coordinate value converted to a double

73393. The return code

73394. Offset of wal file entry

73395. Populate the register containing the index name.

73396. State 3. The integer just read is a column number.

73397. ** This is a helper function for rbuObjIterCacheTableInfo(). It populates ** the pIter->abIndexed[] array.

73398. Used during OR-clause processing

73399. ** Pop an authorization context that was previously pushed ** by sqlite3AuthContextPush

73400. ***** End of the flock lock implementation *****

73401. Open the storage sub-system

73402. If not NULL, the master journal name

73403. Current write is a delete

73404. 0x40..0x4F

73405. Number of distinct values in index

73406. ** Query FTS for the tokenizer implementation named zName.

73407. Number of used entries in aColl[]

73408. One of SQLITE_INSERT, UPDATE, DELETE

73409. Parser context.

73410. ** Set the number of result columns that will be returned by this SQL ** statement. This is now set at compile time, rather than during ** execution of the vdbe program so that sqlite3_column_count() can ** be called on an SQL statement before sqlite3_step().

73411. True if holding the winShmNode mutex

73412. EV: R-30323-21917 Each foreign key constraint in SQLite is ** classified as either immediate or deferred.

73413. **comment:** ** Given a file descriptor, locate the unixInodeInfo object that ** describes that file descriptor. Create a new one if necessary. The ** return value might be uninitialized if an error occurs. *** ** The mutex entered using the unixEnterMutex() function must be held ** when this function is called. *** ** Return an appropriate error code.

label: code-design

73414. ** CAPI3REF: Reset All Bindings On A Prepared Statement ** METHOD: sqlite3_stmt ** ** ^Contrary to the intuition of many, [sqlite3_reset()] does not reset ** the [sqlite3_bind_blob | bindings] on a [prepared statement]. ** ^Use this routine to reset all host parameters to NULL.

73415. ** Transform a UTF-8 integer literal, in either decimal or hexadecimal, ** into a 64-bit signed integer. This routine accepts hexadecimal literals, ** whereas sqlite3Atoi64() does not. ** ** Returns: ** ** 0 Successful transformation. Fits in a 64-bit signed integer. ** 1 Integer too large for a 64-bit signed integer or is malformed ** 2 Special case of 9223372036854775808

73416. Register holding the AUTOINCREMENT counter

73417. Fill in pNew->pLeft and pNew->pRight.

73418. True if previously initialized. MUST BE FIRST!

73419. List of dirty pages to revert

73420. ** Free memory previously obtained from sqlite3Malloc().

73421. pAppData

73422. Check for complete coverage of the page

73423. ** The pcache1.separateCache variable is true if each PCache has its own ** private PGroup (mode-1). pcache1.separateCache is false if the single ** PGroup in pcache1.grp is used for all page caches (mode-2). *** ** Always use a unified cache (mode-2) if ENABLE_MEMORY_MANAGEMENT *** ** Use a unified cache in single-threaded applications that have ** configured a start-time buffer for use as page-cache memory using ** sqlite3_config(SQLITE_CONFIG_PAGECACHE, pBuf, sz, N) with non-NULL ** pBuf argument. *** ** Otherwise use separate caches (mode-1)

73424. Buffer used for any other block

73425. Vector expression to be appended. Might be NULL

73426. unixFile.pInode is always valid here. Otherwise, a different close ** routine (e.g. nolockClose()) would be called instead.

73427. Base class

73428. **comment:** Even though there is no encoding conversion, value_blob() might ** need to call malloc() to expand the result of a zeroblob() ** expression.
label: code-design

73429. SQLITE_OMIT_FLOATING_POINT

73430. SQLITE_NOTFOUND

73431. pMem->flags = 0; // sqlite3VdbeSerialGet() will set this for us

73432. Collating sequence to be reindexed, or NULL

73433. True if pList should be sqlite3_free()d

73434. Set pTC to point to the cheapest remaining token.

73435. Size of PMA in bytes

73436. Construct a record from the query result, but instead of ** saving that record, use it as a key to delete elements from ** the temporary table iParm.

73437. ** CAPI3REF: Evaluate An SQL Statement ** METHOD: sqlite3_stmt ** ** After a [prepared statement] has been prepared using any of ** [sqlite3_prepare_v2()], [sqlite3_prepare_v3()], [sqlite3_prepare16_v2()], ** or [sqlite3_prepare16_v3()] or one of the legacy ** interfaces [sqlite3_prepare()] or [sqlite3_prepare16()] this function ** must be called one or more times to evaluate the statement. ** ** The details of the behavior of the sqlite3_step() interface depend ** on whether the statement was prepared using the newer "vX" interfaces ** [sqlite3_prepare_v3()], [sqlite3_prepare_v2()], [sqlite3_prepare16_v3()], ** [sqlite3_prepare16_v2()] or the older legacy ** interfaces [sqlite3_prepare()] and [sqlite3_prepare16()]. The use of the ** new "vX" interface is recommended for new applications but the legacy ** interface will continue to be supported. ** ** ^In the legacy interface, the return value will be either [SQLITE_BUSY], ** [SQLITE_DONE], [SQLITE_ROW], [SQLITE_ERROR], or [SQLITE_MISUSE]. ** ^With the "v2" interface, any of the other [result codes] or ** [extended result codes] might be returned as well. ** ** ^[SQLITE_BUSY] means that the database engine was unable to acquire the ** database locks it needs to do its job. ^If the statement is a [COMMIT] ** or occurs outside of an explicit transaction, then you can retry the ** statement. If the statement is not a [COMMIT] and occurs within an ** explicit transaction then you should rollback the transaction before ** continuing. ** ** ^[SQLITE_DONE] means that the statement has finished executing ** successfully. sqlite3_step() should not be called again on this virtual ** machine without first calling [sqlite3_reset()] to reset the virtual ** machine back to its initial state. ** ** ^If the SQL statement being executed returns any data, then [SQLITE_ROW] ** is returned each time a new row of data is ready for processing by the ** caller. The values may be accessed using the [column access functions]. ** sqlite3_step() is called again to retrieve the next row of data. ** ** ^[SQLITE_ERROR] means that a run-time error (such as a constraint ** violation) has occurred. sqlite3_step() should not be called again on ** the VM. More information may be found by calling [sqlite3_errmsg()]. ** ^With the legacy interface, a more specific error code (for example, ** [SQLITE_INTERRUPT], [SQLITE_SCHEMA], [SQLITE_CORRUPT], and so forth) ** can be obtained by calling [sqlite3_reset()] on the ** [prepared statement]. ^In the "v2" interface, ** the more specific error code is returned directly by sqlite3_step(). ** ** [SQLITE_MISUSE] means that the this routine was called inappropriately. ** Perhaps it was called on a [prepared statement] that has ** already been [sqlite3_finalize | finalized] or on one that had ** previously returned [SQLITE_ERROR] or [SQLITE_DONE]. Or it could ** be the case that the same database connection is being used by two or ** more threads at the same moment in time. ** ** For all versions of SQLite up to and including 3.6.23.1, a call to ** [sqlite3_reset()] was required after sqlite3_step() returned anything ** other than [SQLITE_ROW] before any subsequent invocation of ** sqlite3_step(). Failure to reset the prepared statement using ** [sqlite3_reset()] would result in an [SQLITE_MISUSE] return from ** sqlite3_step(). But after [version 3.6.23.1] ([dateof:3.6.23.1]), ** sqlite3_step() began ** calling [sqlite3_reset()] automatically in this circumstance rather ** than returning [SQLITE_MISUSE]. This is not considered a compatibility ** break because any application that ever receives an [SQLITE_MISUSE] error ** is broken by definition. The [SQLITE_OMIT_AUTORESET] compile-time option ** can be used to restore the legacy behavior. ** ** Goofy Interface Alert: In the legacy interface, the sqlite3_step() ** API always returns a generic error code, [SQLITE_ERROR], following any ** error other than [SQLITE_BUSY] and [SQLITE_MISUSE]. You must call ** [sqlite3_reset()] or [sqlite3_finalize()] in order to find one of the ** specific [error codes] that better describes the error. ** We admit that this is a goofy design. The problem has been fixed ** with the "v2" interface. If you prepare all of your SQL statements ** using [sqlite3_prepare_v3()] or [sqlite3_prepare_v2()] ** or [sqlite3_prepare16_v2()] or [sqlite3_prepare16_v3()] instead ** of the legacy [sqlite3_prepare()] and [sqlite3_prepare16()] interfaces, ** then the more specific [error codes] are returned directly ** by sqlite3_step(). The use of the "vX" interfaces is recommended.

73438. ** pColumns and pExpr form a vector assignment which is part of the SET ** clause of an UPDATE statement. Like this: ** ** (a,b,c) = (expr1,expr2,expr3) ** Or: (a,b,c) = (SELECT x,y,z FROM ...) ** ** For each term of the vector assignment, append new entries to the ** expression list pList. In the case of a subquery on the RHS, append ** TK_SELECT_COLUMN expressions.

73439. Real value of right operand

73440. ** Generate code for a comparison operator.

73441. SQLITE_IGNORE_FLOCK_LOCK_ERRORS

73442. ** Compare two UTF-8 strings for equality where the first string is ** a GLOB or LIKE expression. Return values: ** ** SQLITE_MATCH: Match ** SQLITE_NOMATCH: No match ** SQLITE_NOWILDCARDMATCH: No match in spite of having * or % wildcards. ** ** Globbing rules: ** ** '*' Matches any sequence of zero or more characters. ** ** '?' Matches exactly one character. ** ** [...] Matches one character from the enclosed list of ** characters. ** ** [...] Matches one character not in the enclosed list. ** ** With the [...] and [...] matching, a ')' character can be included ** in the list by making it the first character after '[' or '^'. A ** range of characters can be specified using '-'. Example: ** "[a-z]" matches any single lower-case letter. To match a '-', make ** it the last character in the list. ** ** Like matching rules: ** ** '%' Matches any sequence of zero or more characters ** ** '_' Matches any one character ** ** Ee Where E is the "esc" character and c is any other ** character, including '%', '_', and esc, match exactly c. ** ** The comments within this routine usually assume glob matching. ** ** This routine is usually quick, but can be N**2 in the worst case.

73443. ** Generate code that initializes multiple registers to string or integer ** constants. The registers begin with iDest and increase consecutively. ** One register is initialized for each character in zTypes[]. For each ** "s" character in zTypes[], the register is a string if the argument is ** not NULL, or OP_Null if the value is a null pointer. For each "I" character ** in zTypes[], the register is initialized to an integer. ** ** If the input string does not end with "X" then an OP_ResultRow instruction ** is generated for the values inserted.

73444. Next entry in sorted order

73445. Pointer to buffer containing changeset

73446. Accumulate results here

73447. **comment:** ** If the third argument is non-NULL, then this function releases a ** reference obtained by an earlier call to winFetch(). The second ** argument passed to this function must be the same as the corresponding ** argument that was passed to the winFetch() invocation. ** ** Or, if the third argument is NULL, then this function is being called ** to inform the VFS layer that, according to POSIX, any existing mapping ** may now be invalid and should be unmapped.
label: code-design

73448. ** Remove from colset pColset any columns that are not also in colset pMerge.

73449. Allocation is valid

73450. Next in LRU list of unpinned pages

73451. Detach this backup from the source pager.

73452. Copy of third arg to preupdate_hook()

73453. ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** Internal interface definitions for SQLite. **

73454. ** Call this routine to record the fact that an OOM (out-of-memory) error ** has happened. This routine will set db->mallocFailed, and also ** temporarily disable the lookaside memory allocator and interrupt ** any running VDBEs.

73455. The parent key is a composite key that includes the IPK column

73456. Size of a[] in bytes

73457. ** Convert a filename from whatever the underlying operating system ** supports for filenames into UTF-8. Space to hold the result is ** obtained from malloc and must be freed by the calling function.

73458. Description of the record

73459. sqlite3CompareAffinity() only returns TEXT if one side or the ** other has no affinity and the other side is TEXT. Hence, ** the only way for cmpaff to be TEXT is for idxaff to be TEXT ** and for the term on the LHS of the IN to have no affinity.

73460. ***** Begin file icu.c *****

73461. data_version when pStruct read

73462. TermSelect object to merge into

73463. ** Initialize bulk memory to be a consistent Mem object. ** ** The minimum amount of initialization feasible is performed.

73464. ** Write an unsigned 32-bit value in big-endian format to the supplied ** buffer.

73465. ** Generate code for the REINDEX command. *** ** REINDEX -- 1 ** REINDEX <collation> -- 2 ** REINDEX ?<database>.<tablename> -- 3 ** REINDEX ?<database>.<indexname> -- 4 *** ** Form 1 causes all indices in all attached databases to be rebuilt. ** Form 2 rebuilds all indices in all databases that use the named ** collating function. Forms 3 and 4 rebuild the named index or all ** indices associated with the named table.

73466. OUT: A pointer to the prepared statement

73467. Highwater mark for nOut

73468. Name of target db index (or null)

73469. 93

73470. ** Return true if the prepared statement is guaranteed to not modify the ** database.

73471. Additional information associated with pTerm

73472. Bytes of content in a[]. Must be a multiple of 8.

73473. Extract the value into this register

73474. Create the imposter table or tables (if required).

73475. Uses the OpenEphemeral opcode

73476. **comment:** ** Interpret the given string as a safety level. Return 0 for OFF, ** 1 for ON or NORMAL, 2 for FULL, and 3 for EXTRA. Return 1 for an empty or ** unrecognized string argument. The FULL and EXTRA option is disallowed ** if the omitFull parameter is 1. *** ** Note that the values returned are one less than the values that ** should be passed into sqlite3BtreeSetSafetyLevel(). This is done ** to support legacy SQL code. The safety level used to be boolean ** and older scripts may have used numbers 0 for OFF and 1 for ON.

label: code-design

73477. ***** Begin file treeview.c *****

73478. funcFlags

73479. Name of function

73480. ** 2009 Oct 23 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file is part of the SQLite FTS3 extension module. Specifically, ** this file contains code to insert, update and delete rows from FTS3 ** tables. It also contains code to merge FTS3 b-tree segments. Some ** of the sub-routines used to merge segments are also used by the query ** code in fts3.c.

73481. ** Delete an entire VDBE.

73482. ** Use nodeAcquire() to obtain the leaf node containing the record with ** rowid iRowid. If successful, set *ppLeaf to point to the node and ** return SQLITE_OK. If there is no such record in the table, set ** *ppLeaf to 0 and return SQLITE_OK. If an error occurs, set *ppLeaf ** to zero and return an SQLite error code.

73483. Prevent by getAndInitPage()

73484. **comment:** ** This function is used by SQL generated to implement the ** ALTER TABLE command. The first argument is the text of a CREATE TABLE or ** CREATE INDEX command. The second is a table name. The table name in ** the CREATE TABLE or CREATE INDEX statement is replaced with the third ** argument and the result returned. Examples: *** ** sqlite_rename_table('CREATE TABLE abc(a, b, c)', 'def') ** -> 'CREATE TABLE def(a, b, c)' *** ** sqlite_rename_table('CREATE INDEX i ON abc(a)', 'def') ** -> 'CREATE INDEX i ON def(a, b, c)'

label: code-design

73485. Index of sub-iterator just advanced

73486. The caller is responsible for zeroing output parameters.

73487. Advance the pragma virtual table cursor to the next row

73488. input ::= expr

73489. ** Call sqlite3WalkExpr() for every expression in list p or until ** an abort request is seen.

73490. Column to query (or -ve for all columns)

73491. Make sure sufficient number of registers have been allocated

73492. Cannot have triggers on a virtual table. If it were possible, ** this block would have to account for hidden column.

73493. Malloc'd space (possibly) used for zTerm

73494. The vdbe must make its own copy of the column-type and other ** column specific strings, in case the schema is reset before this ** virtual machine is deleted.

73495. If this is the first frame written into the log, write the WAL ** header to the start of the WAL file. See comments at the top of ** this source file for a description of the WAL header format.

73496. ** Delete all entries in the FTS5 index.

73497. Database filenames are double-zero terminated if they are not ** URIs with parameters. Hence, they can always be passed into ** sqlite3_uri_parameter().

73498. Omit all terms smaller than this

73499. True for a root-only reader

73500. Output register from the sorter

73501. ** Register this backup object with the associated source pager for ** callbacks when pages are changed or the cache invalidated.

73502. 1020

73503. ** Add connection db to the blocked connections list. It is assumed ** that it is not already a part of the list.

73504. Do as much as possible w/o blocking

73505. Number of the rule by which to reduce

73506. Number of terms contributing to sumEq

73507. If sorting index that was created by a prior OP_OpenEphemeral ** instruction ended up not being needed, then change the OP_OpenEphemeral ** into an OP_Noop.

73508. colset ::= LCP colsetlist RCP

73509. FROM entries not usable at this level

73510. Deleting this row means the whole table is empty. In this case ** delete the contents of all three tables and throw away any ** data in the pendingTerms hash table.

73511. Used by: index_info

73512. Dest pager

73513. True for min() and max() aggregates

73514. The subquery uses a single slot of the FROM clause of the outer ** query. If the subquery has more than one element in its FROM clause, ** then expand the outer query to make space for it to hold all elements ** of the subquery. *** ** Example: *** ** SELECT * FROM tabA, (SELECT * FROM sub1, sub2), tabB; *** ** The outer query has 3 slots in its FROM clause. One slot of the ** outer query (the middle slot) is used by the subquery. The next ** block of code will expand the outer query FROM clause to 4 slots. ** The middle slot is expanded to two slots in order to make space ** for the two elements in the FROM clause of the subquery.

73515. ** Title text to insert in front of each block

73516. ** This function is equivalent to the standard isspace() function. *** ** The standard isspace() can be awkward to use safely, because although it ** is defined to accept an argument of type int, its behavior when passed ** an integer that falls outside of the range of the unsigned char type ** is undefined (and sometimes, "undefined" means segfault). This wrapper ** is defined to accept an argument of type char, and always returns 0 for ** any values that fall outside of the range of the unsigned char type (i.e. ** negative values).

73517. Encoding to use

73518. Disallow both ON and USING clauses in the same join

73519. The LIMIT clause will terminate the loop for us

73520. Arguments for the indexing scheme

73521. Jump here when done, ex: LIMIT reached

73522. Convert TK_COLUMN nodes into TK_AGG_COLUMN and make entries in ** sAggInfo for all TK_AGG_FUNCTION nodes in expressions of the ** SELECT statement.

73523. ** Return the number of times sqlite3MemMalloc() has been called.

73524. Advance to the next row to process.

73525. Resolve the GROUP BY clause. At the same time, make sure ** the GROUP BY clause does not contain aggregate functions.

73526. OUT: Set to true if call is a no-op

73527. ***** End of vtab.c *****

73528. cmd ::= BEGIN transtype trans_opt

73529. ** Disconnect all sqlite3_vtab objects that belong to database connection ** db. This is called when db is being closed.

73530. Number of columns in this table

73531. A main database has just been opened. The following block sets ** (pFd->zWal) to point to a buffer owned by SQLite that contains ** the name of the *-wal file this db connection will use. SQLite ** happens to pass a pointer to this buffer when using xAccess() ** or xOpen() to operate on the *-wal file.

73532. 126

73533. ***** Begin file printf.c *****

73534. 850

73535. Name we are searching for

73536. Default tokenizer module

73537. SELECT rowid, * FROM ... WHERE rowid=?

73538. Want a pending lock?

73539. PERSIST

73540. cmd ::= PRAGMA nm dbnm LP nmnum RP

73541. Number of elements in apVal

73542. Add constraints to reduce the search space on a LIKE or GLOB ** operator. ** ** A like pattern of the form "x LIKE 'aBc%'" is changed into constraints ** ** x>='ABC' AND x<'abd' AND x LIKE 'aBc%' ** ** The last character of the prefix "abc" is incremented to form the ** termination condition "abd". If case is not significant (the default ** for LIKE) then the lower-bound is made all uppercase and the upper- ** bound is made all lowercase so that the bounds also work when comparing ** BLOBS.

73543. ** Helper functions to obtain and relinquish the global mutex. The ** global mutex is used to protect the unixInodeInfo and ** vxworksFileId objects used by this file, all of which may be ** shared by multiple threads. ** ** Function unixMutexHeld() is used to assert() that the global mutex ** is held when required. This function is only used as part of assert() ** statements. e.g. ** ** unixEnterMutex() ** assert(unixMutexHeld()), ** unixEnterLeave()

73544. Current operation

73545. Copy of aPgno[] used for sorting pages

73546. Temp storage must have already been allocated

73547. There must be at least one outstanding reference to the pager if ** in ERROR state. Otherwise the pager should have already dropped ** back to OPEN state.

73548. noop

73549. Flags passed through to VFS open

73550. **comment:** Memory for registers, parameters, cursor, etc, is allocated in one or two ** passes. On the first pass, we try to reuse unused memory at the ** end of the opcode array. If we are unable to satisfy all memory ** requirements by reusing the opcode array tail, then the second ** pass will fill in the remainder using a fresh memory allocation. ** ** This two-pass approach that reuses as much memory as possible from ** the leftover memory at the end of the opcode array. This can significantly ** reduce the amount of memory held by a prepared statement.

label: code-design

73551. Special values interpreted by sqlite3SegReaderCursor()

73552. ** Find the table that is to be indexed. Return early if not found.

73553. SQLITE_OMIT_PAGER_PRAGMAS

73554. ** Store the current database page-size in bytes in p->nPgsz. ** ** If *pRc is non-zero when this function is called, it is a no-op. ** Otherwise, if an error occurs, an SQLite error code is stored in *pRc ** before returning.

73555. ** pExpr is a comparison expression, e.g. '=', '<', IN(...) etc. ** idx_affinity is the affinity of an indexed column. Return true ** if the index with affinity idx_affinity may be used to implement ** the comparison in pExpr.

73556. Fake system time in seconds since 1970.

73557. OUT: Number of bytes at *paToken

73558. If isSimpleCount() returns a pointer to a Table structure, then ** the SQL statement is of the form: ** ** SELECT count(*) FROM <tbl> ** ** where the Table structure returned represents table <tbl>. ** ** This statement is so common that it is optimized specially. The ** OP_Count instruction is executed either on the intkey table that ** contains the data for table <tbl> or on one of its indexes. It ** is better to execute the op on an index, as indexes are almost ** always spread across less pages than their corresponding tables.

73559. BETWEEN

73560. Type of lock currently held on this file

73561. Date/time functions that use 'now', and other functions like ** sqlite_version() that might change over time cannot be used ** in an index.

73562. ** CAPI3REF: Attach A Table To A Session Object ** ** If argument zTab is not NULL, then it is the name of a table to attach ** to the session object passed as the first argument. All subsequent changes ** made to the table while the session object is enabled will be recorded. See ** documentation for [sqlite3session_changeset()] for further details. ** ** Or, if argument zTab is NULL, then changes are recorded for all tables ** in the database. If additional tables are added to the database (by ** executing "CREATE TABLE" statements) after this call is made, changes for ** the new tables are also recorded. ** ** Changes can only be recorded for tables that have a PRIMARY KEY explicitly ** defined as part of their CREATE TABLE statement. It does not matter if the ** PRIMARY KEY is an "INTEGER PRIMARY KEY" (rowid alias) or not. The PRIMARY ** KEY may consist of a single column, or may be a composite key. ** ** It is not an error if the named table does not exist in the database. Nor ** is it an error if the named table does not have a PRIMARY KEY. However, ** no changes will be recorded in either of these scenarios. ** ** Changes are not recorded for individual rows that have NULL values stored ** in one or more of their PRIMARY KEY columns. ** ** SQLITE_OK is returned if the call completes without error. Or, if an error ** occurs, an SQLite error code (e.g. SQLITE_NOMEM) is returned.

73563. IN/OUT: Right hand input list

73564. Appropriate locking method

73565. ** Lower the locking level on file descriptor pFile to eFileLock. eFileLock ** must be either NO_LOCK or SHARED_LOCK. ** ** If the locking level of the file descriptor is already at or below ** the requested locking level, this routine is a no-op. ** ** If handleNFSUnlock is true, then on downgrading an EXCLUSIVE_LOCK to SHARED ** the byte range is divided into 2 parts and the first part is unlocked then ** set to a read lock, then the other part is simply unlocked. This works ** around a bug in BSD NFS lockd (also seen on MacOSX 10.3+) that fails to ** remove the write lock on a region when a read lock is set.

73566. pgoffset

73567. If obtaining a child page for a cursor, we must verify that the page is ** compatible with the root page.

73568. 90

73569. pWriter has an exclusive lock

73570. ** If pCsr->pStmt has not been prepared (i.e. if pCsr->pStmt==0), then ** compose and prepare an SQL statement of the form: ** ** "SELECT <columns> FROM %_content WHERE rowid = ?" ** ** (or the equivalent for a content=xxx table) and set pCsr->pStmt to ** it. If an error occurs, return an SQLite error code.

73571. **comment:** As this "NEAR" object is actually a single phrase that consists ** of a single term only, grab pointers into the poslist managed by the ** fts5_index.c iterator object. This is much faster than synthesizing ** a new poslist the way we have to for more complicated phrase or NEAR ** expressions.

label: code-design

73572. ** Return a list of all triggers on table pTab if there exists at least ** one trigger that must be fired when an operation of type 'op' is ** performed on the table, and, if that operation is an UPDATE, if at ** least one of the columns in pChanges is being modified.

73573. ** This function returns the space in bytes required to store the copy ** of the Expr structure and a copy of the Expr.u.zToken string (if that ** string is defined.)

73574. Never happens because JNODE_RAW is only set by json_set(), ** json_insert() and json_replace() and those routines do not ** call jsonReturn()

73575. ** Decrement the ref-count on a virtual table object. When the ref-count ** reaches zero, call the xDisconnect() method to delete the object.

73576. ** The string z[] is an text representation of a real number. ** Convert this string to a double and write it into *pResult. ** ** The string z[] is length bytes in length (bytes, not characters) and ** uses the encoding enc. The string is not necessarily zero-terminated. ** ** Return TRUE if the result is a valid real number (or integer) and FALSE ** if the string is empty or contains extraneous text. Valid numbers ** are in one of these formats: ** ** [+/-]digits[E[+-]digits] ** [+/-]digits.[digits][E[+-]digits] ** [+/-].digits[E[+-]digits] ** ** Leading and trailing whitespace is ignored for the purpose of determining ** validity. ** ** If some prefix of the input string is a valid number, this routine ** returns FALSE but it still converts the prefix and writes the result ** into *pResult.

73577. Update the global lock state and do debug tracing

73578. Return TRUE if the jsonEachCursor object has been advanced off the end ** of the JSON object

73579. **comment:** We could allocate a fresh RowSetEntry each time one is needed, but it ** is more efficient to pull a preallocated entry from the pool
label: code-design

73580. Unqualified name of the index to create

73581. Move pPage to the front of the list

73582. Index of next entry to replace

73583. ** The second argument passed to this function is the name of a PRAGMA ** setting - "page_size", "auto_vacuum", "user_version" or "application_id". ** This function executes the following on sqlite3rbu.dbRbu: *** ** "PRAGMA main.\$zPragma" *** ** where \$zPragma is the string passed as the second argument, then ** on sqlite3rbu.dbMain: *** ** "PRAGMA main.\$zPragma = \$val" *** ** where \$val is the value returned by the first PRAGMA invocation. *** ** In short, it copies the value of the specified PRAGMA setting from ** dbRbu to dbMain.

73584. ***** End of insert.c *****

73585. Maximum number of PMAs that a single MergeEngine can merge

73586. ** Change the 'auto-vacuum' property of the database. If the 'autoVacuum' ** parameter is non-zero, then auto-vacuum mode is enabled. If zero, it ** is disabled. The default value for the auto-vacuum property is ** determined by the SQLITE_DEFAULT_AUTOVACUUM macro.

73587. expr ::= BITNOT expr

73588. ** Each memory allocation looks like this: ** ----- ** | Title | backtrace pointers | MemBlockHdr |
allocation | EndGuard | ** ----- ** | The application code sees only a pointer to the allocation. We have **
to back up from the allocation pointer to find the MemBlockHdr. The ** MemBlockHdr tells us the size of the allocation and the number of ** backtrace pointers.
There is also a guard word at the end of the ** MemBlockHdr.

73589. Value returned by GetLastError()

73590. The right operand

73591. multiselect_op ::= EXCEPT|INTERSECT

73592. Value from parent table row

73593. SQLITE_WAL_H

73594. Number of entries in this table

73595. 23

73596. ** Turn a SELECT statement (that the pSelect parameter points to) into ** a trigger step. Return a pointer to a TriggerStep structure. *** ** The parser calls this routine when it finds a SELECT statement in ** body of a TRIGGER.

73597. Score of parent node

73598. Generic error

73599. Set to true at EOF

73600. This function works in milliseconds, but the underlying OsSleep() ** API uses microseconds. Hence the 1000's.

73601. Initial aLTerm[] space

73602. ** Argument p points to a buffer containing a varint to be interpreted as a ** position list size field. Read the varint and return the number of bytes ** read. Before returning, set *pnSz to the number of bytes in the position ** list, and *pbDel to true if the delete flag is set, or false otherwise.

73603. Skip over == and IS and ISNULL terms. (Also skip IN terms when ** doing WHERE_ORDERBY_LIMIT processing). *** ** If the current term is a column of an ((?,?) IN (SELECT...)) ** expression for which the SELECT returns more than one column, ** check that it is the only column used by this loop. Otherwise, ** if it is one of two or more, none of the columns can be ** considered to match an ORDER BY term.

73604. Decode any entries that occur before the first term.

73605. Next MAIN_DB file

73606. Statistics

73607. The ORDER BY or GROUP BY clause to be processed

73608. The newly created page cache

73609. The following call to PagerSetPageSize() serves to set the value of ** Pager.pageSize and to allocate the Pager.pTmpSpace buffer.

73610. Hash table of in-memory nodes.

73611. A column in the result set. 0..pEList->nExpr-1

73612. IN/OUT: Offset within a[]

73613. ** Append the complete text of zero-terminated string z[] to the p string.

73614. **comment:** ** Cleanup before returning.
label: code-design

73615. cmd ::= with insert_cmd INTO fullname idlist_opt DEFAULT VALUES

73616. fillInCell() only called for leaves

73617. Used to iterate through entire index

73618. 9

73619. ** Macros indicating that conditional expressions are always true or ** false.

73620. Did we just have a reader lock?

73621. Load the next term on the node into zBuffer. Use realloc() to expand ** the size of zBuffer if required.

73622. **comment:** Next term to insert into %_idx table
label: code-design

73623. NO_TEST

73624. KeyInfo for index

73625. Index in pIn[] where zName is stored

73626. ** Set the StrAccum object to an error mode.

73627. ***** Begin file mutex_unix.c *****

73628. Output segment

73629. nil

73630. Check the following statements are true: *** ** (a) Exactly one of the READWRITE and READONLY flags must be set, and ** (b) if CREATE is set, then READWRITE must also be set, and ** (c) if EXCLUSIVE is set, then CREATE must also be set. ** (d) if DELETEONCLOSE is set, then CREATE must also be set.

73631. Cost of path (pFrom+pWLoop)

73632. Aggregate containing min() or max()

73633. Index structure

73634. ** CAPI3REF: SQL Trace Event Codes ** KEYWORDS: SQLITE_TRACE *** ** These constants identify classes of events that can be monitored ** using the [sqlite3_trace_v2()] tracing logic. The third argument ** to [sqlite3_trace_v2()] is an OR-ed combination of one or more of ** the following constants. ^The first argument to the trace callback ** is one of the following constants. *** ** New tracing constants may be added in future releases. *** ** ^A trace callback has four arguments: xCallback(T,C,P,X). ** ^The T argument is one of the integer type codes above. ** ^The C argument is a copy of the context pointer passed in as the ** fourth argument to [sqlite3_trace_v2()]. ** The P and X arguments are pointers whose meanings depend on T. *** ** <dl> ** [[SQLITE_TRACE_STMT]] <dt>SQLITE_TRACE_STMT</dt> ** <dd>^An SQLITE_TRACE_STMT callback is invoked when a prepared statement ** first begins running and possibly at other times during the ** execution of the prepared statement, such as at the start of each ** trigger subprogram. ^The P argument is a pointer to the ** [prepared statement]. ^The X argument is a pointer to a string which ** is the unexpanded SQL text of the prepared statement or an SQL comment ** that indicates the invocation of a trigger. ^The callback can compute ** the same text that would have been returned by the legacy [sqlite3_trace()] ** interface by using the X argument when X begins with "--" and invoking ** [sqlite3_expanded_sql(P)] otherwise. *** ** [[SQLITE_TRACE_PROFILE]] <dt>SQLITE_TRACE_PROFILE</dt> ** <dd>^An SQLITE_TRACE_PROFILE callback provides approximately the same ** information as is provided by the [sqlite3_profile()] callback. ** ^The P argument is a pointer to the [prepared statement] and the ** X argument points to a 64-bit integer which is the estimated of ** the number of nanosecond that the prepared statement took to run. ** ^The SQLITE_TRACE_PROFILE callback is invoked when the statement finishes. *** ** [[SQLITE_TRACE_ROW]] <dt>SQLITE_TRACE_ROW</dt> ** <dd>^An SQLITE_TRACE_ROW callback is invoked whenever a prepared ** statement generates a single row of result. ** ^The P argument is a pointer to the [prepared statement] and the ** X argument is unused. *** ** [[SQLITE_TRACE_CLOSE]] <dt>SQLITE_TRACE_CLOSE</dt> ** <dd>^An SQLITE_TRACE_CLOSE callback is invoked when a database ** connection closes. ** ^The P argument is a pointer to the [database connection] object ** and the X argument is unused. ** </dl>

73635. Copy the p->u.zToken string, if any.
73636. Instruction opcode
73637. %d or %u, but not %x, %o
73638. **comment:** Populate the array of registers beginning at regNew with the new ** row data. This array is used to check constants, create the new ** table and index records, and as the values for any new.* references ** made by triggers. *** If there are one or more BEFORE triggers, then do not populate the ** registers associated with columns that are (a) not modified by ** this UPDATE statement and (b) not accessed by new.* references. The ** values for registers not modified by the UPDATE must be reloaded from ** the database after the BEFORE triggers are fired anyway (as the trigger ** may have modified them). So not loading those that are not going to ** be used eliminates some redundant opcodes.
label: code-design
73639. Sync the journal file if required.
73640. Step 6: Loop through rows of the RHS. Compare each row to the LHS. ** If any comparison is NULL, then the result is NULL. If all ** comparisons are FALSE then the final result is FALSE. *** For a scalar LHS, it is sufficient to check just the first row ** of the RHS.
73641. Append the current configuration cookie
73642. ** Return a pointer to a string containing the 'declaration type' of the ** expression pExpr. The string may be treated as static by the caller. *** Also try to estimate the size of the returned value and return that ** result in *pEstWidth. *** The declaration type is the exact datatype definition extracted from the ** original CREATE TABLE statement if the expression is a column. The ** declaration type for a ROWID field is INTEGER. Exactly when an expression ** is considered a column can be complex in the presence of subqueries. The ** result-set expression in all of the following SELECT statements is ** considered a column by this function. *** SELECT col FROM tbl; ** SELECT (SELECT col FROM tbl; ** SELECT (SELECT col FROM tbl); ** SELECT abc FROM (SELECT col AS abc FROM tbl); *** The declaration type for any expression other than a column is NULL. *** This routine has either 3 or 6 parameters depending on whether or not ** the SQLITE_ENABLE_COLUMN_METADATA compile-time option is used.
73643. Current pre-update operation
73644. Instruction number of OP_Function opcode
73645. ** Implementation of the sqlite3_pcache.xRekey method.
73646. This is the PGroup.lru element
73647. Wal filename for this main db file
73648. Array of sub-vdbes
73649. Open the RBU database
73650. **comment:** ** Close a file & cleanup AFP specific locking context
label: code-design
73651. If lisPSample, the reason for inclusion
73652. The argument list
73653. ** Search along zPath to find the node specified. Return a pointer ** to that node, or NULL if zPath is malformed or if there is no such ** node. *** If pApnd!=0, then try to append new nodes to complete zPath if it is ** possible to do so and if no existing node corresponds to zPath. If ** new nodes are appended *pApnd is set to 1.
73654. eidlist_opt ::=
73655. One iterator for each phrase
73656. Final number of cells on page
73657. The aMx[] array translates the 3rd character of each format ** spec into a max size: a b c d e f
73658. ** Make sure at least one set of Win32 APIs is available.
73659. Allocated space not yet assigned
73660. ** An instance of the following structure is passed as the first ** argument to sqlite3VdbeKeyCompare and is used to control the ** comparison of the two index keys. *** Note that aSortOrder[] and aColl[] have nField+1 slots. There ** are nField slots for the columns of an index then one extra slot ** for the rowid at the end.
73661. 70
73662. ** Change the maximum number of in-memory pages that are allowed ** before attempting to spill pages to journal.
73663. _FTS3_TOKENIZER_H_
73664. Database filename UTF-8 encoded
73665. **comment:** ** CAPI3REF: Mutexes *** The SQLite core uses these routines for thread ** synchronization. Though they are intended for internal ** use by SQLite, code that links against SQLite is ** permitted to use any of these routines. *** The SQLite source code contains multiple implementations ** of these mutex routines. An appropriate implementation ** is selected automatically at compile-time. The following ** implementations are available in the SQLite core: *** ** SQLITE_MUTEX_PTHREADS ** SQLITE_MUTEX_W32 ** SQLITE_MUTEX_NOOP ** *** The SQLITE_MUTEX_NOOP implementation is a set of routines ** that does no real locking and is appropriate for use in ** a single-threaded application. The SQLITE_MUTEX_PTHREADS and ** SQLITE_MUTEX_W32 implementations are appropriate for use on Unix ** and Windows. *** If SQLite is compiled with the SQLITE_MUTEX_APPDEF preprocessor ** macro defined (with "-DSQLITE_MUTEX_APPDEF=1"), then no mutex ** implementation is included with the library. In this case the ** application must supply a custom mutex implementation using the ** [SQLITE_CONFIG_MUTEX] option of the sqlite3_config() function ** before calling sqlite3_initialize() or any other public sqlite3_ ** function that calls sqlite3_initialize(). *** ^The sqlite3_mutex_alloc() routine allocates a new ** mutex and returns a pointer to it. ^The sqlite3_mutex_alloc() ** routine returns NULL if it is unable to allocate the requested ** mutex. The argument to sqlite3_mutex_alloc() must one of these ** integer constants: *** ** SQLITE_MUTEX_FAST ** SQLITE_MUTEX_RECURSIVE ** SQLITE_MUTEX_STATIC_MASTER ** SQLITE_MUTEX_STATIC_MEM ** SQLITE_MUTEX_STATIC_OPEN ** SQLITE_MUTEX_STATIC_PRNG ** SQLITE_MUTEX_STATIC_LRU ** SQLITE_MUTEX_STATIC_PMEM ** SQLITE_MUTEX_STATIC_APP1 ** SQLITE_MUTEX_STATIC_APP2 ** SQLITE_MUTEX_STATIC_APP3 ** SQLITE_MUTEX_STATIC_VFS1 ** SQLITE_MUTEX_STATIC_VFS2 ** SQLITE_MUTEX_STATIC_VFS3 ** *** ^The first two constants (SQLITE_MUTEX_FAST and SQLITE_MUTEX_RECURSIVE) ** cause sqlite3_mutex_alloc() to create ** a new mutex. ^The new mutex is recursive when SQLITE_MUTEX_RECURSIVE ** is used but not necessarily so when SQLITE_MUTEX_FAST is used. ** The mutex implementation does not need to make a distinction ** between SQLITE_MUTEX_RECURSIVE and SQLITE_MUTEX_FAST if it does ** not want to. SQLite will only request a recursive mutex in ** cases where it really needs one. If a faster non-recursive mutex ** implementation is available on the host platform, the mutex subsystem ** might return such a mutex in response to SQLITE_MUTEX_FAST. *** ^The other allowed parameters to sqlite3_mutex_alloc() (anything other ** than SQLITE_MUTEX_FAST and SQLITE_MUTEX_RECURSIVE) each return ** a pointer to a static preexisting mutex. ^Nine static mutexes are ** used by the current version of SQLite. Future versions of SQLite ** may add additional static mutexes. Static mutexes are for internal ** use by SQLite only. Applications that use SQLite mutexes should ** use only the dynamic mutexes returned by SQLITE_MUTEX_FAST or ** SQLITE_MUTEX_RECURSIVE. *** ^Note that if one of the dynamic mutex parameters (SQLITE_MUTEX_FAST ** or SQLITE_MUTEX_RECURSIVE) is used then sqlite3_mutex_alloc() ** returns a different mutex on every call. ^For the static ** mutex types, the same mutex is returned on every call that has ** the same type number. *** ^The sqlite3_mutex_free() routine deallocates a previously ** allocated dynamic mutex. Attempting to deallocate a static ** mutex results in undefined behavior. *** ^The sqlite3_mutex_enter() and sqlite3_mutex_try() routines attempt ** to enter a mutex. ^If another thread is already within the mutex, ** sqlite3_mutex_enter() will block and sqlite3_mutex_try() will return ** SQLITE_BUSY. ^The sqlite3_mutex_try() interface returns [SQLITE_OK] ** upon successful entry. ^(Mutexes created using ** SQLITE_MUTEX_RECURSIVE can be entered multiple times by the same thread. ** In such cases, the ** mutex must be exited an equal number of times before another thread ** can enter.)^ If the same thread tries to enter any mutex other ** than an SQLITE_MUTEX_RECURSIVE more than once, the behavior is undefined. *** ^Some systems (for example, Windows 95) do not support the operation ** implemented by sqlite3_mutex_try(). On those systems, sqlite3_mutex_try() ** will always return SQLITE_BUSY. The SQLite core only ever uses ** sqlite3_mutex_try() as an optimization so this is acceptable ** behavior.)*** ^The sqlite3_mutex_leave() routine exits a mutex that was ** previously entered by the same thread. The behavior ** is undefined if the mutex is not currently entered by the ** calling thread or is not currently allocated. *** ^If the argument to sqlite3_mutex_enter(), sqlite3_mutex_try(), or ** sqlite3_mutex_leave() is a NULL pointer, then all three routines ** behave as no-ops. *** See also: [sqlite3_mutex_held()] and [sqlite3_mutex_noheld()].
label: code-design
73666. ** If SQLITE_OMIT_AUTOINCREMENT is defined, then the three routines ** above are all no-ops
73667. Name of the module
73668. ** Initialize the auxiliary information for a disk block. ** ** Return SQLITE_OK on success. If we see that the page does ** not contain a well-formed database page, then return ** SQLITE_CORRUPT. Note that a return of SQLITE_OK does not ** guarantee that the page is well-formed. It only shows that ** we failed to

detect any corruption.

73669. Advance to the next output block

73670. ** Called when iAmt bytes are read from offset iOff of the wal file while ** the rbu object is in capture mode. Record the frame number of the frame ** being read in the aFrame[] array.

73671. If p5 is zero, the seek operation that positioned the cursor prior to ** OP_Delete will have also set the pC->movetoTarget field to the rowid of ** the row that is being deleted

73672. Frame to read from WAL file

73673. ** Helper function for fts3NodeWrite().

73674. If the iterator is not at a real match, skip forward until it is.

73675. Page number to obtain

73676. If there was an INDEXED BY clause, then only that one index is ** considered.

73677. If we reach this point, that means the transformation is required.

73678. ** Return a nul-terminated copy of the string indicated by pln. If nIn ** is non-negative, then it is the length of the string in bytes. Otherwise, ** the length of the string is determined using strlen(). ** ** It is the responsibility of the caller to eventually free the returned ** buffer using sqlite3_free(). If an OOM error occurs, NULL is returned.

73679. This branch is taken when the journal path required by ** the database being opened will be more than pVfs->mxPathname ** bytes in length. This means the database cannot be opened, ** as it will not be possible to open the journal file or even ** check for a hot-journal before reading.

73680. 195

73681. xSync - sync transaction

73682. Pseudocode used to decode sorting results

73683. If this is the right table of a LEFT OUTER JOIN, allocate and ** initialize a memory cell that records if this table matches any ** row of the left table of the join.

73684. **comment:** Number of contiguous term-less nodes

label: code-design

73685. Link an element into the hash table

73686. The VM under construction

73687. Table object

73688. Address register for the output-A subroutine

73689. vdbePmaReadBlob() return code

73690. Old index in b.apCell[]

73691. groupby_opt

73692. Write the xxx_node table

73693. The RHS of the IN operator

73694. Various SF_* values

73695. Used to iterate through cells

73696. Populate the PagerSavepoint structures just allocated.

73697. FTS5 Configuration object

73698. Figure out how many bytes are required by this new entry

73699. Skip accumulator loading if true

73700. Expression to initialize phrases in

73701. Number of bytes of leaf data written

73702. True if non-integer value was input to the sum

73703. Pathname of file to be opened

73704. Pointer to tail

73705. Zero the output variables in case an error occurs.

73706. Pointer to cell to delete

73707. ** 2014 Jun 09 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This is an SQLite module implementing full-text search.

73708. Conflict only if the rowid of the existing index entry ** is different from old-rowid

73709. xOpen

73710. If nLoop is zero, then there are no FROM terms in the query. Since ** in this case the query may return a maximum of one row, the results ** are already in the requested order. Set isOrdered to nOrderBy to ** indicate this. Or, if nLoop is greater than zero, set isOrdered to ** -1, indicating that the result set may or may not be ordered, ** depending on the loops added to the current plan.

73711. IN/OUT: List to sort

73712. ** Implementation of the xBestIndex method for FTS5 tables. Within the ** WHERE constraint, it searches for the following: ** ** 1. A MATCH constraint against the special column. ** 2. A MATCH constraint against the "rank" column. ** 3. An == constraint against the rowid column. ** 4. A < or <= constraint against the rowid column. ** 5. A > or >= constraint against the rowid column. ** ** Within the ORDER BY, either: ** ** 5. ORDER BY rank [ASC|DESC] ** 6. ORDER BY rowid [ASC|DESC] ** ** Costs are assigned as follows: ** ** a) If an unusable MATCH operator is present in the WHERE clause, the ** cost is unconditionally set to 1e50 (a really big number). ** ** a) If a MATCH operator is present, the cost depends on the other ** constraints also present. As follows:
** ** * No other constraints: cost=1000.0 ** * One rowid range constraint: cost=750.0 ** * Both rowid range constraints: cost=500.0 ** * An == rowid constraint: cost=100.0 ** ** b) Otherwise, if there is no MATCH. ** ** * No other constraints: cost=1000000.0 ** * One rowid range constraint: cost=750000.0 ** * Both rowid range constraints: cost=250000.0 ** * An == rowid constraint: cost=10.0 ** ** Costs are not modified by the ORDER BY clause.

73713. ** Print into memory obtained from sqliteMalloc(). Use the internal ** %-conversion extensions.

73714. in hash, if not, try to find a spot for it

73715. Path by which to filter cJSON

73716. Value of pOp at the top of the loop

73717. Two or more indexes share the same root page. There must ** be imposter tables. So just return true. The assert is not ** useful in that case.

73718. ** Generate code for a DELETE FROM statement. ** ** DELETE FROM table_wxyz WHERE a<5 AND b NOT NULL; ** ___/___ / ** pTabList pWhere

73719. ** CAPI3REF: Test For Auto-Commit Mode ** KEYWORDS: {autocommit mode} ** METHOD: sqlite3 ** ** ^The sqlite3_get_autocommit() interface returns non-zero or ** zero if the given database connection is or is not in autocommit mode, ** respectively. ^Autocommit mode is on by default. ** ^Autocommit mode is disabled by a [BEGIN] statement. ** ^Autocommit mode is re-enabled by a [COMMIT] or [ROLLBACK]. ** ** If certain kinds of errors occur on a statement within a multi-statement ** transaction (errors including [SQLITE_FULL], [SQLITE_IOERR], ** [SQLITE_NOMEM], [SQLITE_BUSY], and [SQLITE_INTERRUPT]) then the ** transaction might be rolled back automatically. The only way to ** find out whether SQLite automatically rolled back the transaction after ** an error is to use this function. ** ** If another thread changes the autocommit status of the database ** connection while this routine is running, then the return value ** is undefined.

73720. ** Merge the entire database so that there is one segment for each ** iIndex/iLangid combination.

73721. Free up as much memory as possible from the page cache

73722. #include "fts3.h"

73723. True for reverse-order IN operations

73724. Phrase to advance token of

73725. OP_Open should use BTREE_FORDELETE

73726. Opcode: OpenEphemeral P1 P2 * P4 P5 ** Synopsis: nColumn=P2 ** ** Open a new cursor P1 to a transient table. ** The cursor is always opened read/write even if ** the main database is read-only. The ephemeral ** table is deleted automatically when the cursor is closed. ** ** P2 is the number of columns in the ephemeral table. ** The cursor points to a BTree table if P4==0 and to a BTree index ** if P4 is not 0. If P4 is not NULL, it points to a KeyInfo structure ** that defines the format of keys in the index. ** ** The P5 parameter can be a mask of the BTREE_* flags defined ** in btree.h. These flags control aspects of the operation of ** the btree. The BTREE OMIT_JOURNAL and BTREE SINGLE flags are ** added automatically.

73727. True if the conversion to IN is valid

73728. OUT: Acquired page object
73729. **comment:** Initializing the tail of the buffer is not necessary. Everything ** works fine if the following `memset()` is omitted. But initializing ** the memory prevents valgrind from complaining, so we are willing to ** take the performance hit.
label: code-design

73730. An OE_ code for handling a NOT NULL constraint
73731. Length of the keyword in characters
73732. expr ::= expr STAR|SLASH|REM expr
73733. If recursive mutexes are not available, we will have to ** build our own. See below.
73734. ***** Continuing where we left off in `status.c` *****
73735. 18
73736. A VList object records a mapping between parameters/variables/wildcards ** in the SQL statement (such as \$abc, @pqr, or :xyz) and the integer ** variable number associated with that parameter. See the format description ** on the `sqlite3VListAdd()` routine for more information. A VList is really ** just an array of integers.
73737. Fail the operation and rollback the transaction
73738. White space is ignored
73739. This sample is being added because the prefix that ends in column ** iCol occurs many times in the table. However, if we have already ** added a sample that shares this prefix, there is no need to add ** this one. Instead, upgrade the priority of the highest priority ** existing sample that shares this prefix.
73740. aSalt[0] is a copy of the value stored in the wal file header. It ** is incremented each time the wal file is restarted.
73741. Largest key seen since xTruncate()
73742. Vdbe to iterate through the opcodes of
73743. Convenient short-hand
73744. ** This function is the implementation of the xUpdate callback used by ** FTS3 virtual tables. It is invoked by SQLite each time a row is to be ** inserted, updated or deleted. *** A delete specifies a single argument - the rowid of the row to remove. *** Update and insert operations pass: *** 1. The "old" rowid, or NULL. ** 2. The "new" rowid. ** 3. Values for each of the nCol matchable columns. ** 4. Values for the two hidden columns (<tablename> and "rank").
73745. Assign argvIndex values to each constraint in use.
73746. The structure record
73747. boolean
73748. ** This "finder" function attempts to determine the best locking strategy ** for the database file "filePath". It then returns the `sqlite3_io_methods` ** object that implements that strategy. *** This is for Mac OSX only.
73749. ON CONFLICT policy to code trigger program with
73750. where_opt
73751. ** This function is a no-op unless `SQlite_DEBUG` is defined when this module ** is compiled. In that case, this function is essentially an assert() ** statement used to verify that the contents of the pIter->aFirst[] array ** are correct.
73752. ** An ephemeral string value (signified by the `MEM_Ephem` flag) contains ** a pointer to a dynamically allocated string where some other entity ** is responsible for deallocating that string. Because the register ** does not control the string, it might be deleted without the register ** knowing it. *** This routine converts an ephemeral string into a dynamically allocated ** string that the register itself controls. In other words, it ** converts an `MEM_Ephem` string into a string with `Pz==P.zMalloc`.
73753. This block sets stack variable iPg to the leaf page number that may ** contain term (pTerm/nTerm), if it is present in the segment.
73754. Unsigned loop counter
73755. Linked list of all unfreed memory
73756. Return code from execution
73757. ** This function is called when inserting, deleting or updating a row of ** table pTab to generate VDBE code to perform foreign key constraint ** processing for the operation. *** For a DELETE operation, parameter regOld is passed the index of the ** first register in an array of (`pTab->nCol+1`) registers containing the ** rowid of the row being deleted, followed by each of the column values ** of the row being deleted, from left to right. Parameter regNew is passed ** zero in this case. *** For an INSERT operation, regOld is passed zero and regNew is passed the ** first register of an array of (`pTab->nCol+1`) registers containing the new ** row data. *** For an UPDATE operation, this function is called twice. Once before ** the original record is deleted from the table using the calling convention ** described for DELETE. Then again after the original record is deleted ** but before the new record is inserted using the INSERT convention.
73758. Both x>EXPR and x<EXPR
73759. function to free pUser
73760. Left operand
73761. log(Number of iterations due to IN)
73762. The database connection;
73763. Position list for deferred tokens
73764. Return true if the cursor was opened using the OP_OpenSorter opcode.
73765. When floating-point is omitted, double and int64 are the same thing
73766. Flags used only if debugging
73767. Value returned by SetFilePointerEx()
73768. ** Load the contents of the %_config table into memory.
73769. **comment:** *** CAPI3REF: Configuring The SQLite Library *** The `sqlite3_config()` interface is used to make global configuration ** changes to SQLite in order to tune SQLite to the specific needs of ** the application. The default configuration is recommended for most ** applications and so this routine is usually not necessary. It is ** provided to support rare applications with unusual needs. *** The `sqlite3_config()` interface is not threadsafe. The application ** must ensure that no other SQLite interfaces are invoked by other ** threads while `sqlite3_config()` is running. *** The `sqlite3_config()` interface ** may only be invoked prior to library initialization using ** `[sqlite3_initialize()]` or after shutdown by `[sqlite3_shutdown()]`. ** ^If `sqlite3_config()` is called after `[sqlite3_initialize()]` and before ** `[sqlite3_shutdown()]` then it will return `SQlite_MISUSE`. ** Note, however, that `^sqlite3_config()` can be called as part of the ** implementation of an application-defined `[sqlite3_os_init()]`. *** The first argument to `sqlite3_config()` is an integer ** [configuration option] that determines ** what property of SQLite is to be configured. Subsequent arguments ** vary depending on the [configuration option] ** in the first argument. *** ^When a configuration option is set, `sqlite3_config()` returns `[SQLITE_OK]`. ** ^If the option is unknown or SQLite is unable to set the option ** then this routine returns a non-zero [error code].
label: code-design
73770. If this is an auto-vacuum database, update the pointer-map entries ** for any b-tree or overflow pages that pTo now contains the pointers to.
73771. Here code is inserted which will execute if the parser ** stack every overflows
73772. `sqlite3_test_control(SQLITE_TESTCTRL_SORTER_MMAP, db, nMax);`
73773. Statements created by `rdbObjIterPrepareAll()`
73774. Map cursor numbers to bitmasks
73775. ** pStmt: ** SELECT rowid, <fts> FROM <fts> ORDER BY +rank; ** aIdx[]: ** There is one entry in the aIdx[] array for each phrase in the query, ** the value of which is the offset within aPoslist[] following the last ** byte of the position list for the corresponding phrase.
73776. Compute in mxSafeFrame the index of the last frame of the WAL that is ** safe to write into the database. Frames beyond mxSafeFrame might ** overwrite database pages that are in use by active readers and thus ** cannot be backfilled from the WAL.
73777. A negative cache-size value C indicates that the cache is $\text{abs}(C)$ ** KiB in size.
73778. This function normally generates a nested loop for all tables in ** pTabList. But if the WHERE_OR_SUBCLAUSE flag is set, then we should ** only generate code for the first table in pTabList and assume that ** any cursors associated with subsequent tables are uninitialized.
73779. Unable to flatten compound queries
73780. Create the destination temporary table if necessary
73781. ** Expression p should encode a floating point value between 1.0 and 0.0. ** Return 1024 times this value. Or return -1 if p is not a floating point ** value between 1.0 and 0.0.
73782. One particular FROM clause in a nested query
73783. To iterate through levels
73784. All keys from this input segment have been transferred to the output. ** Set both the first and last page-numbers to 0 to indicate that the ** segment is now empty.

73785. IMP: R-51414-32910
73786. 59
73787. ***** End of main.c *****
73788. SQLITE_NULLEQ or zero
73789. Number of attached databases
73790. Deferred search tokens, if any
73791. **comment:** ** This version of the memory allocator is used only if the ** SQLITE_MEMDEBUG macro is defined
label: code-design
73792. Opcode: ElseNotEq * P2 * * * * * This opcode must immediately follow an OP_Lt or OP_Gt comparison operator. ** If result of an OP_Eq comparison on the same two operands ** would have been NULL or false (0), then jump to P2. ** If the result of an OP_Eq comparison on the two previous operands ** would have been true (1), then fall through.
73793. SQLITE_TEST
73794. Load the built-in tokenizers into the hash table
73795. ** Register a new 2nd-generation geometry function for use with the ** r-tree MATCH operator.
73796. Allocate the SegmentWriter structure
73797. Postlist of this iterator
73798. In this case, call balance_nonroot() to redistribute cells ** between pPage and up to 2 of its sibling pages. This involves ** modifying the contents of pParent, which may cause pParent to ** become overfull or underfull. The next iteration of the do-loop ** will balance the parent page to correct this. ** * * If the parent page becomes overfull, the overflow cell or cells ** are stored in the pSpace buffer allocated immediately below. ** A subsequent iteration of the do-loop will deal with this by ** calling balance_nonroot() (balance_deeper() may be called first, ** but it doesn't deal with overflow cells - just moves them to a ** different page). Once this subsequent call to balance_nonroot() ** has completed, it is safe to release the pSpace buffer used by ** the previous call, as the overflow cell data will have been ** copied either into the body of a database page or into the new ** pSpace buffer passed to the latter call to balance_nonroot().
73799. Flags to enable/disable optimizations
73800. Term.pExpr is an IS operator
73801. Allocate and populate the new session object.
73802. File from which to do direct overflow read
73803. ** Transfer the contents of pFrom to pTo. Any existing value in pTo is ** freed. If pFrom contains ephemeral data, a copy is made. ** * * pFrom contains an SQL NULL when this routine returns.
73804. New change object
73805. If no test above fails then the indices must be compatible
73806. Do not analyze the TEMP database
73807. Integer and Real
73808. szPma
73809. Address of array containing old row
73810. ** CAPI3REF: Conflict resolution modes ** KEYWORDS: {conflict resolution mode} ** * * These constants are returned by [sqlite3_vtab_on_conflict()] to ** inform a [virtual table] implementation what the [ON CONFLICT] mode ** is for the SQL statement being evaluated. ** * * Note that the [SQLITE_IGNORE] constant is also used as a potential ** return value from the [sqlite3_set_authorizer()] callback and that ** [SQLITE_ABORT] is also a [result code].
73811. ** Implementation of matchinfo() function.
73812. User name used to authenticate
73813. There exists another row with the new.* primary key.
73814. (5)
73815. Tables that are indexable, satisfying case 2
73816. True if this constraint is usable
73817. Index identifier string
73818. Take the jump if the BETWEEN is NULL
73819. If REPLACE conflict resolution might be invoked, open cursors on all ** indexes in case they are needed to delete records.
73820. cmd ::= VACUUM
73821. If the column contains an "AS <name>" phrase, use <name> as the name
73822. 98..9f,
73823. Try to expand the master using the newly freed chunk
73824. ** Implementation of the stat_push SQL function: stat_push(P,C,R) ** Arguments: ** * * P Pointer to the Stat4Accum object created by stat_init() ** C Index of left-most column to differ from previous row ** R Rowid for the current row. Might be a key record for ** WITHOUT ROWID tables. ** * * This SQL function always returns NULL. It's purpose is to accumulate ** statistical data and/or samples in the Stat4Accum object about the ** index being analyzed. The stat_get() SQL function will later be used to ** extract relevant information for constructing the sqlite_statN tables. ** * * The R parameter is only used for STAT3 and STAT4
73825. ***** End of sqliteLimit.h *****
73826. **comment:** ** 2007 August 28 **** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** * * May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** * * * * * This file contains the common header for all mutex implementations. ** The sqliteInt.h header #includes this file so that it is available ** to all source files. We break it out in an effort to keep the code ** better organized. ** * * NOTE: source files should *not* #include this header file directly. ** Source files should #include the sqliteInt.h file and let that file ** include this one indirectly.
label: code-design
73827. The cell may contain a pointer to an overflow page. If so, write ** the entry for the overflow page into the pointer map.
73828. INSTEAD
73829. Initialize each of the component segment iterators.
73830. Bytes of space at aAlloc
73831. First argument to the function
73832. OUT: Blob data in malloc'd buffer
73833. Possible values for Rtree.eCoordType:
73834. Total number of columns in the index
73835. If an error has already occurred, this is a no-op
73836. Now manipulate the actual database free-list structure. There are two ** possibilities. If the free-list is currently empty, or if the first ** trunk page in the free-list is full, then this page will become a ** new free-list trunk page. Otherwise, it will become a leaf of the ** first trunk page in the current free-list. This block tests if it ** is possible to add the page as a new free-list leaf.
73837. At least this one VM is active
73838. ** Return a pointer to the WalIndexHdr structure in the wal-index.
73839. Begin generating code.
73840. **comment:** ** The sqlite3_mutex_leave() routine exits a mutex that was previously ** entered by the same thread. The behavior is undefined if the mutex ** is not currently entered. If a NULL pointer is passed as an argument ** this function is a no-op.
label: code-design
73841. Make sure the locking sequence is correct. ** (1) We never move from unlocked to anything higher than shared lock. ** (2) SQLite never explicitly requests a pending lock. ** (3) A shared lock is always held when a reserve lock is requested.
73842. Count non-delimiter characters.
73843. Number of levels in this index
73844. The index expression
73845. Set the szLeaf field
73846. Index used, or NULL

73847. ** An instance of this object represents a single database file. *** A single database file can be in use at the same time by two ** or more database connections. When two or more connections are ** sharing the same database file, each connection has its own ** private Btree object for the file and each of those Btrees points ** to this one BtShared object. BtShared.nRef is the number of ** connections currently sharing this database file. *** Fields in this structure are accessed under the BtShared.mutex ** mutex, except for nRef and pNext which are accessed under the ** global SQLITE_MUTEX_STATIC_MASTER mutex. The pPager field ** may not be modified once it is initially set as long as nRef>0. ** The pSchema field may be set once under BtShared.mutex and ** thereafter is unchanged as long as nRef>0. *** isPending: ** If a BtShared client fails to obtain a write-lock on a database ** table (because there exists one or more read-locks on the table), ** the shared-cache enters 'pending-lock' state and isPending is ** set to true. *** The shared-cache leaves the 'pending lock' state when either of ** the following occur: *** 1) The current writer (BtShared.pWriter) concludes its transaction, OR ** 2) The number of locks held by other connections drops to zero. *** while in the 'pending-lock' state, no connection may start a new ** transaction. *** This feature is included to help prevent writer-starvation.

73848. **comment:** An unusual case: this is the first term to be added to the node ** and the static node buffer (p->nNodeSize bytes) is not large ** enough. Use a separately malloced buffer instead. This wastes ** p->nNodeSize bytes, but since this scenario only comes about when ** the database contain two terms that share a prefix of almost 2KB, ** this is not expected to be a serious problem.

label: code-design

73849. ** Return the 64-bit integer value associated with cell iCell of ** node pNode. If pNode is a leaf node, this is a rowid. If it is ** an internal node, then the 64-bit integer is a child page number.

73850. Number of columns in record

73851. TODO1: Updating pgidx here.

73852. Total sizes of each column

73853. IN/OUT: Left-hand buffer pointer

73854. ABOVE: Height enforcement enabled. BELOW: Height enforcement off

73855. transtype ::=

73856. Number of identifiers on the list

73857. Phrase object

73858. Bytes of data stored in pFd

73859. First in list of all open cursors

73860. Leaf to add to tree

73861. ** Find the mode, uid and gid of file zFile.

73862. Minimum local payload in non-LEAFDATA tables

73863. Pointer to phrase counter

73864. SubProgram.nMem is set to the number of memory cells used by the ** program stored in SubProgram.aOp. As well as these, one memory ** cell is required for each cursor used by the program. Set local ** variable nMem (and later, VdbeFrame.nChildMem) to this value.

73865. ** Return TRUE if the given expression is a constant which would be ** unchanged by OP_Affinity with the affinity given in the second ** argument. *** This routine is used to determine if the OP_Affinity operation ** can be omitted. When in doubt return FALSE. A false negative ** is harmless. A false positive, however, can result in the wrong ** answer.

73866. Where to jump to continue with the next IN case

73867. **comment:** ** If the third argument is non-NULL, then this function releases a ** reference obtained by an earlier call to unixFetch(). The second ** argument passed to this function must be the same as the corresponding ** argument that was passed to the unixFetch() invocation. *** Or, if the third argument is NULL, then this function is being called ** to inform the VFS layer that, according to POSIX, any existing mapping ** may now be invalid and should be unmapped.

label: code-design

73868. ***** Begin file hash.h *****

73869. True to count the table

73870. SQLITE_STMTSTATUS_RUN

73871. ** Check to make sure the given table is writable. If it is not ** writable, generate an error message and return 1. If it is ** writable return 0;

73872. flags fit in 8 bits

73873. Input flags to control the opening

73874. ** Delete memory allocated for the column names of a table or view (the ** Table.aCol[] array).

73875. !SQLITE_OMIT_WAL

73876. Invoke xProgress() when nVmStep reaches this

73877. Term to seek to (or NULL/0)

73878. 270

73879. EVIDENCE-OF: R-43316-37308 A value of 2 (0x02) means the page is an ** interior index b-tree page.

73880. Original value of pNew->prereq

73881. Opening flags

73882. Generate code that will continue to the next row if ** the IN constraint is not satisfied

73883. The B*Tree structure for this database file

73884. The WHEN clause of the expression (may be NULL)

73885. ***** Begin file fts3_unicode.c *****

73886. Disable nuisance warnings on Borland compilers

73887. Used to loop through N hash tables

73888. RHS is entirely deferred. So we assume it matches every row. ** Advance the LHS iterator to find the next row visited.

73889. SQLITE_WIN32_FILEMAPPING_API

73890. Name of the database file to open

73891. **comment:** If the HAS_MOVED file-control is unimplemented, assume that the file ** has not been moved. That is the historical behavior of SQLite: prior to ** version 3.8.3, it never checked

label: code-design

73892. ** Begin iterating through the set of documents in index pIdx matched by ** the MATCH expression passed as the first argument. If the "bDesc" ** parameter is passed a non-zero value, iteration is in descending rowid ** order. Or, if it is zero, in ascending order. *** If iterating in ascending rowid order (bDesc==0), the first document ** visited is that with the smallest rowid that is larger than or equal ** to parameter iFirst. Or, if iterating in ascending order (bDesc==1), ** then the first document visited must have a rowid smaller than or ** equal to iFirst. *** Return SQLITE_OK if successful, or an SQLite error code otherwise. It ** is not considered an error if the query does not match any documents.

73893. Offset of db file entry

73894. ** If the node is dirty, write it out to the database.

73895. If TEMP was specified, then the trigger name may not be qualified.

73896. If connected to a ZIPVFS backend, override the page size and ** offset with actual values obtained from ZIPVFS.

73897. Read and write return code from here

73898. 1 sec timeout

73899. Pointer to TK_STRING expression with pattern prefix

73900. ** Expression pExpr is a comparison between two vector values. Compute ** the result of the comparison (1, 0, or NULL) and write that ** result into register dest. *** The caller must satisfy the following preconditions: ** ** if pExpr->op==TK_IS: op==TK_EQ and p5==SQLITE_NULLEQ ** if pExpr->op==TK_ISNOT: op==TK_NE and p5==SQLITE_NULLEQ ** otherwise: op==pExpr->op and p5==0

73901. ** Copy as much content as we can from the WAL back into the database file ** in response to an sqlite3_wal_checkpoint() request or the equivalent. *** The amount of information copies from WAL to database might be limited ** by active readers. This routine will never overwrite a database page ** that a concurrent reader might be using. *** All I/O barrier operations (a.k.a fsyncs) occur in this routine when ** SQLite is in WAL-mode in synchronous=NORMAL. That means that if ** checkpoints are always run by a background thread or background ** process, foreground threads will never block on a lengthy fsync call. *** Fsync is called on the WAL before writing content out of the WAL and ** into the database. This ensures that if the new content is persistent ** in the WAL and can be recovered following a power-loss or hard reset. *** Fsync is also called on the database file if (and only if) the entire ** WAL content is copied into the database file. This second fsync makes ** it safe to delete the WAL since the new content will persist in the ** database file. *** This routine uses and updates the nBackfill field of the wal-index header. ** This is the only routine that will increase the value of nBackfill. ** (A WAL reset or recovery will revert nBackfill

to zero, but not increase ** its value.) *** The caller must be holding sufficient locks to ensure that no other ** checkpoint is running (in any other thread or process) at the same ** time.

73902. AFTER => ID

73903. If this is an UPDATE that is part of a patchset, then all PK and ** modified fields are present in the new.* record. The old.* record ** is currently completely empty. This block shifts the PK fields from ** new.* to old.*, to accommodate the code that reads these arrays.

73904. One of SQLITE_UPDATE, INSERT, DELETE

73905. 120

73906. **comment:** ** CAPI3REF: Name Of The Folder Holding Database Files *** ^{If this global variable is made to point to a string which is ** the name of a folder (a.k.a. directory), then all database files ** specified with a relative pathname and created or accessed by ** SQLite when using a built-in windows [sqlite3_vfs | VFS] will be assumed ** to be relative to that directory.}^ If this variable is a NULL ** pointer, then SQLite assumes that all database files specified ** with a relative pathname are relative to the current directory ** for the process. Only the windows VFS makes use of this global ** variable; it is ignored by the unix VFS. *** Changing the value of this variable while a database connection is ** open can result in a corrupt database. *** It is not safe to read or modify this variable in more than one ** thread at a time. It is not safe to read or modify this variable ** if a [database connection] is being used at the same time in a separate ** thread. ** It is intended that this variable be set once ** as part of process initialization and before any SQLite interface ** routines have been called and that this variable remain unchanged ** thereafter. *** ^{The [data_store_directory pragma] may modify this variable and cause ** it to point to memory obtained from [sqlite3_malloc]. Furthermore, ** the [data_store_directory pragma] always assumes that any string ** that this variable points to is held in memory obtained from ** [sqlite3_malloc] and the pragma may attempt to free that memory ** using [sqlite3_free].}** Hence, if this variable is modified directly, either it should be ** made NULL or made to point to memory obtained from [sqlite3_malloc] ** or else the use of the [data_store_directory pragma] should be avoided.

label: code-design

73907. Optimal text encoding

73908. Allocate and populate the Fts3Table structure.

73909. If the destination database has not yet been locked (i.e. if this ** is the first call to backup_step() for the current backup operation), ** try to set its page size to the same as the source database. This ** is especially important on ZipVFS systems, as in that case it is ** not possible to create a database file that uses one page size by ** writing to it with another.

73910. ** Hash and comparison functions when the mode is FTS3_HASH_BINARY

73911. Number of entries in aConstraint

73912. didn't get, must be busy

73913. Label for the end of the overall SELECT stmt

73914. The whole database

73915. New table number

73916. Callback for expressions

73917. ** Return the number of times the Step function of an aggregate has been ** called. *** This function is deprecated. Do not use it for new code. It is ** provide only to avoid breaking legacy code. New aggregate function ** implementations should keep their own counts within their aggregate ** context.

73918. Write the folded case of the last character read to the output

73919. Store only the name

73920. ***** The following block contains those class members that change during ** routine operation. Class members not in this block are either fixed ** when the pager is first created or else only change when there is a ** significant mode change (such as changing the page_size, locking_mode, ** or the journal_mode). From another view, these class members describe ** the "state" of the pager, while other class members describe the ** "configuration" of the pager.

73921. Collating sequence for each term of the key

73922. only mkdir if leaf dir != "." or "/" or ".."

73923. 0 if leaf==1. 4 if leaf==0

73924. ***** Continuing where we left off in vdbe.c *****

73925. Tree contains a TK_SELECT operator

73926. ** Formulate and prepare a statement to UPDATE a row from database db. ** Assuming a table structure like this: *** CREATE TABLE x(a, b, c, d, PRIMARY KEY(a, c)); *** The UPDATE statement looks like this: *** UPDATE x SET ** a = CASE WHEN ?2 THEN ?3 ELSE a END, ** b = CASE WHEN ?5 THEN ?6 ELSE b END, ** c = CASE WHEN ?8 THEN ?9 ELSE c END, ** d = CASE WHEN ?11 THEN ?12 ELSE d END ** WHERE a = ?1 AND c = ?7 AND (?13 OR ** (?5==0 OR b IS ?4) AND (?11==0 OR d IS ?10) AND **) *** For each column in the table, there are three variables to bind: *** ?(i*3+1) The old.* value of the column, if any. ** ?(i*3+2) A boolean flag indicating that the value is being modified. ** ?(i*3+3) The new.* value of the column, if any. ** Also, a boolean flag that, if set to true, causes the statement to update ** a row even if the non-PK values do not match. This is required if the ** conflict-handler is invoked with CHANGESET_DATA and returns ** CHANGESET_REPLACE. This is variable "?nCol*3+1)". *** If successful, SQLITE_OK is returned and SessionApplyCtx.pUpdate is left ** pointing to the prepared version of the SQL statement.

73927. SQLITEINT_H

73928. OUT: Imposter PK clause

73929. ** The diff hook implementations.

73930. True if getNextNode() sees a unary -

73931. compiler-defined memory barrier

73932. cap min request size at 2^12

73933. An index of the table

73934. Return the number of cells in a node

73935. The rest are extensions, not normally found in printf()

73936. Size of database content section

73937. An expression from the ORDER BY clause

73938. Extension API function pointers.

73939. Value returned by preupdate_new/old

73940. If iCurrentBlock>=iLeafEndBlock, this is an EOF condition. All leaf ** blocks have already been traversed.

73941. If the SQLITE_TESTCTRL_FAULT_INSTALL callback is registered to a ** function that returns SQLITE_ERROR when passed the argument 200, that ** forces worker threads to run sequentially and deterministically ** for testing purposes.

73942. Right operand

73943. ** Return 1 if database is read-only or 0 if read/write. Return -1 if ** no such database exists.

73944. ** Mark this cursor as an incremental blob cursor.

73945. Opcode: MaxPgcnt P1 P2 P3 * * * * Try to set the maximum page count for database P1 to the value in P3. ** Do not let the maximum page count fall below the current page count and ** do not change the maximum page count value if P3==0. ** * Store the maximum page count after the change in register P2.

73946. Invoke MustBeInt to coerce the child key value to an integer (i.e. ** apply the affinity of the parent key). If this fails, then there ** is no matching parent key. Before using MustBeInt, make a copy of ** the value. Otherwise, the value inserted into the child key column ** will have INTEGER affinity applied to it, which may not be correct.

73947. PRIVATE: Cache that owns this page

73948. File handle object

73949. ** Find the current time (in Universal Coordinated Time). Write the ** current time and date as a Julian Day number into *prNow and ** return 0. Return 1 if the time and date cannot be found.

73950. arguments

73951. Now get the read-lock SHARED_LOCK

73952. Delete the row

73953. Statements to read/write/delete a record from xxx_parent

73954. ** Terminator values for position-lists and column-lists.

73955. Allocate the sqlite3_vtab structure

73956. constructor for a JsonEachCursor object for json_tree().

73957. Extract arguments from the current row of the ephemeral table and ** invoke the VUpdate method.

73958. Verify that an index entry exists for the current table row
73959. The first node of the scan
73960. Enable mutex on core
73961. OUT: Error message (if any)
73962. If true, skip the first field
73963. Open and close a Pager connection.
73964. Jump here if the row had already been deleted before any BEFORE ** trigger programs were invoked. Or if a trigger program throws a ** RAISE(IGNORE) exception.
73965. If the useJournal flag is clear, the journal-mode must be "OFF". ** And if the journal-mode is "OFF", the journal file must not be open.
73966. P4 is a pointer to a KeyInfo structure
73967. **comment:** ** GENERATED CODE ENDS HERE (mkportersteps.tcl)

label: code-design

73968. ** Clear any existing type flags from a Mem and replace them with f
73969. Convert "PRAGMA journal_mode" into "PRAGMA main.journal_mode"
73970. **comment:** copy max significant digits to significand
label: code-design

73971. **comment:** Extra argument
label: code-design

73972. OUT: Size of root node in bytes
73973. ** Do some work towards applying the RBU update to the target db. ** ** Return SQLITE_DONE if the update has been completely applied, or ** SQLITE_OK if no error occurs but there remains work to do to apply ** the RBU update. If an error does occur, some other error code is ** returned. ** ** Once a call to sqlite3rbu_step() has returned a value other than ** SQLITE_OK, all subsequent calls on the same RBU handle are no-ops ** that immediately return the same value.
73974. Available for future enhancements
73975. The USING clause of a join
73976. Allocate a VDBE
73977. ***** Continuing where we left off in os_unix.c *****
73978. 139
73979. Description of segment
73980. ** Adjust the cost C by the costMult factor T. This only occurs if ** compiled with -DSQLITE_ENABLE_COSTMULT
73981. For tracing shifts, the names of all terminals and nonterminals ** are required. The following table supplies these names
73982. Number of work-quanta to perform
73983. One (or both) of the named databases did not exist or an OOM ** error was hit. Or there is a transaction open on the destination ** database. The error has already been written into the pDestDb ** handle. All that is left to do here is free the sqlite3_backup ** structure.
73984. The left part of the FROM clause already seen
73985. Used to iterate through changes
73986. 510
73987. A buffer to hold the magic header
73988. No more than 1GiB per allocation
73989. ***** Begin file msvc.h *****
73990. 66
73991. Name of database session is attached to
73992. ** Helper type used by fts3EvalIncrPhraseNext() and incrPhraseTokenNext().
73993. Number of expressions on the list
73994. Statement to bind value to
73995. if decimal point is present
73996. Recursively resolve names in all subqueries
73997. Index level just updated
73998. ** Return the value of a system call. Return NULL if zName is not a ** recognized system call name. NULL is also returned if the system call ** is currently undefined.
73999. amalgamator: dontcache
74000. Last frame in log
74001. Height of this node in tree
74002. OUT: New tokenizer handle
74003. Separate file holding temporary table
74004. 1510
74005. case_operand ::=
74006. True if sqlite3_free(idxStr) is needed
74007. Like SRT_Queue, but unique results only
74008. Pointer to buffer to copy data from
74009. Prerequisites of pExpr
74010. 111
74011. **comment:** The sqlite3P2Values() routine is able to run faster if it knows ** the value of the largest JUMP opcode. The smaller the maximum ** JUMP opcode the better, so the mkopcodeh.tcl script that ** generated this include file strives to group all JUMP opcodes ** together near the beginning of the list.
label: code-design

74012. The next block of code is equivalent to: ** ** pIter += getVarint32(pIter, nPayload); ** ** The code is inlined to avoid a function call.
74013. ** Retry the changes accumulated in the pApply->constraints buffer.

74014. **comment:** Take steps to avoid spinning forever if there is a protocol error. ** ** Circumstances that cause a RETRY should only last for the briefest ** instances of time. No I/O or other system calls are done while the ** locks are held, so the locks should not be held for very long. But ** if we are unlucky, another process that is holding a lock might get ** paged out or take a page-fault that is time-consuming to resolve, ** during the few nanoseconds that it is holding the lock. In that case, ** it might take longer than normal for the lock to free. ** ** After 5 RETRYs, we begin calling sqlite3OsSleep(). The first few ** calls to sqlite3OsSleep() have a delay of 1 microsecond. Really this ** is more of a scheduler yield than an actual delay. But on the 10th ** an subsequent retries, the delays start becoming longer and longer, ** so that on the 100th (and last) RETRY we delay for 323 milliseconds. ** The total delay time before giving up is less than 10 seconds.
label: code-design

74015. ***** End of btree.c *****

74016. ** If an error has already occurred when this function is called, it ** immediately returns zero (without doing any work). Or, if an error ** occurs during the execution of this function, it sets the error code ** in the sqlite3rbu object indicated by the first argument and returns ** zero. ** ** The iterator passed as the second argument is guaranteed to point to ** a table (not an index) when this function is called. This function ** attempts to create any imposter table required to write to the main ** table b-tree of the table before returning. Non-zero is returned if ** an imposter table are created, or zero otherwise. ** ** An imposter table is required in all cases except RBU_PK_VTAB. Only ** virtual tables are written to directly. The imposter table has the ** same schema as the actual target table (less any UNIQUE constraints). ** More precisely, the "same schema" means the same columns, types, ** collation sequences. For tables that do not have an external PRIMARY ** KEY, it also means the same PRIMARY KEY declaration.

74017. Current leaf data
74018. ** Discard the contents of the cache.
74019. If non-zero, last token to include

74020. ** Overwrite cell iCell of node pNode with the contents of pCell.

74021. **comment:** ** Under certain circumstances, b-tree nodes (dolists) can be loaded into ** memory incrementally instead of all at once. This can be a big performance ** win (reduced IO and CPU) if SQLite stops calling the virtual table xNext() ** method before retrieving all query results (as may happen, for example, ** if a query has a LIMIT clause). *** Incremental loading is used for b-tree nodes FTS3_NODE_CHUNK_THRESHOLD ** bytes and larger. Nodes are loaded in chunks of FTS3_NODE_CHUNKSIZE bytes. ** The code is written so that the hard lower-limit for each of these values ** is 1. Clearly such small values would be inefficient, but can be useful ** for testing purposes. *** If this module is built with SQLITE_TEST defined, these constants may ** be overridden at runtime for testing purposes. File fts3_test.c contains ** a Tcl interface to read and write the values.

label: code-design

74022. Result of strlen(zDb)

74023. ** Create the shadow table named zPost, with definition zDefn. Return ** SQLITE_OK if successful, or an SQLite error code otherwise.

74024. ** Convert pMem to type integer. Invalidate any prior representations.

74025. If we get this far, it means we need to compute the table names. ** Note that the call to sqlite3ResultSetOfSelect() will expand any ** "" elements in the results set of the view and will assign cursors ** to the elements of the FROM clause. But we do not want these changes ** to be permanent. So the computation is done on a copy of the SELECT ** statement that defines the view.

74026. cmd ::= DROP TABLE ifexists fullname

74027. Because sqlite3_value_double() returns 0.0 if the argument is not ** something that can be converted into a number, we have: ** IMP: R-01992-00519 Abs(X) returns 0.0 if X is a string or blob ** that cannot be converted to a numeric value.

74028. Destructor for the codec

74029. ** Return a pointer to the WalCkptInfo structure in the wal-index.

74030. ** Sleep for a little while. Return the amount of time slept.

74031. xNextSystemCall

74032. Data type mismatch

74033. Jump to label "D"

74034. List of Table objects to delete after code gen

74035. Call this routine when reloading pages

74036. Where to write new cell content in data[]

74037. 6x

74038. ** This function may only be called while a write-transaction is active in ** rollback. If the connection is in WAL mode, this call is a no-op. ** Otherwise, if the connection does not already have an EXCLUSIVE lock on ** the database file, an attempt is made to obtain one. *** If the EXCLUSIVE lock is already held or the attempt to obtain it is ** successful, or the connection is in WAL mode, SQLITE_OK is returned. ** Otherwise, either SQLITE_BUSY or an SQLITE_IOERR_XXX error code is ** returned.

74039. Number of bytes in current position list

74040. Use the entire master

74041. The column on the LHS of the term. -1 for IPK

74042. Table being renamed

74043. jump, in1, in3

74044. ** Close all file descriptors accumulated in the unixInodeInfo->pUnused list.

74045. Value for 'idx' column of %_segdir

74046. phrase ::= phrase PLUS STRING star_opt

74047. pAux passed to create_module()

74048. ** Create a new tokenizer instance.

74049. Record to read varints from

74050. ** An instance of this object describes where to put of the results of ** a SELECT statement.

74051. Segment-reader for this token

74052. index of next token to be returned

74053. Number of columns in stat4 records

74054. Size of azConfig[]

74055. Second callback for SELECTs

74056. End of cell-pointer array

74057. Must be power of 2

74058. Bytes of payload on final overflow page

74059. a: p0<<28 | p2<<14 | p4 (unmasked)

74060. pPager->pFirstSynced = 0;

74061. Which entry in pAggInfo->aCol[] or ->aFunc[]

74062. Complete text of the cell

74063. Size of aOp array

74064. Width of the current field

74065. Count the total number of references to pOuterNC and all of its ** parent contexts. After resolving references to expressions in ** pItem->pSelect, check if this value has changed. If so, then ** SELECT statement pItem->pSelect must be correlated. Set the ** pItem->fg.isCorrelated flag if this is the case.

74066. Index of last valid frame in the WAL

74067. **comment:** TODO(shess) current implementation requires pInput to be ** nul-terminated. This should either be fixed, or pInput/nBytes ** should be converted to zInput.

label: code-design

74068. fts3_tokenize_vtab.c

74069. This is always a CONSTRAINT conflict.

74070. Fewer features

74071. Max length of an SQL string

74072. A-vs-B comparision jump

74073. Add the PK key for this row to the temporary table

74074. Right operand: 0==FALSE, 1==TRUE, 2==UNKNOWN or NULL

74075. Notice that even though SF_Distinct has been cleared from p->selFlags, ** the sDistinct.isTnct is still set. Hence, isTnct represents the ** original setting of the SF_Distinct flag, not the current setting

74076. Address of the output-A subroutine

74077. return value form SQLITE_FCNTL_PAGMA

74078. At this point pParent may have at most one overflow cell. And if ** this overflow cell is present, it must be the cell with ** index iParentIdx. This scenario comes about when this function ** is called (indirectly) from sqlite3BtreeDelete().

74079. Remove pPage from dirty list

74080. ** CAPI3REF: 64-Bit Integer Types ** KEYWORDS: sqlite_int64 sqlite_uint64 ** ** Because there is no cross-platform way to specify 64-bit integer types ** SQLite includes typedefs for 64-bit signed and unsigned integers. ** ** The sqlite3_int64 and sqlite3_uint64 are the preferred type definitions. ** The sqlite_int64 and sqlite_uint64 types are supported for backwards ** compatibility only. ** ** ^The sqlite3_int64 and sqlite_int64 types can store integer values ** between -922372036854775808 and +922372036854775807 inclusive. ^The ** sqlite3_uint64 and sqlite_uint64 types can store integer values ** between 0 and +18446744073709551615 inclusive.

74081. True if there is an INDEXED BY clause

74082. create a copy of the lock path _only_ if the conch is taken

74083. Boolean. True if locking_mode==EXCLUSIVE

74084. 32

74085. Releasing a reader lock or an exclusive lock

74086. Dx

74087. Set to true after xInput finished
74088. IcuCursor.aChar[]
74089. Source pager
74090. Delete the %_docszie record
74091. Another thread is (in the process of) initializing the static ** mutexes
74092. WITH clause attached to this select. Or NULL.
74093. The virtual machine under construction
74094. Opcode: Expire P1 * * * * * Cause precompiled statements to expire. When an expired statement ** is executed using sqlite3_step() it will either automatically ** reprepare itself (if it was originally created using sqlite3_prepare_v2()) ** or it will fail with SQLITE_SCHEMA. *** If P1 is 0, then all SQL statements become expired. If P1 is non-zero, ** then only the currently executing statement is expired.
74095. Size of zAppend in bytes (or -1)
74096. Index of cell to delete
74097. ** Argument pTerm must be a synonym iterator. Return the current rowid ** that it points to.
74098. 28..2f ()*+,-/
74099. ** Obtain a patchset object containing all changes recorded by the ** session object passed as the first argument. *** It is the responsibility of the caller to eventually free the buffer ** using sqlite3_free().
74100. File format to use for encoding
74101. If the buffer is now larger than SESSIONS_STRM_CHUNK_SIZE, pass ** its contents to the xOutput() callback.
74102. Bind the new.* PRIMARY KEY values to the SELECT statement.
74103. ** Convert zData into one or more integers according to the conversion ** specifier zFormat. *** zFormat[] contains 4 characters for each integer converted, except for ** the last integer which is specified by three characters. The meaning ** of a four-character format specifiers ABCD is: *** A: number of digits to convert. Always "2" or "4". ** B: minimum value. Always "0" or "1". ** C: maximum value, decoded as: ** a: 12 ** b: 14 ** c: 24 ** d: 31 ** e: 59 ** f: 9999 ** D: the separator character, or \000 to indicate this is the ** last number to convert. *** Example: To translate an ISO-8601 date YYYY-MM-DD, the format would ** be "40f-21a-20c". The "40f-" indicates the 4-digit year followed by "-". ** The "21a-" indicates the 2-digit month followed by "-". The "20c" indicates ** the 2-digit day which is the last integer in the set. *** The function returns the number of successful conversions.
74104. ** The xConnect() and xCreate() methods for the virtual table. All the ** work is done in function fts5InitVtab().
74105. Number of tables to check. (Number of root pages.)
74106. 1: (end_constraints && !bRev && endEq)
74107. Number of arguments. -1 means any number
74108. ** Create a session object. This session object will record changes to ** database zDb attached to connection db.
74109. '+' or '-' or 0 for prefix
74110. RBU object
74111. Current 'rowid' value
74112. flag ASCII delimiters
74113. Normalize realvalue to within 10.0 > realvalue >= 1.0
74114. ** Return true if the Btree passed as the only argument is sharable.
74115. Used when p4type is P4_COLLSEQ
74116. If an AS-name match is found, mark this ORDER BY column as being ** a copy of the iCol-th result-set column. The subsequent call to ** sqlite3ResolveOrderGroupBy() will convert the expression to a ** copy of the iCol-th result-set expression.
74117. Column index
74118. Checked before the virtual table is created
74119. Size of zToken in bytes
74120. IMP: R-04300-56712
74121. Omit children but continue walking siblings
74122. Check if the cursor is past the end of the docid range specified ** by Fts3Cursor.iMinDocid/iMaxDocid. If so, set the EOF flag.
74123. dbSize before the current transaction
74124. Update the pager's copy of the change-counter. Otherwise, the ** next time a read transaction is opened the cache will be ** flushed (as the change-counter values will not match).
74125. Check that the array is in order and contains no duplicate entries.
74126. The current doclist-index page is full. Write it to disk and push ** a copy of iRowid (which will become the first rowid on the next ** doclist-index leaf page) up into the next level of the b-tree ** hierarchy. If the node being flushed is currently the root node, ** also push its first rowid upwards.
74127. The first element of the array
74128. Detect a pid change and reset the PRNG. There is a race condition ** here such that two or more threads all trying to open databases at ** the same instant might all reset the PRNG. But multiple resets ** are harmless.
74129. Record to decode
74130. SQLITE_BTREE_H
74131. Jump here to skip excluded rows
74132. ** Make sure all writes to a particular file are committed to disk.
74133. An exclusive process lock is held
74134. ***** Begin file wal.c *****
74135. Docid to bind for SQL_SELECT_DOCSIZE
74136. If the column is declared as "<name> PRIMARY KEY COLLATE <type>", ** then an index may have been created on this column before the ** collation type was added. Correct this if it is the case.
74137. Opcode: SeekLE P1 P2 P3 P4 * * * Synopsis: key=r[P3@P4] *** If cursor P1 refers to an SQL table (B-Tree that uses integer keys), ** use the value in register P3 as a key. If cursor P1 refers ** to an SQL index, then P3 is the first in an array of P4 registers ** that are used as an unpacked index key. *** Reposition cursor P1 so that it points to the largest entry that ** is less than or equal to the key value. If there are no records ** less than or equal to the key and P2 is not zero, then jump to P2. *** This opcode leaves the cursor configured to move in reverse order, ** from the end toward the beginning. In other words, the cursor is ** configured to use Prev, not Next. *** If the cursor P1 was opened using the OPFLAG_SEEKEQ flag, then this ** opcode will always land on a record that equally equals the key, or ** else jump immediately to P2. When the cursor is OPFLAG_SEEKEQ, this ** opcode must be followed by an IdxGE opcode with the same arguments. ** The IdxGE opcode will be skipped if this opcode succeeds, but the ** IdxGE opcode will be used on subsequent loop iterations. *** See also: Found, NotFound, SeekGt, SeekGe, SeekLt
74138. ** This function is called at the start of every write transaction. ** There must already be a RESERVED or EXCLUSIVE lock on the database ** file when this routine is called. *** Open the journal file for pager pPager and write a journal header ** to the start of it. If there are active savepoints, open the sub-journal ** as well. This function is only used when the journal file is being ** opened to write a rollback log for a transaction. It is not used ** when opening a hot journal file to roll it back. *** If the journal file is already open (as it may be in exclusive mode), ** then this function just writes a journal header to the start of the ** already open file. *** Whether or not the journal file is opened by this function, the ** Pager.pInJournal bitvec structure is allocated. *** Return SQLITE_OK if everything is successful. Otherwise, return ** SQLITE_NOMEM if the attempt to allocate Pager.pInJournal fails, or ** an IO error code if opening or writing the journal file fails.
74139. Sector-size field of journal header
74140. ** CAPI3REF: Checkpoint Mode Values ** KEYWORDS: {checkpoint mode} *** These constants define all valid values for the "checkpoint mode" passed ** as the third parameter to the [sqlite3_wal_checkpoint_v2()] interface. ** See the [sqlite3_wal_checkpoint_v2()] documentation for details on the ** meaning of each of these checkpoint modes.
74141. Forward declaration required by incrVacuumStep().
74142. # include <pthread.h>
74143. CONTENT
74144. Pointer to the end of the last xRead()
74145. ** Call this routine when the database connection is closing in order ** to clean up loaded extensions
74146. The expression list to be analyzed.

74147. !defined(SQLITE OMIT VIEW) || !defined(SQLITE OMIT VIRTUALTABLE)
74148. ccons ::= CONSTRAINT nm
74149. Number of PtrMap pages to be freed
74150. Address of loop counter
74151. For hash-table collisions
74152. First (only) step of trigger program
74153. Strings with " doubled and enclosed in ", NULL pointers replaced by SQL NULL. %Q
74154. 210
74155. Number of fields in the record
74156. Cursor opened by OpenSorter (if in use)
74157. *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used by the compiler to add foreign key ** support to compiled SQL statements.
74158. pMethod
74159. Includes the ALL keyword
74160. ** Like free() but works for allocations obtained from sqlite3MemMalloc() ** or sqlite3MemRealloc(). ** For this low-level routine, we already know that pPrior!=0 since ** cases where pPrior==0 will have been intercepted and dealt with ** by higher-level routines.
74161. Next unparsed byte of the header
74162. Parse context to store any error in
74163. Object used to write leaf nodes
74164. Resolve the expressions in the LIMIT and OFFSET clauses. These ** are not allowed to refer to any names, so pass an empty NameContext.
74165. ***** End destructor definitions *****
74166. Page number of the page to check
74167. End the virtual table scan
74168. Notify of page size changes
74169. IN/OUT: Mask of phrases seen
74170. ** Get or set the "averages" values.
74171. synopsis: if key(P1)!=trim(r[P3],P4) goto P2
74172. Each entry in the following array defines a rule for folding a range ** of codepoints to lower case. The rule applies to a range of nRange ** codepoints starting at codepoint iCode. ** If the least significant bit in flags is clear, then the rule applies ** to all nRange codepoints (i.e. all nRange codepoints are upper case and ** need to be folded). Or, if it is set, then the rule only applies to ** every second codepoint in the range, starting with codepoint C. ** The 7 most significant bits in flags are an index into the aiOff[] ** array. If a specific codepoint C does require folding, then its lower ** case equivalent is ((C + aiOff[flags>>1]) & 0xFFFF). ** The contents of this array are generated by parsing the CaseFolding.txt ** file distributed as part of the "Unicode Character Database". See ** <http://www.unicode.org> for details.
74173. ** Implementation of the sqlite3_pcachexxCachesize method. ** Configure the cache_size limit for a cache.
74174. Address of the end of the query
74175. **comment:** Size of malloc'd buffer at zMalloc
 label: code-design
74176. **comment:** ** 2010 July 12 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains an implementation of the "dbstat" virtual table. ** The dbstat virtual table is used to extract low-level formatting ** information from an SQLite database in order to implement the ** "sqlite3_analyzer" utility. See the ..//tool/spaceanal.tcl script ** for an example implementation. ** Additional information is available on the "dbstat.html" page of the ** official SQLite documentation.
 label: code-design
74177. ** pNew and pOld are both candidate non-periodic samples selected for ** the same column (pNew->iCol==pOld->iCol). Ignoring this column and ** considering only any trailing columns and the sample hash value, this ** function returns true if sample pNew is to be preferred over pOld. ** In other words, if we assume that the cardinalities of the selected ** column for pNew and pOld are equal, is pNew to be preferred over pOld. ** This function assumes that for each argument sample, the contents of ** the anEq[] array from pSample->anEq[pSample->iCol+1] onwards are valid.
74178. Average nEq values for keys not in aSample
74179. ** Determine if we are dealing with Windows NT. ** We ought to be able to determine if we are compiling for Windows 9x or ** Windows NT using the _WIN32_WINNT macro as follows: ** #if defined(_WIN32_WINNT) ** # define SQLITE_OS_WINNT 1 ** #else ** # define SQLITE_OS_WINNT 0 ** #endif ** However, Visual Studio 2005 does not set _WIN32_WINNT by default, as ** it ought to, so the above test does not work. We'll just assume that ** everything is Windows NT unless the programmer explicitly says otherwise ** by setting SQLITE_OS_WINNT to 0.
74180. lastErrno set by seekAndWrite
74181. pLogArg
74182. Page cache buffer management: ** These routines implement SQLITE_CONFIG_PAGECACHE.
74183. The number of integers to encode
74184. synopsis: r[P2]=count()
74185. If existing WhereLoop p is better than pTemplate, pTemplate can be ** discarded. WhereLoop p is better if: ** (1) p has no more dependencies than pTemplate, and ** (2) p has an equal or lower cost than pTemplate
74186. Offset of first term on leaf
74187. Session object
74188. Frame number corresponding to aPgno[0]
74189. Power of two >= nSeg
74190. IN/OUT: Current structure of index
74191. IN/OUT: Value read from position-list
74192. ASC => ID
74193. **comment:** Temporary space used to build header
 label: code-design
74194. **comment:** Counter to prevent infinite loop of reprepares
 label: code-design
74195. Mask of affinity bits
74196. ** This function is used to process a single interior node when searching ** a b-tree for a term or term prefix. The node data is passed to this ** function via the zNode/nNode parameters. The term to search for is ** passed in zTerm/nTerm. ** If piFirst is not NULL, then this function sets *piFirst to the blockid ** of the child node that heads the sub-tree that may contain the term. ** If piLast is not NULL, then *piLast is set to the right-most child node ** that heads a sub-tree that may contain a term for which zTerm/nTerm is ** a prefix. ** If an OOM error occurs, SQLITE_NOMEM is returned. Otherwise, SQLITE_OK.
74197. ** Insert or remove data to or from the index. Each time a document is ** added to or removed from the index, this function is called one or more ** times. ** For an insert, it must be called once for each token in the new document. ** If the operation is a delete, it must be called (at least) once for each ** unique token in the document with an iCol value less than zero. The iPos ** argument is ignored for a delete.
74198. Unless it is empty, flush the hash table to disk
74199. ** Free the iterator object passed as the second argument.
74200. Copy over the row data
74201. IMP: R-42790-23372
74202. Length of the parameter token
74203. Term to write to internal node
74204. Max args passed to user function by sub-program

74205. Bytes of space for iSerial as varint
74206. Reusable bulk memory
74207. Register holding max rowid
74208. Result from sqlite3Step()
74209. PLAN => ID
74210. ** The two values that may be meaningfully bound to the :1 parameter in ** statements SQL_REPLACE_STAT and SQL_SELECT_STAT.
74211. Size of zStr
74212. True when updating a view (INSTEAD OF trigger)
74213. ** These #defines should enable >2GB file support on POSIX if the ** underlying operating system supports it. If the OS lacks ** large file support, or if the OS is windows, these should be no-ops. *** Ticket #2739: The _LARGEFILE_SOURCE macro must appear before any ** system #includes. Hence, this block of code must be the very first ** code in all source files. *** Large file support can be disabled using the -DSQLITE_DISABLE_LFS switch ** on the compiler command line. This is necessary if you are compiling ** on a recent machine (ex: Red Hat 7.2) but you want your code to work ** on an older machine (ex: Red Hat 6.0). If you compile on Red Hat 7.2 ** without this option, LFS is enable. But LFS does not exist in the kernel ** in Red Hat 6.0, so the code won't work. Hence, for maximum binary ** portability you should omit LFS. *** The previous paragraph was written in 2005. (This paragraph is written ** on 2008-11-28.) These days, all Linux kernels support large files, so ** you should probably leave LFS enabled. But some embedded platforms might ** lack LFS in which case the SQLITE_DISABLE_LFS macro might still be useful. *** Similar is true for Mac OS X. LFS is only supported on Mac OS X 9 and later.
74214. ** Add a new subitem to the tree. The moreToFollow flag indicates that this ** is not the last item in the tree.
74215. Freeblock not in ascending order
74216. ** Macros for looping over all elements of a hash table. The idiom is ** like this: *** Fts3Hash h; ** Fts3HashElem *p; ** ... ** for(p=fts3HashFirst(&h); p; p=fts3HashNext(p)){ ** SomeStructure *pData = fts3HashData(p); ** // do something with pData ** }
74217. Undo any changes made by applyAffinity() to the input registers.
74218. ** Return a list of N comma separated question marks, where N is the number ** of columns in the %_content table (one for the docid plus one for each ** user-defined text column). *** If argument zFunc is not NULL, then all but the first question mark ** is preceded by zFunc and an open bracket, and followed by a closed ** bracket. For example, if zFunc is "zip" and the FTS3 table has three ** user-defined text columns, the following string is returned: *** "?, zip(?), zip(?), zip(?)" *** The pointer returned points to a buffer allocated by sqlite3_malloc(). It ** is the responsibility of the caller to eventually free it. *** If *pRc is not SQLITE_OK when this function is called, it is a no-op (and ** a NULL pointer is returned). Otherwise, if an OOM error is encountered ** by this function, NULL is returned and *pRc is set to SQLITE_NOMEM. If ** no error occurs, *pRc is left unmodified.
74219. If the statement completed successfully, invoke the profile callback
74220. Straight index lookup
74221. The column containing the blob
74222. IN/OUT: SegmentNode handle
74223. OUT: Return data in this Mem structure.
74224. In the current implementation, the rSetup value is either zero ** or the cost of building an automatic index (NlogN) and the NlogN ** is the same for compatible WhereLoops.
74225. pIdx is a UNIQUE index (or a PRIMARY KEY) and has the right number ** of columns. If each indexed column corresponds to a foreign key ** column of pFKey, then this index is a winner.
74226. OE_Rollback etc.
74227. This is likely to be an append
74228. ** Determine the current size of a file in bytes
74229. Value returned by last rbu_step() call
74230. Bytes of memory space allocated for Vdbe.aOp[]
74231. Begin generating the code that will insert the table record into ** the SQLITE_MASTER table. Note in particular that we must go ahead ** and allocate the record number for the table entry now. Before any ** PRIMARY KEY or UNIQUE keywords are parsed. Those keywords will cause ** indices to be created and the table record must come before the ** indices. Hence, the record number for the table must be allocated ** now.
74232. **comment:** ** Mark all temporary registers as being unavailable for reuse.
label: code-design
74233. Value for "idx" field
74234. IMPLEMENTATION-OF: R-44699-57140 This mode works the same way as ** SQLITE_CHECKPOINT_RESTART with the addition that it also ** truncates the log file to zero bytes just prior to a ** successful return. *** In theory, it might be safe to do this without updating the ** wal-index header in shared memory, as all subsequent reader or ** writer clients should see that the entire log file has been ** checkpointed and behave accordingly. This seems unsafe though, ** as it would leave the system in a state where the contents of ** the wal-index header do not match the contents of the ** file-system. To avoid this, update the wal-index header to ** indicate that the log file contains zero valid frames.
74235. Attach the temporary database as 'vacuum_db'. The synchronous pragma ** can be set to 'off' for this file, as it is not recovered if a crash ** occurs anyway. The integrity of the database is maintained by a ** (possibly synchronous) transaction opened on the main database before ** sqlite3BtreeCopyFile() is called. *** An optimisation would be to use a non-journalled pager. ** (Later:) I tried setting "PRAGMA vacuum_db.journal_mode=OFF" but ** that actually made the VACUUM run slower. Very little journaling ** actually occurs when doing a vacuum since the vacuum_db is initially ** empty. Only the journal header is written. Apparently it takes more ** time to parse and run the PRAGMA to turn journaling off than it does ** to write the journal header file.
74236. New array of Db pointers
74237. Append data to the wal-index. It is not necessary to lock the ** wal-index to do this as the SQLITE_SHM_WRITE lock held on the wal-index ** guarantees that there are no other writers, and no data that may ** be in use by existing readers is being overwritten.
74238. ** Implementation of the changes() SQL function. *** IMP: R-62073-11209 The changes() SQL function is a wrapper ** around the sqlite3_changes() C/C++ function and hence follows the same ** rules for counting changes.
74239. Input stream call (or NULL)
74240. ** This function is called after a transaction has been committed. It ** invokes callbacks registered with sqlite3_wal_hook() as required.
74241. File descriptor for database pTo
74242. OUT: Bytes in returned value
74243. ** Return the full pathname of the journal file.
74244. Affinity string to modify
74245. ** Generate code into the current Vdbe to evaluate the given ** expression. Attempt to store the results in register "target". ** Return the register where results are stored. *** With this routine, there is no guarantee that results will ** be stored in target. The result might be stored in some other ** register if it is convenient to do so. The calling function ** must check the return code and move the results to the desired ** register.
74246. Content
74247. This is only called if it is guaranteed that the phrase has at least ** one incremental token. In which case the bIncr flag is set.
74248. **comment:** ** CAPI3REF: Mutex Methods Object ** ** An instance of this structure defines the low-level routines ** used to allocate and use mutexes. *** Usually, the default mutex implementations provided by SQLite are ** sufficient, however the application has the option of substituting a custom ** implementation for specialized deployments or systems for which SQLite ** does not provide a suitable implementation. In this case, the application ** creates and populates an instance of this structure to pass ** to sqlite3_config() along with the [SQLITE_CONFIG_MUTEX] option. ** Additionally, an instance of this structure can be used as an ** output variable when querying the system for the current mutex ** implementation, using the [SQLITE_CONFIG_GETMUTEX] option. *** ^The xMutexInit method defined by this structure is invoked as ** part of system initialization by the sqlite3_initialize() function. ** ^The xMutexInit routine is called by SQLite exactly once for each ** effective call to [sqlite3_initialize()]. *** ^The xMutexEnd method defined by this structure is invoked as ** part of system shutdown by the sqlite3_shutdown() function. The ** implementation of this method is expected to release all outstanding ** resources obtained by the mutex methods implementation, especially ** those obtained by the xMutexInit method. ^The xMutexEnd() ** interface is invoked exactly once for each call to [sqlite3_shutdown()]. *** ^The remaining seven methods defined by this structure (xMutexAlloc, ** xMutexFree, xMutexEnter, xMutexTry, xMutexLeave, xMutexHeld and ** xMutexNoheld) implement the following interfaces (respectively): *** ** [sqlite3_mutex_alloc()] ** [sqlite3_mutex_free()] ** [sqlite3_mutex_enter()] ** [sqlite3_mutex_try()] ** [sqlite3_mutex_leave()] ** [sqlite3_mutex_held()] ** [sqlite3_mutex_noheld()] ** *** The only difference is that the public sqlite3_XXX functions enumerated ** above silently ignore any invocations that pass a NULL pointer instead ** of a valid mutex handle. The implementations of the methods defined ** by this structure are not required to handle this case, the results ** of passing a NULL pointer instead of a valid mutex handle are undefined ** (i.e. it is acceptable to

provide an implementation that segfaults if ** it is passed a NULL pointer). *** The xMutexInit() method must be threadsafe. It must be harmless to ** invoke xMutexInit() multiple times within the same process and without ** intervening calls to xMutexEnd(). Second and subsequent calls to ** xMutexInit() must be no-ops. *** xMutexInit() must not use SQLite memory allocation ([sqlite3_malloc()] ** and its associates). Similarly, xMutexAlloc() must not use SQLite memory ** allocation for a static mutex. ^However xMutexAlloc() may use SQLite ** memory allocation for a fast or recursive mutex. *** ^SQLite will invoke the xMutexEnd() method when [sqlite3_shutdown()] is ** called, but only if the prior call to xMutexInit returned SQLITE_OK. ** If xMutexInit fails in any way, it is expected to clean up after itself ** prior to returning.

label: code-design

- 74249. Variable used as the phrase counter
- 74250. Number of backends currently in use
- 74251. End of the loop over all rowids/primary-keys.
- 74252. jump
- 74253. Change to WRITER_LOCKED state. *** WAL mode sets Pager.eState to PAGER_WRITER_LOCKED or CACHEMOD ** when it has an open transaction, but never to DBMOD or FINISHED. ** This is because in those states the code to roll back savepoint ** transactions may copy data from the sub-journal into the database ** file as well as into the page cache. Which would be incorrect in ** WAL mode.
- 74254. Check if the iterator is now at EOF. If so, return early.
- 74255. src table col -> target table col
- 74256. ** The second argument points to an FKey object representing a foreign key ** for which pTab is the parent table. An UPDATE statement against pTab ** is currently being processed. For each column of the table that is ** actually updated, the corresponding element in the aChange[] array ** is zero or greater (if a column is unmodified the corresponding element ** is set to -1). If the rowid column is modified by the UPDATE statement ** the bChngRowid argument is non-zero. *** This function returns true if any of the columns that are part of the ** parent key for FK constraint *p are modified.
- 74257. Some views have defined column names
- 74258. ** Fts5SegIter.iLeafOffset currently points to the first byte of the ** "nSuffix" field of a term. Function parameter nKeep contains the value ** of the "nPrefix" field (if there was one - it is passed 0 if this is ** the first term in the segment). *** This function populates: *** Fts5SegIter.term ** Fts5SegIter.rowid ** accordingly and leaves (Fts5SegIter.iLeafOffset) set to the content of ** the first position list. The position list belonging to document ** (Fts5SegIter.iRowid).
- 74259. ** Something has moved cursor "p" out of place. Maybe the row it was ** pointed to was deleted out from under it. Or maybe the btree was ** rebalanced. Whatever the cause, try to restore "p" to the place it ** is supposed to be pointing. If the row was deleted out from under the ** cursor, set the cursor to point to a NULL row.
- 74260. Name of the file being opened
- 74261. The pager cache has created a new page. Its content needs to ** be initialized. But first some error checks: *** (1) The maximum page number is 2^31 ** (2) Never try to fetch the locking page
- 74262. ** Store the union of cells p1 and p2 in p1.
- 74263. ** Parse a complete JSON string. Return 0 on success or non-zero if there ** are any errors. If an error occurs, free all memory associated with ** pParse. *** pParse is uninitialized when this routine is called.
- 74264. Building a VDBE program
- 74265. Statement for reading and writing
- 74266. The special 'table-name' column
- 74267. fts5YYWILDCARD
- 74268. Index of root in pWriter->aNodeWriter
- 74269. 1 for tables and indices to be opened
- 74270. Obtain pointers to the hash-table and page-number array containing ** the entry that corresponds to frame pWal->hdr.mxFrame. It is guaranteed ** that the page said hash-table and array reside on is already mapped.
- 74271. The following calls to preupdate_new() and preupdate_old() can not ** fail. This is because they cache their return values, and by the ** time control flows to here they have already been called once from ** within sessionPreupdateHash(). The first two asserts below verify ** this (that the method has already been called).
- 74272. The new entry
- 74273. Right hand serial type
- 74274. 97
- 74275. Pointer to an array of results
- 74276. Base register holding constraint values
- 74277. OUT: Inverse of pChangeset
- 74278. If filePath==NULL that means we are dealing with a transient file ** that does not need to be locked.
- 74279. The definition of this CTE
- 74280. EVIDENCE-OF: R-06529-47362 Following the size varint are one or more ** additional varints, one per column.
- 74281. Information about the function
- 74282. OUT: Set to true if row really does exist
- 74283. P4 is a pointer to BtreeNext() or BtreePrev()
- 74284. EXCEPT
- 74285. Zero from this point onwards on cursor reset
- 74286. Don't allow access, but don't generate an error
- 74287. Begin a transaction for database iDb. ** Then modify the schema cookie (since the ALTER TABLE modifies the ** schema). Open a statement transaction if the table is a virtual ** table.
- 74288. only 1 core, use our own zone to contention over global locks, ** e.g. we have our own dedicated locks
- 74289. Space for aKey if aBuffer and pMap wont work
- 74290. op2 value for LIKE/REGEXP/GLOB
- 74291. True if this op is a delete
- 74292. Sorter cursor to read from
- 74293. Name of rank function
- 74294. Cursor to table into which insert is written
- 74295. ** Shutdown the mutex system. This call frees resources allocated by ** sqlite3MutexInit().
- 74296. ** Return the rowid that the cursor currently points to.
- 74297. Cursor of the RHS table
- 74298. ** Maximum pathname length (in chars) for WinNT. This should normally be ** UNICODE_STRING_MAX_CHARS.
- 74299. Value for the stat column of sqlite_stat1
- 74300. Store the results of the setup-query in Queue.
- 74301. ** Change the values for the BTS_SECURE_DELETE and BTS_OVERWRITE flags: *** newFlag==0 Both BTS_SECURE_DELETE and BTS_OVERWRITE are cleared ** newFlag==1 BTS_SECURE_DELETE set and BTS_OVERWRITE is cleared ** newFlag==2 BTS_SECURE_DELETE cleared and BTS_OVERWRITE is set ** newFlag==(-1) No changes ** ** This routine acts as a query if newFlag is less than zero ** ** With BTS_OVERWRITE set, deleted content is overwritten by zeros, but ** freelist leaf pages are not written back to the database. Thus in-page ** deleted content is cleared, but freelist deleted content is not. ** ** With BTS_SECURE_DELETE, operation is like BTS_OVERWRITE with the addition ** that freelist leaf pages are written back into the database, increasing ** the amount of disk I/O.
- 74302. Value to return for s3_vtab_on_conflict()
- 74303. Database in which the table lives
- 74304. Map from pIdx cols to child table cols
- 74305. The direct assignment in the previous line is possible only because ** the WO_ and SQLITE_INDEX_CONSTRAINT_ codes are identical. The ** following asserts verify this fact.
- 74306. Continue down into children
- 74307. xRowid
- 74308. ***** End of ctime.c *****
- 74309. True if iRoot is the root of an index b-tree

74310. By convertCompoundSelectToSubquery()

74311. Top of the update loop

74312. Size of allocated array at aElem

74313. ** Implementation of the like() SQL function. This function implements ** the build-in LIKE operator. The first argument to the function is the ** pattern and the second argument is the string. So, the SQL statements: *** ** A LIKE B *** ** is implemented as like(B, A). If there is an escape character E, *** ** A LIKE B ESCAPE E *** ** is mapped to like(B, A, E).

74314. xSqllog, void*

74315. onconf ::=

74316. ***** Begin file vdbeInt.h *****

74317. Parameters to bind

74318. Used to iterate from pExpr to root

74319. Current position

74320. Invalid key size: 0x80 0x80 0x00

74321. Translator union

74322. Really a cast to BLOB

74323. INITIALLY => ID

74324. End of the overrideable system calls

74325. Name of r-tree table

74326. ** Register a new geometry function for use with the r-tree MATCH operator.

74327. **comment:** ** This function is called after the contents of page iPage of the ** source database have been modified. If page iPage has already been ** copied into the destination database, then the data written to the ** destination is now invalidated. The destination copy of iPage needs ** to be updated with the new data before the backup operation is ** complete. *** It is assumed that the mutex associated with the BtShared object ** corresponding to the source database is held when this function is ** called.
label: code-design

74328. ** Free an RbuState object allocated by rbuLoadState().

74329. Allocate cursors for Current, Queue, and Distinct.

74330. String to return via *pzImposterCols

74331. Copy of p->aOp

74332. Compute only a unique prefix of the key

74333. Max iDivisor is max(u32) / BITVEC_NPTR + 1.

74334. ** Allocate and populate an sqlite3_index_info structure. It is the ** responsibility of the caller to eventually release the structure ** by passing the pointer returned by this function to sqlite3_free().

74335. Rowid components

74336. languageid=? parameter (or NULL)

74337. Mark off any other ORDER BY terms that reference pLoop

74338. if(!pIncr->bUseThread)

74339. xDelete

74340. ** Join thread pTask->thread.

74341. SQLITE_OMIT_OR_OPTIMIZATION

74342. ** This function determines if the machine is running a version of Windows ** based on the NT kernel.

74343. ** The first element in the apVal[] array is assumed to contain the docid ** (an integer) of a row about to be deleted. Remove all terms from the ** full-text index.

74344. ***** End of bitvec.c *****

74345. Different number of columns

74346. 219

74347. Maximum local payload in non-LEAFDATA tables

74348. Previously generated index key

74349. aKWHash[i] is the hash value for the i-th keyword

74350. Pointer to function information

74351. Whether this is an 'x IN(SELECT...)' or an 'x IN(<exprlist>)' ** expression it is handled the same way. An ephemeral table is ** filled with index keys representing the results from the ** SELECT or the <exprlist>. ** If the 'x' expression is a column value, or the SELECT... ** statement returns a column value, then the affinity of that ** column is used to build the index keys. If both 'x' and the ** SELECT... statement are columns, then numeric affinity is used ** if either column has NUMERIC or INTEGER affinity. If neither ** 'x' nor the SELECT... statement are columns, then numeric affinity ** is used.

74352. OUT: New virtual table

74353. ** 2001 September 16 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This header file (together with its companion C source-code file ** "os.c") attempt to abstract the underlying operating system so that ** the SQLite library will work on both POSIX and windows systems. *** ** This header file is #include-ed by sqliteInt.h and thus ends up ** being included by every source file.

74354. new.* record for second change

74355. ***** Include hwtime.h in the middle of os_common.h *****

74356. No DISTINCT keyword and no aggregate functions

74357. Buffer to assemble wal-header in

74358. Size of buffer at aDoclist

74359. ***** End of os_common.h *****

74360. Slots allocated for azResult[]

74361. Do not gather statistics on system tables

74362. Update the count of rows that are inserted

74363. ** Load the next leaf page into the segment iterator.

74364. ** Write nBuf bytes of random data to the supplied buffer zBuf.

74365. **comment:** Groupid for the file
label: code-design

74366. ** 2000-05-29 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Driver template for the LEMON parser generator. *** ** The "lemon" program processes an LALR(1) input grammar file, then uses ** this template to construct a parser. The "lemon" program inserts text ** at each "%%" line. Also, any "P-a-r-s-e" identifier prefix (without the ** interstitial "-" characters) contained in this template is changed into ** the value of the %name directive from the grammar. Otherwise, the content ** of this template is copied straight through into the generate parser ** source file. *** ** The following is the concatenation of all %include directives from the ** input grammar file:

74367. Label "L"

74368. An instance of the SubstContext object describes an substitution edit ** to be performed on a parse tree. *** ** All references to columns in table iTable are to be replaced by corresponding ** expressions in pEList.

74369. WITHOUT

74370. Implements a UNIQUE constraint

74371. ** This function constructs and returns a pointer to a nul-terminated ** string containing some SQL clause or list based on one or more of the ** column names currently stored in the pIter->azTblCol[] array.

74372. 370

74373. now an error

74374. Refs more than one table

74375. ***** sqlite3_column_ ***** The following routines are used to access elements of the current row ** in the result set.

74376. Currently, Fts5SegIter.iLeafOffset points to the first byte of ** position-list content for the current rowid. Back it up so that it ** points to the start of the position-list size field.

74377. pOrderBy is really a GROUP BY

74378. ** Allocate and return a pointer to a new sqlite3_value object. If ** the second argument to this function is NULL, the object is allocated ** by calling sqlite3ValueNew(). ** ** Otherwise, if the second argument is non-zero, then this function is ** being called indirectly by sqlite3Stat4ProbeSetValue(). If it has not ** already been allocated, allocate the UnpackedRecord structure that ** that function will return to its caller here. Then return a pointer to ** an sqlite3_value within the UnpackedRecord.a[] array.

74379. Coroutine supplying data

74380. Original value of pNew->u.btree.nBtm

74381. ** Change the key on an open database. If the current database is not ** encrypted, this routine will encrypt it. If pNew==0 or nNew==0, the ** database is decrypted. ** ** The code to implement this API is not available in the public release ** of SQLite.

74382. ** Estimate the number of rows that will be returned based on ** an equality constraint x=VALUE and where that VALUE occurs in ** the histogram data. This only works when x is the left-most ** column of an index and sqlite_stat3 histogram data is available ** for that index. When pExpr==NULL that means the constraint is ** "x IS NULL" instead of "x=VALUE". ** ** Write the estimated row count into *pnRow and return SQLITE_OK. ** If unable to make an estimate, leave *pnRow unchanged and return ** non-zero. ** ** This routine can fail if it is unable to load a collating sequence ** required for string comparison, or if unable to allocate memory ** for a UTF conversion required for comparison. The error is stored ** in the pParse structure.

74383. SQLITE_OMIT_COMPOUND_SELECT

74384. Copy Btree meta values

74385. **comment:** The FROM clause term to get the next index
label: code-design

74386. Figure out if this is a leaf or an internal node.

74387. p is always 4-byte aligned

74388. This index is a superset of the primary key

74389. **comment:** ** The fts3 built-in tokenizers - "simple", "porter" and "icu"- are ** implemented in files fts3_tokenizer1.c, fts3_porter.c and fts3_icu.c ** respectively. The following three forward declarations are for functions ** declared in these files used to retrieve the respective implementations. ** ** Calling sqlite3Fts3SimpleTokenizerModule() sets the value pointed ** to by the argument to point to the "simple" tokenizer implementation. ** And so on.
label: code-design

74390. **comment:** Not used
label: code-design

74391. ORDER

74392. ** All unique filenames are held on a linked list headed by this ** variable:

74393. Check for integer primary key out of range

74394. Cursor pointing at blob row

74395. ** Generate code to return a single integer value.

74396. The definition of the module

74397. Wal frame containing data for iDbpage

74398. Table triggers are being coded for

74399. Page number in db file

74400. Mask of all well-ordered loops

74401. **** If we reach this point, flattening is permitted. ****

74402. SQLITE_ENABLE_EXPENSIVE_ASSERT

74403. xShmUnmap

74404. ** For each row modified during a session, there exists a single instance of ** this structure stored in a SessionTable.aChange[] hash table.

74405. tokenizer for inserts and queries

74406. ***** Continuing where we left off in btmutex.c *****

74407. The database connection to which it belongs

74408. Opcode: Blob P1 P2 * P4 * ** Synopsis: r[P2]=P4 (len=P1) ** ** P4 points to a blob of data P1 bytes long. Store this ** blob in register P2.

74409. **comment:** Remove the redundant segments from the %_data table
label: code-design

74410. Index of database holding TABLE

74411. synopsis: P3 columns in r[P2]

74412. Used to store transient return codes

74413. Phrase expression node

74414. SQLITE_OMIT_ATTACH

74415. Extract a leaf from the trunk

74416. Check that the trigger name is not reserved and that no trigger of the ** specified name exists

74417. Callback

74418. The backup process p has already copied page iPage. But now it ** has been modified by a transaction on the source pager. Copy ** the new data into the backup.

74419. ***** End of os_unix.c *****

74420. The initial database size

74421. Bind values to the DELETE statement. If conflict handling is required, ** bind values for all columns and set bound variable (nCol+1) to true. ** Or, if conflict handling is not required, bind just the PK column ** values and, if it exists, set (nCol+1) to false. Conflict handling ** is not required if: ** ** this is a patchset, or ** *(pbRetry==0), or ** * all columns of the table are PK columns (in this case there is ** no (nCol+1) variable to bind to).

74422. **comment:** ** Backwards Compatibility Hack: ** ** Historical versions of SQLite accepted strings as column names in ** indexes and PRIMARY KEY constraints and in UNIQUE constraints. Example: ** ** CREATE TABLE xyz(a,b,c,d,e,PRIMARY KEY('a'),UNIQUE('b','c' COLLATE trim) ** CREATE INDEX abc ON xyz('c','d' DESC,'e' COLLATE nocase DESC); ** ** This is goofy. But to preserve backwards compatibility we continue to ** accept it. This routine does the necessary conversion. It converts ** the expression given in its argument from a TK_STRING into a TK_ID ** if the expression is just a TK_STRING with an optional COLLATE clause. ** If the expression is anything other than TK_STRING, the expression is ** unchanged.
label: code-design

74423. No collation sequence of this type for this encoding is registered. ** Call the collation factory to see if it can supply us with one.

74424. **comment:** ** 2015-06-08 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This module contains C code that generates VDBE code used to process ** the WHERE clause of SQL statements. ** ** This file was originally part of where.c but was split out to improve ** readability and editability. This file contains utility routines for ** analyzing Expr objects in the WHERE clause.
label: code-design

74425. The GetProcAddressA() routine is only available on Windows CE.

74426. Integrity check context

74427.

74428. The page that contains a pointer to pDbPage

74429. **comment:** ** This function is exactly the same as sqlite3_create_function(), except ** that it is designed to be called by internal code. The difference is ** that if a malloc() fails in sqlite3_create_function(), an error code ** is returned and the mallocFailed flag cleared.
label: code-design

74430. column number to retrieve

74431. ** Make sure the number of reserved bits is the same in the destination ** pager as it is in the source. This comes up when a VACUUM changes the ** number of reserved bits to the "optimal" amount.

74432. (Phrases that will be loaded)^4.

74433. **comment:** ** Populate the low-level memory allocation function pointers in ** sqlite3GlobalConfig.m with pointers to the routines in this file.
label: code-design

74434. Length of zDb in bytes

74435. ** 2001 September 22 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This is the implementation of generic hash-tables ** used in SQLite.

74436. ***** Begin %parse_failure code *****

74437. Columns to search (NULL -> all columns)

74438. ** Close a tokenization cursor previously opened by a call to ** porterOpen() above.

74439. True (1) if iJD is valid

74440. **comment:** Number of unused bytes on page
label: code-design

74441. Leaf page (iLeafPgno+1)

74442. REINDEXEDESCAPEACHECHECKYBEFOREIGNOREGEXPLAINSTEADATABASESELECT

74443. ** Mark all NULL entries in the Object passed in as JNODE_REMOVE.

74444. BEGIN

74445. ** Invoke the pre-update hook. If this is an UPDATE or DELETE pre-update call, ** then cursor passed as the second argument should point to the row about ** to be updated or deleted. If the application calls sqlite3_prewrite_old(), ** the required value will be read from the row the cursor points to.

74446. ** CAPI3REF: Status Parameters for database connections ** KEYWORDS: {SQLITE_DBSTATUS options} ** These constants are the available integer "verbs" that can be passed as ** the second argument to the [sqlite3_db_status()] interface. ** New verbs may be added in future releases of SQLite. Existing verbs ** might be discontinued. Applications should check the return code from ** [sqlite3_db_status()] to make sure that the call worked. ** The [sqlite3_db_status()] interface will return a non-zero error code ** if a discontinued or unsupported verb is invoked. ** ** <dl> **
[[SQLITE_DBSTATUS_LOOKASIDE_USED]] ^(<dt>SQLITE_DBSTATUS_LOOKASIDE_USED</dt> ** <dd>This parameter returns the number of lookaside memory slots currently ** checked out.</dd>) ^ ** [[SQLITE_DBSTATUS_LOOKASIDE_HIT]]
^(<dt>SQLITE_DBSTATUS_LOOKASIDE_HIT</dt> ** <dd>This parameter returns the number malloc attempts that were ** satisfied using lookaside memory. Only the high-water value is meaningful; ** the current value is always zero.) ^ ** [[SQLITE_DBSTATUS_LOOKASIDE_MISS_SIZE]] **
^(<dt>SQLITE_DBSTATUS_LOOKASIDE_MISS_SIZE</dt> ** <dd>This parameter returns the number malloc attempts that might have ** been satisfied using lookaside memory but failed due to the amount of ** memory requested being larger than the lookaside slot size. ** Only the high-water value is meaningful; ** the current value is always zero.) ^ ** [[SQLITE_DBSTATUS_LOOKASIDE_MISS_FULL]] **
^(<dt>SQLITE_DBSTATUS_LOOKASIDE_MISS_FULL</dt> ** <dd>This parameter returns the number malloc attempts that might have ** been satisfied using lookaside memory but failed due to all lookaside ** memory already being in use. ** Only the high-water value is meaningful; ** the current value is always zero.) ^ ** [[SQLITE_DBSTATUS_CACHE_USED]] ^(<dt>SQLITE_DBSTATUS_CACHE_USED</dt> ** <dd>This parameter returns the approximate number of bytes of heap ** memory used by all pager caches associated with the database connection.) ^ ** ^The highwater mark associated with SQLITE_DBSTATUS_CACHE_USED is always 0. ** [[SQLITE_DBSTATUS_CACHE_USED_SHARED]] **
^(<dt>SQLITE_DBSTATUS_CACHE_USED_SHARED</dt> ** <dd>This parameter is similar to DBSTATUS_CACHE_USED, except that if a ** pager cache is shared between two or more connections the bytes of heap ** memory used by that pager cache is divided evenly between the attached ** connections.) ^ In other words, if none of the pager caches associated ** with the database connection are shared, this request returns the same ** value as DBSTATUS_CACHE_USED. Or, if one or more of the pager caches are ** shared, the value returned by this call will be smaller than that returned ** by DBSTATUS_CACHE_USED. ^The highwater mark associated with ** SQLITE_DBSTATUS_CACHE_USED_SHARED is always 0. ** [[SQLITE_DBSTATUS_STMT_USED]]
^(<dt>SQLITE_DBSTATUS_STMT_USED</dt> ** <dd>This parameter returns the approximate number of bytes of heap ** and lookaside memory used by all prepared statements associated with ** the database connection.) ^ ** ^The highwater mark associated with SQLITE_DBSTATUS_STMT_USED is always 0. ** <dd> ** ** [[SQLITE_DBSTATUS_CACHE_HIT]] ^(<dt>SQLITE_DBSTATUS_CACHE_HIT</dt> ** <dd>This parameter returns the number of pager cache hits that have ** occurred.) ^ ^The highwater mark associated with SQLITE_DBSTATUS_CACHE_HIT ** is always 0. ** </dd> ** **
[[SQLITE_DBSTATUS_CACHE_MISS]] ^(<dt>SQLITE_DBSTATUS_CACHE_MISS</dt> ** <dd>This parameter returns the number of pager cache misses that have ** occurred.) ^ ^The highwater mark associated with SQLITE_DBSTATUS_CACHE_MISS ** is always 0. ** </dd> ** **
[[SQLITE_DBSTATUS_CACHE_WRITE]] ^(<dt>SQLITE_DBSTATUS_CACHE_WRITE</dt> ** <dd>This parameter returns the number of dirty cache entries that have ** been written to disk. Specifically, the number of pages written to the ** wal file in wal mode databases, or the number of pages written to the ** database file in rollback mode databases. Any pages written as part of ** transaction rollback or database recovery operations are not included. ** If an IO or other error occurs while writing a page to disk, the effect ** on subsequent SQLITE_DBSTATUS_CACHE_WRITE requests is undefined.) ^ ^The ** highwater mark associated with SQLITE_DBSTATUS_CACHE_WRITE is always 0. ** </dd> ** ** [[SQLITE_DBSTATUS_DEFERRED_FKS]]
^(<dt>SQLITE_DBSTATUS_DEFERRED_FKS</dt> ** <dd>This parameter returns zero for the current value if and only if ** all foreign key constraints (deferred or immediate) have been ** resolved.) ^ ^The highwater mark is always 0. ** </dd> ** </dl>

74447. **comment:** ** Deallocate a register, making available for reuse for some other ** purpose. ** ** If a register is currently being used by the column cache, then ** the deallocation is deferred until the column cache line that uses ** the register becomes stale.
label: code-design

74448. Query prefix

74449. **comment:** The expression may actually be of the form (x, y) IN (SELECT...). ** In this case there is a separate term for each of (x) and (y). ** However, the nIn multiplier should only be applied once, not once ** for each such term. The following loop checks that pTerm is the ** first such term in use, and sets nIn back to 0 if it is not.
label: code-design

74450. ** Set an entry in the wal-index that will map database page number ** pPage into WAL frame iFrame.

74451. Skip partial indices for which the WHERE clause is not true

74452. ** PRAGMA temp_store_directory ** PRAGMA temp_store_directory = ""|"directory_name" ** ** Return or set the local value of the temp_store_directory flag. Changing ** the value sets a specific directory to be used for temporary files. ** Setting to a null string reverts to the default temporary directory search. ** If temporary directory is changed, then invalidateTempStorage. **

74453. SQL function return value

74454. Page-size field of journal header

74455. Return the next entry on the list

74456. **comment:** DISTINCT keyword not used
label: code-design

74457. ** This function is called before attempting a hot-journal rollback. It ** syncs the journal file to disk, then sets pPager->journalHdr to the ** size of the journal file so that the pager_playback() routine knows ** that the entire journal file has been synced. ** ** Syncing a hot-journal to disk before attempting to roll it back ensures ** that if a power-failure occurs during the rollback, the process that ** attempts rollback following system recovery sees the same journal ** content as this process. ** ** If everything goes as planned, SQLITE_OK is returned. Otherwise, ** an SQLite error code.

74458. Write the new new.* record. Consists of a copy of all values ** from the original old.* record, except for the PK columns, which ** are set to "undefined".

74459. Information for virtual tables

74460. Bytes of leaf page data so far

74461. The query under construction

74462. ** Allocate or deallocate a block of nReg consecutive registers.

74463. **comment:** Verify that there is enough key space remaining to avoid ** a buffer overread. The "d1+serial_type1+2" subexpression will ** always be greater than or equal to the amount of required key space. ** Use that approximation to avoid the more expensive call to ** sqlite3VdbeSerialTypeLen() in the common case.
label: code-design

74464. Implement a co-routine that will return a single row of the result ** set on each invocation.

74465. Current state of incremental merge
74466. NULL pBitvec tests
74467. iff(!pSorter->bUseThreads)
74468. isMutexInit
74469. FTS3 table snippet comes from
74470. Thread will use this task object
74471. ** Return the current file-size of an rbuVfs-file.
74472. New vdbe frame to execute in
74473. Mutex to protect pMain
74474. Pointer to aType[nField]
74475. Malloc'd buffer to load entire frame
74476. ***** End of trigger.c *****
74477. ** This routine is called when an INITIALLY IMMEDIATE or INITIALLY DEFERRED ** clause is seen as part of a foreign key definition. The isDeferred ** parameter is 1 for INITIALLY DEFERRED and 0 for INITIALLY IMMEDIATE. ** The behavior of the most recently created foreign key is adjusted ** accordingly.
74478. Current index in apPage[i]
74479. Array for converting from half-bytes (nybbles) into ASCII hex ** digits.
74480. ifdef SQLITE_NO_SYNC elif HAVE_FULLFSYNC
74481. Hash table key
74482. Conversion buffer
74483. Trash all content in the buffer being freed
74484. **comment:** ** CAPI3REF: Obtain Conflicting Row Values From A Changeset Iterator ** ** This function should only be used with iterator objects passed to a ** conflict-handler callback by [sqlite3changeset_apply()] with either ** [SQLITE_CHANGESET_DATA] or [SQLITE_CHANGESET_CONFLICT]. If this function ** is called on any other iterator, [SQLITE_MISUSE] is returned and *ppValue ** is set to NULL. ** ** Argument iVal must be greater than or equal to 0, and less than the number ** of columns in the table affected by the current change. Otherwise, ** [SQLITE_RANGE] is returned and *ppValue is set to NULL. ** ** If successful, this function sets *ppValue to point to a protected ** sqlite3_value object containing the iVal'th value from the ** "conflicting row" associated with the current conflict-handler callback ** and returns SQLITE_OK. ** ** If some other error occurs (e.g. an OOM condition), an SQLite error code ** is returned and *ppValue is set to NULL.
label: code-design
74485. ** Maximum pathname length (in bytes) for Win32. The MAX_PATH macro is in ** characters, so we allocate 4 bytes per character assuming worst-case of ** 4-bytes-per-character for UTF8.
74486. Set nSum to the number of distinct (iCol+1) field prefixes that ** occur in the stat4 table for this index. Set sumEq to the sum of ** the nEq values for column iCol for the same set (adding the value ** only once where there exist duplicate prefixes).
74487. pPage->nCell must be greater than zero. If this is the root-page ** the cursor would have been INVALID above and this for(;;) loop ** not run. If this is not the root-page, then the moveToChild() routine ** would have already detected db corruption. Similarly, pPage must ** be the right kind (index or table) of b-tree page. Otherwise ** a moveToChild() or moveToRoot() call would have detected corruption.
74488. Do not allow both MEMSYS5 and MEMSYS3 to be defined together. If they ** are, disable MEMSYS3
74489. **comment:** No page should ever be explicitly rolled back that is in use, except ** for page 1 which is held in use in order to keep the lock on the ** database active. However such a page may be rolled back as a result ** of an internal error resulting in an automatic call to ** sqlite3PagerRollback().
label: code-design
74490. Must be count(*)
74491. ** If this is a WAL database, store a pointer to pSnapshot. Next time a ** read transaction is opened, attempt to read from the snapshot it ** identifies. If this is not a WAL database, return an error.
74492. The parent table
74493. Coverage: it may be that this sqlite3_step() cannot fail. There ** is already a transaction open, so the prepared statement cannot ** throw an SQLITE_SCHEMA exception. The only database page the ** statement reads is page 1, which is guaranteed to be in the cache. ** And no memory allocations are required.
74494. Bit Field Type
74495. Bytes to append to the prefix
74496. ** CAPI3REF: Move a BLOB Handle to a New Row ** METHOD: sqlite3_blob ** ** ^This function is used to move an existing [BLOB handle] so that it points ** to a different row of the same database table. ^The new row is identified ** by the rowid value passed as the second argument. Only the row can be ** changed. ^The database, table and column on which the blob handle is open ** remain the same. Moving an existing [BLOB handle] to a new row is ** faster than closing the existing handle and opening a new one. ** ** ^The new row must meet the same criteria as for [sqlite3_blob_open()] - ** it must exist and there must be either a blob or text value stored in ** the nominated column.^ ^If the new row is not present in the table, or if ** it does not contain a blob or text value, or if another error occurs, an ** SQLite error code is returned and the blob handle is considered aborted. ** ^All subsequent calls to [sqlite3_blob_read()], [sqlite3_blob_write()] or ** [sqlite3_blob_reopen()] on an aborted blob handle immediately return ** SQLITE_ABORT. ^Calling [sqlite3_blob_bytes()] on an aborted blob handle ** always returns zero. ** ** ^This function sets the database handle error code and message.
74497. **comment:** This helper routine to saveAllCursors does the actual work of saving ** the cursors if and when a cursor is found that actually requires saving. ** The common case is that no cursors need to be saved, so this routine is ** broken out from its caller to avoid unnecessary stack pointer movement.
label: code-design
74498. Size of the cell content on the main b-tree page
74499. Return the size in bytes of a PCache object. Used to preallocate ** storage space.
74500. Type of RHS table. IN_INDEX_*
74501. PseudoTable input register
74502. The LIMIT clause will jump out of the loop for us
74503. **comment:** ** If the sqlite3PcacheFetch() routine is unable to allocate a new ** page because no clean pages are available for reuse and the cache ** size limit has been reached, then this routine can be invoked to ** try harder to allocate a page. This routine might invoke the stress ** callback to spill dirty pages to the journal. It will then try to ** allocate the new page and will only fail to allocate a new page on ** an OOM error. ** ** This routine should be invoked only after sqlite3PcacheFetch() fails.
label: code-design
74504. A SQLITE_FUNC_CONSTANT or _SLOCNG function
74505. Also an error
74506. ** Return the size of the Nth element of the cell array
74507. Spill is ok, but do not sync
74508. Size of azTb!Col[] array
74509. If a rollback journal is in use, them make sure the page that is about ** to change is in the rollback journal, or if the page is a new page off ** then end of the file, make sure it is marked as PGHDR_NEED_SYNC.
74510. The index that triggers the constraint
74511. Insert new divider cells into pParent.
74512. The methods of the json_each virtual table
74513. ** Set the journal-mode for this pager. Parameter eMode must be one of: ** ** PAGER_JOURNALMODE_DELETE ** PAGER_JOURNALMODE_TRUNCATE ** PAGER_JOURNALMODE_PERSIST ** PAGER_JOURNALMODE_OFF ** PAGER_JOURNALMODE_MEMORY ** PAGER_JOURNALMODE_WAL ** ** The journalmode is set to the value specified if the change is allowed. ** The change may be disallowed for the following reasons: ** ** An in-memory database can only have its journal_mode set to _OFF ** or _MEMORY. ** ** Temporary databases cannot have _WAL journalmode. ** ** The returned indicate the current (possibly updated) journal-mode.
74514. synopsis: r[P1]=r[P1]+P2
74515. True if the TEMPORARY keyword is present
74516. Find out what flags are present
74517. Number of bytes required for *p

74518. expr ::= expr NOT NULL
74519. The WHERE clause to be scanned
74520. ** Close a handle opened by an earlier call to sqlite3Fts5IndexOpen().
74521. 60
74522. The btree to be checked
74523. RAISE => ID
74524. ** 2014 May 31 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

74525. Wal handle to write to
74526. Name of database within pSrcDb
74527. Block id
74528. Pointer to sampled record
74529. Number of registers allocated
74530. The foreign key linking pSrc to pTab
74531. ** Add node pNode to the node hash table.
74532. Number of columns to test for changes
74533. 34
74534. **comment:** "Slow Change". Value constant during a ** single query - might change over time
label: code-design
74535. ** Attempt to release up to n bytes of non-essential memory currently ** held by SQLite. An example of non-essential memory is memory used to ** cache database pages that are not currently in use.
74536. If the pager is in persistent-journal mode, then the physical ** journal-file may extend past the end of the master-journal name ** and 8 bytes of magic data just written to the file. This is ** dangerous because the code to rollback a hot-journal file ** will not be able to find the master-journal name to determine ** whether or not the journal is hot. *** ** Easiest thing to do in this scenario is to truncate the journal ** file to the required size.
74537. The WhereScan object being initialized
74538. Statement holding lock on pIndex
74539. ** Versions of isspace(), isalnum() and isdigit() to which it is safe ** to pass signed char values.
74540. Step 1c
74541. nScratch
74542. Test this expression
74543. **comment:** ** CAPI3REF: Flags for the xAccess VFS method ** ** These integer constants can be used as the third parameter to ** the xAccess method of an [sqlite3_vfs] object. They determine ** what kind of permissions the xAccess method is looking for. ** With SQLITE_ACCESS_EXISTS, the xAccess method ** simply checks whether the file exists. ** With SQLITE_ACCESS_READWRITE, the xAccess method ** checks whether the named directory is both readable and writable ** (in other words, if files can be added, removed, and renamed within ** the directory). ** The SQLITE_ACCESS_READWRITE constant is currently used only by the ** [temp_store_directory pragma], though this could change in a future ** release of SQLite. ** With SQLITE_ACCESS_READ, the xAccess method ** checks whether the file is readable. The SQLITE_ACCESS_READ constant is ** currently unused, though it might be used in a future release of ** SQLite.
label: code-design
74544. ** This is a public wrapper for the winUtf8ToUnicode() function.
74545. Check for locking consistency
74546. Expr in which substitution occurs
74547. **comment:** 1 bit per page in the db (see above)
label: code-design
74548. Non-logarithmic stat1 data for this index
74549. Result of query pNextIdx
74550. ** CAPI3REF: Determine The Number Of Foreign Key Constraint Violations ** ** This function may only be called with an iterator passed to an ** SQLITE_CHANGESET_FOREIGN_KEY conflict handler callback. In this case ** it sets the output variable to the total number of known foreign key ** violations in the destination database and returns SQLITE_OK. *** ** In all other cases this function returns SQLITE_MISUSE.
74551. if SQLITE_OMIT_LOAD_EXTENSION is defined:
74552. Statement to write %_content table
74553. The WAL file to which we write
74554. ** When this function is called, *pp points to the first byte following a ** varint that is part of a doclist (or position-list, or any other list ** of varints). This function moves *pp to point to the start of that varint, ** and sets *pVal by the varint value. *** ** Argument pStart points to the first byte of the doclist that the ** varint is part of.
74555. Cursor number of cursor that does not need seeking
74556. Pointer to page cache object
74557. Page containing the cell
74558. The decimal point
74559. ** Check to see if zTabName is really the name of a pragma. If it is, ** then register an eponymous virtual table for that pragma and return ** a pointer to the Module object for the new virtual table.
74560. ** Read the first token from the nul-terminated string at *pz.
74561. ** Returns the character that should be used as the directory separator.
74562. Number of terms in the join
74563. 35
74564. Offset in aKey[] to read from
74565. Some combination of SQLITE_FUNC_*
74566. **comment:** ** Allocate memory. This routine is like sqlite3_malloc() except that it ** assumes the memory subsystem has already been initialized.
label: code-design
74567. Search constraints.
74568. ** Parse dates of the form *** ** YYYY-MM-DD HH:MM:SS ** YYYY-MM-DD HH:MM:SS ** YYYY-MM-DD HH:MM ** YYYY-MM-DD ** ** Write the result into the DateTime structure and return 0 ** on success and 1 if the input string is not a well-formed ** date.
74569. Number of bytes of header space
74570. ***** Begin file expr.c *****
74571. IMP: R-52476-28732
74572. Condition (1) is true
74573. **comment:** ** Include the malloc.h header file, if necessary. Also set define macro ** SQLITE_MALLOCOSIZE to the appropriate function name, which is _msize() ** for MSVC and malloc_usable_size() for most other systems (e.g. Linux). ** The memory size function can always be overridden manually by defining ** the macro SQLITE_MALLOCOSIZE to the desired function name.
label: code-design
74574. ** Return true if, for the purposes of tokenization, codepoint iCode is ** considered a token character (not a separator).
74575. Database holding shared-memory
74576. Value of 'name' column
74577. Total size of allocation
74578. ***** Begin file json1.c *****
74579. ** Allocate a new expression node from a zero-terminated token that has ** already been dequoted.
74580. Compute a safe address to jump to if we discover that the table for ** this loop is empty and can never contribute content.
74581. ** Populate the pCsr->iOffset and pCsr->szPage member variables. Based on ** the current value of pCsr->iPageno.

74582. ** SQLite value pRowid contains the rowid of a row that may or may not be ** present in the FTS3 table. If it is, delete it and adjust the contents ** of subsidiary data structures accordingly.

74583. ***** Begin file pragma.c *****

74584. tcons ::= PRIMARY KEY LP sortlist autoinc RP onconf

74585. offset to first byte to lock

74586. 54

74587. Hash table for in-memory data

74588. same as TK_ESCAPE

74589. String containing int array to decode

74590. The entire poslist will not fit on this leaf. So it needs ** to be broken into sections. The only qualification being ** that each varint must be stored contiguously.

74591. If control reaches this point, it means the transformation is ** one of the forms like "+NNN days".

74592. start tokenizing at the beginning

74593. First argument passed to xCallback()

74594. Name of database holding this table

74595. The row containing the glob

74596. List of active virtual machines

74597. ** Retrieve a record from the %_data table. ** ** If an error occurs, NULL is returned and an error left in the ** Fts5Index object.

74598. '/.' / or c-style comment

74599. 42

74600. ** Implementation of the offsets() function for FTS3

74601. ** Return the number of bytes required to store value iVal as a varint.

74602. ** This routine is called to implement sqlite3_wal_checkpoint() and ** related interfaces. ** ** Obtain a CHECKPOINT lock and then backfill as much information as ** we can from WAL into the database. ** ** If parameter xBusy is not NULL, it is a pointer to a busy-handler ** callback. In this case this function runs a blocking checkpoint.

74603. Pointer to bytes counter

74604. Number of tokens in each fragment

74605. If no preexisting index is available for the IN clause ** and IN_INDEX_NOOP is an allowed reply ** and the RHS of the IN operator is a list, not a subquery ** and the RHS is not constant or has two or fewer terms, ** then it is not worth creating an ephemeral table to evaluate ** the IN operator so return IN_INDEX_NOOP.

74606. ** If the SessionInput object passed as the only argument is a streaming ** object and the buffer is full, discard some data to free up space.

74607. ** The header string that appears at the beginning of every ** SQLite database.

74608. ** An instance of this structure contains information needed to generate ** code for a SELECT that contains aggregate functions. ** ** If Expr.op==TK_AGG_COLUMN or TK_AGG_FUNCTION then Expr.pAggInfo is a ** pointer to this structure. The Expr.iColumn field is the index in ** AggInfo.aCol[] or AggInfo.aFunc[] of information needed to generate ** code for that node. ** ** AggInfo.pGroupBy and AggInfo.aFunc.pExpr point to fields within the ** original Select structure that describes the SELECT statement. These ** fields do not need to be freed when deallocating the AggInfo structure.

74609. This call frees pShmNode if required

74610. Column reference

74611. String encoding (0 for blobs)

74612. skip trailing spaces

74613. rc==0 here means that one of the keys ran out of fields and ** all the fields up to that point were equal. Return the default_rc ** value.

74614. Re-open the databases.

74615. If the RBU database contains no data_xxx tables, declare the RBU ** update finished.

74616. If nRec is 0 and this rollback is of a transaction created by this ** process and if this is the final header in the journal, then it means ** that this part of the journal was being filled but has not yet been ** synced to disk. Compute the number of pages based on the remaining ** size of the file. ** ** The third term of the test was added to fix ticket #2565. ** When rolling back a hot journal, nRec==0 always means that the next ** chunk of the journal contains zero pages to be rolled back. But ** when doing a ROLLBACK and the nRec==0 chunk is the last chunk in ** the journal, it means that the journal might contain additional ** pages that need to be rolled back and that the number of pages ** should be computed based on the journal file size.

74617. Size of database in pages

74618. Pointers to the extension init functions

74619. **comment:** Do not need to test for a HAVING clause. If HAVING is present but ** there is no ORDER BY, we will get an error.
label: test

74620. Read the values returned by the SELECT into local variables.

74621. Info for single OR-term scan

74622. SQLITE_STATUS_SCRATCH_USED

74623. nHeap

74624. For each aggregate function

74625. SQLITE_WIN32_HAS_ANSI

74626. Statement to query database with

74627. If non-zero, transform OP_rowid to OP_AddImm(1)

74628. Value of 'pgSize' column

74629. Do any ON CASCADE, SET NULL or SET DEFAULT operations required to ** handle rows (possibly in other tables) that refer via a foreign key ** to the row just updated.

74630. Used to hold return values of fstat()

74631. The index whose column is to be loaded

74632. ** Register a trace function. The pArg from the previously registered trace ** is returned. ** ** A NULL trace function means that no tracing is executes. A non-NULL ** trace is a pointer to a function that is invoked at the start of each ** SQL statement.

74633. **comment:** Unused (padding) field
label: code-design

74634. Number of == or IN terms

74635. The look-ahead token

74636. Ok for sqlite3_open_v2()

74637. ** If the SELECT passed as the second argument has an associated WITH ** clause, pop it from the stack stored as part of the Parse object. ** ** This function is used as the xSelectCallback2() callback by ** sqlite3SelectExpand() when walking a SELECT tree to resolve table ** names and other FROM clause elements.

74638. Tokenizer to add exceptions to

74639. ** End of interface to code in fts5_index.c. *****

74640. A single column of result

74641. Byte offset of token within input text

74642. Number of mmap pages currently outstanding

74643. **comment:** Suppress warning about unused %extra_argument variable
label: code-design

74644. ** Each builtin conversion character (ex: the 'd' in "%d") is described ** by an instance of the following structure

74645. ** Change the opcode at addr into OP_Noop

74646. The virtual machine

74647. **comment:** ** Argument is a pointer to an Fts5Data structure that contains a leaf ** page. This macro evaluates to true if the leaf contains no terms, or ** false if it contains at least one term.
label: code-design

74648. ** Register hooks to call when sqlite3BeginBenignMalloc() and ** sqlite3EndBenignMalloc() are called, respectively.

74649. Table containing the row to be deleted

74650. Value for Fts3Phrase.iColumn
74651. ** Make every page in the cache clean.
74652. Maximum cost of a set of paths
74653. Cause p to be overwritten by pTemplate
74654. Return an error if we get here
74655. Insert the new divider cell into pParent.
74656. fts3_expr.c
74657. flags passed through to sqlite3_vfs.xOpen()
74658. Memory cell used to implement LEFT OUTER JOIN
74659. 0x
74660. Jump here if the key is OK
74661. ** Sort all elements on the list of RowSetEntry objects into order of ** increasing v.
74662. same as TK_STRING, synopsis: r[P2]=P4'
74663. Two or more AND-connected terms
74664. 1 for new.* ref mask, 0 for old.* ref mask
74665. Pointer to busy-handler function
74666. "ndlt" column of stat[34] entry
74667. Index of conflict record value to fetch
74668. RHS must not overlap with this mask
74669. ** Do not accept any file descriptor less than this value, in order to avoid ** opening database file using file descriptors that are commonly used for ** standard input, output, and error.
74670. 131
74671. If we created a new conch file (not just updated the contents of a ** valid conch file), try to match the permissions of the database
74672. Begin a transaction and take an exclusive lock on the main database ** file. This is done before the sqlite3BtreeGetPageSize(pMain) call below, ** to ensure that we do not try to change the page-size on a WAL database.
74673. ** PRAGMA [schema].locking_mode ** PRAGMA [schema].locking_mode = (normal|exclusive)
74674. Length of zSql in bytes.
74675. **comment:** **** Begin Proxy Locking

Proxy locking is a "uber-locking-method" in this sense: It uses the ** other locking methods on secondary lock files. Proxy locking is a ** meta-layer over top of the primitive locking implemented above. For ** this reason, the division that implements proxy locking is deferred ** until late in the file (here) after all of the other I/O methods have ** been defined - so that the primitive locking methods are available ** as services to help with the implementation of proxy locking. ** * * * * * The default locking schemes in SQLite use byte-range locks on the ** database file to coordinate safe, concurrent access by multiple readers ** and writers [<http://sqlite.org/lockingv3.html>]. The five file locking ** states (UNLOCKED, PENDING, SHARED, RESERVED, EXCLUSIVE) are implemented ** as POSIX read & write locks over fixed set of locations (via fcntl). ** on AFP and SMB only exclusive byte-range locks are available via fcntl ** with _IOWR('z', 23, struct ByteRangeLockPB2) to track the same 5 states. ** To simulate a F_RDLCK on the shared range, on AFP a randomly selected ** address in the shared range is taken for a SHARED lock, the entire ** shared range is taken for an EXCLUSIVE lock): *** * * * * *
PENDING_BYT 0x40000000 ** RESERVED_BYT 0x40000001 ** SHARED_RANGE 0x40000002 -> 0x40000200 *** * * * * * This works well on the local file system, but shows a nearly 100x ** slowdown in read performance on AFP because the AFP client disables ** the read cache when byte-range locks are present. Enabling the read ** cache exposes a cache coherency problem that is present on all OS X ** supported network file systems. NFS and AFP both observe the ** close-to-open semantics for ensuring cache coherency ** [http://nfs.sourceforge.net/#faq_a8], which does not effectively ** address the requirements for concurrent database access by multiple ** readers and writers ** [<http://www.nabble.com/SQLite-on-NFS-cache-coherency-td15655701.html>]. *** * * * * * To address the performance and cache coherency issues, proxy file locking ** changes the way database access is controlled by limiting access to a ** single host at a time and moving file locks off of the database file ** and onto a proxy file on the local file system. *** * * * * * Using proxy locks ** ----- * * * * * C APIs * * * * *
sqlite3_file_control(db, dbname, SQLITE_FCNTL_SET_LOCKPROXYFILE, ** <proxy_path> | ":auto"); ** sqlite3_file_control(db, dbname,
SQLITE_FCNTL_GET_LOCKPROXYFILE, ** &<proxy_path>); *** * * * * * SQL pragmas *** * * * * * PRAGMA [database].lock_proxy_file=<proxy_path> | :auto: **
PRAGMA [database].lock_proxy_file ** * * * * * Specifying ":auto:" means that if there is a conch file with a matching ** host ID in it, the proxy path in the conch file will be used, otherwise ** a proxy path based on the user's temp dir ** (via fcnstr(_CS_DARWIN_USER_TEMP_DIR,...)) will be used and the ** actual proxy file name is generated from the name and path of the ** database file. For example: *** * * * * * For database path "/Users/me/foo.db" ** The lock path will be "
<tmpdir>/sqliteplocks/_Users_me_foo.db:auto:" *** * * * * * Once a lock proxy is configured for a database connection, it can not ** be removed, however it may be switched to a different proxy path via ** the above APIs (assuming the conch file is not being held by another ** connection or process). *** * * * * * How proxy locking works ** ----- * * * * * Proxy file locking relies primarily on two new supporting files: *** * * * * * conch file to limit access to the database file to a single host ** at a time ** * * * * * proxy file to act as a proxy for the advisory locks normally ** taken on the database ** * * * * * The conch file - to use a proxy file, sqlite must first "hold the conch" ** by taking an sqlite-style shared lock on the conch file, reading the ** contents and comparing the host's unique host ID (see below) and lock ** proxy path against the values stored in the conch. The conch file is ** stored in the same directory as the database file and the file name ** is patterned after the database file name as ".<dbname>-conch". ** If the conch file does not exist, or its contents do not match the ** host ID and/or proxy path, then the lock is escalated to an exclusive ** lock and the conch file contents is updated with the host ID and proxy ** path and the lock is downgraded to a shared lock again. If the conch ** is held by another process (with a shared lock), the exclusive lock ** will fail and SQLITE_BUSY is returned. *** * * * * * The proxy file - a single-byte file used for all advisory file locks ** normally taken on the database file. This allows for safe sharing ** of the database file for multiple readers and writers on the same ** host (the conch ensures that they all use the same local lock file). *** * * * * * Requesting the lock proxy does not immediately take the conch, it is ** only taken when the first request to lock database file is made. ** This matches the semantics of the traditional locking behavior, where ** opening a connection to a database file does not take a lock on it. ** The shared lock and an open file descriptor are maintained until ** the connection to the database is closed. *** * * * * * The proxy file and the lock file are never deleted so they only need ** to be created the first time they are used. *** * * * * * Configuration options ** ----- * * * * * SQLITE_PREFER_PROXY_LOCKING ** * * * * * Database files accessed on non-local file systems are ** automatically configured for proxy locking, lock files are ** named automatically using the same logic as ** PRAGMA lock_proxy_file=":auto:" *** * * * * *
SQLITE_PROXY_DEBUG ** * * * * * Enables the logging of error messages during host id file ** retrieval and creation *** * * * * * LOCKPROXYDIR ** * * * * * Overrides the default directory used for lock proxy files that ** are named automatically via the ":auto:" setting *** * * * * * SQLITE_DEFAULT_PROXYDIR_PERMISSIONS *** * * * * * Permissions to use when creating a directory for storing the ** lock proxy files, only used when LOCKPROXYDIR is not set. *** * * * * * As mentioned above, when compiled with SQLITE_PREFER_PROXY_LOCKING, ** setting the environment variable SQLITE_FORCE_PROXY_LOCKING to 1 will ** force proxy locking to be used for every database file opened, and 0 ** will force automatic proxy locking to be disabled for all database ** files (explicitly calling the SQLITE_FCNTL_SET_LOCKPROXYFILE pragma or ** sqlite_file_control API is not affected by SQLITE_FORCE_PROXY_LOCKING).
label: code-design
74676. ** Check that the output buffer is large enough for the temporary file ** name in the following format: *** * * * * *
<temporary_directory>/etilqs_XXXXXXXXXXXXXX\0\0" *** * * * * * If not, return SQLITE_ERROR. The number 17 is used here in order to ** account for the space used by the 15 character random suffix and the ** two trailing NUL characters. The final directory separator character ** has already added if it was not already present.
74677. Size value to return
74678. Last entry in the stack
74679. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, append the string representation of integer iVal ** to the buffer. No nul-terminator is written. ** * * * * * If an OOM condition is encountered, set *pRc to SQLITE_NOMEM before ** returning.
74680. If a NEAR() clump or phrase may only match a specific set of columns, ** then an object of the following type is used to record the set of columns. ** Each entry in the aiCol[] array is a column that may be matched. *** * * * * * This object is used by fts5_expr.c and fts5_index.c.
74681. Locate the table in question
74682. '
74683. Now that the read-lock has been obtained, check that neither the ** value in the aReadMark[] array or the contents of the wal-index ** header have changed. *** * * * * * It is necessary to check that the wal-index header did not change ** between the time it was read and when the shared-lock was obtained ** on WAL_READ_LOCK(mxI) was obtained to account for the possibility ** that the log file may have been wrapped by a writer, or that frames ** that occur later in the log than pWal->hdr.mxFrame may have been ** copied into the database by a checkpoint. If either of these things ** happened, then reading the database with the current value of ** pWal->hdr.mxFrame risks reading a corrupted snapshot. So, retry ** instead. *** * * * * * Before checking that the live wal-index header has

not changed ** since it was read, set Wal.minFrame to the first frame in the wal ** file that has not yet been checkpointed. This client will not need ** to read any frames earlier than minFrame from the wal file - they ** can be safely read directly from the database file. ** ** Because a ShmBarrier() call is made between taking the copy of ** nBackfill and checking that the wal-header in shared-memory still ** matches the one cached in pWal->hdr, it is guaranteed that the ** checkpointer that set nBackfill was not working with a wal-index ** header newer than that cached in pWal->hdr. If it were, that could ** cause a problem. The checkpointer could omit to checkpoint ** a version of page X that lies before pWal->minFrame (call that version ** A) on the basis that there is a newer version (version B) of the same ** page later in the wal file. But if version B happens to like past ** frame pWal->hdr.mxFrame - then the client would incorrectly assume ** that it can read version A from the database file. However, since ** we can guarantee that the checkpointer that set nBackfill could not ** see any pages past pWal->hdr.mxFrame, this problem does not come up.

74684. First of real indices on the table

74685. The page getter method for when the pager is an error state

74686. ***** This routine is used inside of assert() only ***** *** Verify that the cursor holds the mutex on its BtShared

74687. FROM clause is not a view

74688. Operands are equal

74689. Now get the read-lock

74690. FROMFULLGLOBYIFISNULLORDERRESTRICTRIGHTROLLBACKROWUNIONUSING

74691. ** Generate code for an IN expression. ** ** x IN (SELECT ...) ** x IN (value, value, ...) ** ** The left-hand side (LHS) is a scalar or vector expression. The ** right-hand side (RHS) is an array of zero or more scalar values, or a ** subquery. If the RHS is a subquery, the number of result columns must ** match the number of columns in the vector on the LHS. If the RHS is ** a list of values, the LHS must be a scalar. ** ** The IN operator is true if the LHS value is contained within the RHS. ** The result is false if the LHS is definitely not in the RHS. The ** result is NULL if the presence of the LHS in the RHS cannot be ** determined due to NULLs. ** ** This routine generates code that jumps to destIfFalse if the LHS is not ** contained within the RHS. If due to NULLs we cannot determine if the LHS ** is contained in the RHS then jump to destIfNull. If the LHS is contained ** within the RHS then fall through. ** ** See the separate in-operator.md documentation file in the canonical ** SQLite source tree for additional information.

74692. Number of level-0 PMAs to merge

74693. ** Print tracing information for a SHIFT action

74694. ** This function is only called from within an assert() expression. It ** checks that the sqlite3.nTransaction variable is correctly set to ** the number of non-transaction savepoints currently in the ** linked list starting at sqlite3.pSavepoint. ** ** Usage: ** ** assert(checkSavepointCount(db));

74695. Things to be changed

74696. Length of zAbsoluteName string

74697. Total number of pages restored from journal

74698. Forward references to structures used for WAL

74699. comment: Optional %extra_argument parameter

label: code-design

74700. ** NOTE: All sub-platforms where the GetVersionEx[AW] functions are ** deprecated are always assumed to be based on the NT kernel.

74701. zPath is a path to a WAL or journal file. The following block derives ** the path to the associated database file from zPath. This block handles ** the following naming conventions: ** ** <path to db>-journal" ** "<path to db>-wal" ** "<path to db>-journalNN" ** "<path to db>-walNN" ** ** where NN is a decimal number. The NN naming schemes are ** used by the test_multiplex.c module.

74702. ** An instance of this object serves as a cursor into the rollback journal. ** The cursor can be either for reading or writing.

74703. ** This function checks if argument pFrom refers to a CTE declared by ** a WITH clause on the stack currently maintained by the parser. And, ** if currently processing a CTE expression, if it is a recursive ** reference to the current CTE. ** ** If pFrom falls into either of the two categories above, pFrom->pTab ** and other fields are populated accordingly. The caller should check ** (pFrom->pTab!=0) to determine whether or not a successful match ** was found. ** ** Whether or not a match is found, SQLITE_OK is returned if no error ** occurs. If an error does occur, an error message is stored in the ** parser and some error code other than SQLITE_OK returned.

74704. One byte past end of position list

74705. ** Open the dynamic library located at zPath and return a handle.

74706. Size of BLOB written into %_stat

74707. zSpan holds DB.TABLE.COLUMN

74708. If the first temporary PMA file has not been opened, open it now.

74709. xSFunc

74710. Second varint to store in hint

74711. The WHERE clause

74712. case 1 trumps case 3

74713. 4th argument to snippet()

74714. Verify that the MergeEngine is assigned to a single thread

74715. ** Finalize all statements and free all allocations that are specific to ** the current object (table/index pair).

74716. Set to true if a 'special' error

74717. True to trace VDBE execution

74718. True if cell coverage checking should be done

74719. True if pages are on backing store

74720. True if we have seen a malloc failure

74721. A slot for the record has already been allocated in the ** SQLITE_MASTER table. We just need to update that slot with all ** the information we've collected. ** ** The VM register number pParse->regRowid holds the rowid of an ** entry in the sqlite_master table that was created for this vtab ** by sqlite3StartTable().

74722. Test that this call is being made from within an SQLITE_DELETE or ** SQLITE_UPDATE pre-update callback, and that iIdx is within range.

74723. Need to make a copy of path if we extracted the value ** from the conch file or the path was allocated on the stack

74724. ** Turn the pExpr expression into an alias for the iCol-th column of the ** result set in pEList. ** ** If the reference is followed by a COLLATE operator, then make sure ** the COLLATE operator is preserved. For example: ** ** SELECT a+b, c+d FROM t1 ORDER BY 1 COLLATE nocase; ** ** Should be transformed into: ** ** SELECT a+b, c+d FROM t1 ORDER BY (a+b) COLLATE nocase; ** ** The nSubquery parameter specifies how many levels of subquery the ** alias is removed from the original expression. The usual value is ** zero but it might be more if the alias is contained within a subquery ** of the original expression. The Expr.op2 field of TK_AGG_FUNCTION ** structures must be increased by the nSubquery amount.

74725. Copy of fifth arg to _apply()

74726. Destination for coroutine A

74727. ***** End of vdbe.h *****

74728. Name of the column we are looking for

74729. ***** End of sqlite3ext.h *****

74730. In this case there is room on the trunk page to insert the page ** being freed as a new leaf. ** ** Note that the trunk page is not really full until it contains ** usableSize/4 - 2 entries, not usableSize/4 - 8 entries as we have ** coded. But due to a coding error in versions of SQLite prior to ** 3.6.0, databases with freelist trunk pages holding more than ** usableSize/4 - 8 entries will be reported as corrupt. In order ** to maintain backwards compatibility with older versions of SQLite, ** we will continue to restrict the number of entries to usableSize/4 - 8 ** for now. At some point in the future (once everyone has upgraded ** to 3.6.0 or later) we should consider fixing the conditional above ** to read "usableSize/4-2" instead of "usableSize/4-8". ** ** EVIDENCE-OF: R-19920-11576 However, newer versions of SQLite still ** avoid using the last six entries in the freelist trunk page array in ** order that database files created by newer versions of SQLite can be ** read by older versions of SQLite.

74731. Extra data associated with the callback

74732. ** Move the cursor down to a new child page. The newPgno argument is the ** page number of the child page to move to. ** ** This function returns SQLITE_CORRUPT if the page-header flags field of ** the new child page does not match the flags field of the parent (i.e. ** if an intkey page appears to be the parent of a non-intkey page, or ** vice-versa).

74733. Files only.

74734. Request a pointer to the required page from the VFS

74735. ** Generate VDBE code for a COMMIT or ROLLBACK statement. ** Code for ROLLBACK is generated if eType==TK_ROLLBACK. Otherwise ** code is generated for a COMMIT.

74736. ** Search the list of WhereLoops in *ppPrev looking for one that can be ** replaced by pTemplate. ** ** Return NULL if pTemplate does not belong on the WhereLoop list. ** In other words if pTemplate ought to be dropped from further consideration. ** ** If pX is a WhereLoop that pTemplate can replace, then return the ** link that points to pX. ** ** If pTemplate cannot replace any existing element of the list but needs ** to be added to the list as a new entry, then return a pointer to the ** tail of the list.

74737. Maximum allowed segments per level

74738. Trigger name for error reporting

74739. Begin the search at this node

74740. Now that no other errors can occur, finish filling in the BtCursor ** variables and link the cursor into the BtShared list.

74741. Checksum does not match

74742. ** Maximum pathname length (in chars) for Win32. This should normally be ** MAX_PATH.

74743. ** If this is a no-op implementation, implement everything as macros.

74744. Maximum allowed string or blob size

74745. Check that sqlite3VdbeUsesBtree() was not called on this VM

74746. User-data pointer

74747. Index interior and leaf nodes

74748. URI args to copy

74749. Figure out where to write the new Expr structure.

74750. Only valid in paren mode

74751. ** Macros to determine if a column is hidden. IsOrdinaryHiddenColumn() ** only works for non-virtual tables (ordinary tables and views) and is ** always false unless SQLITE_ENABLE_HIDDEN_COLUMNS is defined. The ** IsHiddenColumn() macro is general purpose.

74752. IMPLEMENTATION-OF: R-53536-42575 The sqlite3_libversion() function returns ** a pointer to the to the sqlite3_version[] string constant.

74753. 24

74754. !defined(SQLITE_OMIT_LOAD_EXTENSION)

74755. IMP: R-36257-52125

74756. Number of nested brackets

74757. Check for key annihilation.

74758. ** Formulate and prepare a SELECT statement to retrieve a row from table ** zTab in database zDb based on its primary key. i.e. ** ** SELECT * FROM zDb.zTab WHERE pk1 = ? AND pk2 = ? AND ...

74759. ** BEGIN CODE GENERATED BY tool/mkctime.tcl

74760. Pointer to new system call value

74761. Search for existing entry. If found, remove it from the hash table. ** Code below may link it back in.

74762. Name of database within pDestDb

74763. Source and destination indices

74764. flags field of new pages before shuffling

74765. Name of new table, or database name

74766. Number of columns stored in the index

74767. All filesystem writes are atomic

74768. ** Try to initialize the pCache->pFree and pCache->pBulk fields. Return ** true if pCache->pFree ends up containing one or more free pages.

74769. Write the %_segdir record.

74770. The default page size and offset

74771. ** During a rollback, when the pager reloads information into the cache ** so that the cache is restored to its original state at the start of ** the transaction, for each page restored this routine is called. ** ** This routine needs to reset the extra data section at the end of the ** page to agree with the restored data.

74772. ** This routine checks for a byte-order mark at the beginning of the ** UTF-16 string stored in *pMem. If one is present, it is removed and ** the encoding of the Mem adjusted. This routine does not do any ** byte-swapping, it just sets Mem.enc appropriately. ** ** The allocation (static, dynamic etc.) and encoding of the Mem may be ** changed by this function.

74773. Initialize the INTEGER value of a ROWID.

74774. ** A new segment has just been written to level iLvl of index structure ** pStruct. This function determines if any segments should be promoted ** as a result. Segments are promoted in two scenarios: ** ** a) If the segment just written is smaller than one or more segments ** within the previous populated level, it is promoted to the previous ** populated level. ** ** b) If the segment just written is larger than the newest segment on ** the next populated level, then that segment, and any other adjacent ** segments that are also smaller than the one just written, are ** promoted. ** ** If one or more segments are promoted, the structure object is updated ** to reflect this.

74775. ** These macros can be used to test, set, or clear bits in the ** Expr.flags field.

74776. ** Context object used by sqlite3Fts5StorageIntegrity().

74777. OUT: Starting offset of token

74778. Round up if true. Round down if false

74779. OUT: score for the cell

74780. ** Remove the entry with rowid=iDelete from the r-tree structure.

74781. 304

74782. pSqllogArg

74783. ** Return the VDBE address or label to jump to in order to break ** out of a WHERE loop.

74784. Index.aiColumn

74785. For looping over tables in pDb

74786. unknown or unsupported join type

74787. Used to full index in sorted order

74788. ** time(Timestring, MOD, MOD, ...) ** ** Return HH:MM:SS

74789. ** The minimum number of cells allowed for a node is a third of the ** maximum. In Gutman's notation: ** ** m = M/3 ** ** If an R*-tree "Reinsert" operation is required, the same number of ** cells are removed from the overfull node and reinserted into the tree.

74790. ** Change pMem so that its MEM_Str or MEM_Blob value is stored in ** MEM.zMalloc, where it can be safely written. ** ** Return SQLITE_OK on success or SQLITE_NOMEM if malloc fails.

74791. The database number

74792. 16

74793. RHS of comparison

74794. ** Return the estimated number of output rows from a WHERE clause

74795. Vdbe to add scanstatus entry to

74796. Parent node (or NULL for root node)

74797. Modify the CREATE TABLE statement.

74798. FTS3 vtab object

74799. Stop reporting errors after this many

74800. Move the *.oal file to *.wal. At this point connection p->db is ** holding a SHARED lock on the target database file (because it is ** in WAL mode). So no other connection may be writing the db. ** ** In order to ensure that there are no database readers, an EXCLUSIVE ** lock is obtained here before the *.oal is moved to *.wal.

74801. Drop the shadow tables

74802. ** Return the number of bytes allocated for the expression structure ** passed as the first argument. This is always one of EXPR_FULLSCREEN, ** EXPR_REDUCEDSIZE or EXPR_TOKENONLYSIZE.

74803. Created using CREATE INDEX

74804. Restriction 18.

74805. xDlOpen

74806. OUT: Size of position list in bytes

74807. Changegroup object to add changeset to
74808. A routine to convert a binary TK_IS or TK_ISNOT expression into a ** unary TK_ISNULL or TK_NOTNULL expression.
74809. The VDBE
74810. **comment:** The subquery whose WHERE clause is to be augmented
label: documentation
74811. **comment:** ** This structure is passed around through all the sanity checking routines ** in order to keep track of some global state information. *** ** The aRef[] array is allocated so that there is 1 bit for each page in ** the database. As the integrity-check proceeds, for each page used in ** the database the corresponding bit is set. This allows integrity-check to ** detect pages that are used twice and orphaned pages (both of which ** indicate corruption).
label: code-design
74812. **comment:** ** The macro unlikely() is a hint that surrounds a boolean ** expression that is usually false. Macro likely() surrounds ** a boolean expression that is usually true. These hints could, ** in theory, be used by the compiler to generate better code, but ** currently they are just comments for human readers.
label: code-design
74813. ** The following table is searched linearly, so it is good to put the ** most frequently used conversion types first.
74814. One of TK_INSERT, TK_UPDATE, TK_DELETE
74815. Iterator for skipping through tbl/idx
74816. An index associated with pTab
74817. The next entry is on the current page.
74818. The rowid searching for
74819. List to scan and in which to make substitutes
74820. ** The bitmask datatype defined below is used for various optimizations. *** ** Changing this from a 64-bit to a 32-bit type limits the number of ** tables in a join to 32 instead of 64. But it also reduces the size ** of the library by 738 bytes on ix86.
74821. Value for integer types
74822. Adjust nOut using stat3/stat4 data. Or, if there is no stat3/stat4 ** data, using some other estimate.
74823. One-time test address
74824. Search for an existing index that will work for this IN operator
74825. completely fill the hash, then just add it without
74826. Repeat the above with the next record to be updated, until ** all record selected by the WHERE clause have been updated.
74827. ** pMem currently only holds a string type (or maybe a BLOB that we can ** interpret as a string if we want to). Compute its corresponding ** numeric type, if has one. Set the pMem->u.r and pMem->u.i fields ** accordingly.
74828. Number of symbolic links followed so far
74829. 37
74830. ** Register the r-tree module with database handle db. This creates the ** virtual table module "rtree" and the debugging/analysis scalar ** function "rtreenode".
74831. 1st argument to snippet()
74832. Space available in zTemp[]
74833. Name of new function
74834. Write the node here
74835. ** Close a file.
74836. ** Register the fts3tok module with database connection db. Return SQLITE_OK ** if successful or an error code if sqlite3_create_module() fails.
74837. The new hash table
74838. Next are the tables used to determine what action to take based on the ** current state and lookahead token. These tables are used to implement ** functions that take a state number and lookahead value and return an ** action integer. *** ** Suppose the action integer is N. Then the action is determined as ** follows *** **
 $\leq N \leq \text{fts5YY_MAX_SHIFT}$ Shift N. That is, push the lookahead ** token onto the stack and goto state N. *** ** N between $\text{fts5YY_MIN_SHIFTREDUCE}$
 Shift to an arbitrary state then ** and $\text{fts5YY_MAX_SHIFTREDUCE}$ reduce by rule N- fts5YY_MIN_REDUCE ** and fts5YY_MAX_REDUCE *** ** N == $\text{fts5YY_ERROR_ACTION}$ A syntax error has occurred. *** ** N == $\text{fts5YY_ACCEPT_ACTION}$ The parser accepts its input. *** ** N == fts5YY_NO_ACTION No such action. Denotes unused ** slots in the $\text{fts5yy_action}[]$ table. *** ** The action table is constructed as a single large table named $\text{fts5yy_action}[]$. ** Given state S and lookahead X, the action is computed as either: *** ** (A) $N = \text{fts5yy_action}[\text{fts5yy_shift_ofst}[S] + X]$ ** (B) $N = \text{fts5yy_default}[S]$ *** ** The (A) formula is preferred. The B formula is used instead if: ** (1) The $\text{fts5yy_shift_ofst}[S]+X$ value is out of range, or ** (2) $\text{fts5yy_lookahead}[\text{fts5yy_shift_ofst}[S]+X]$ is not equal to X, or ** (3) $\text{fts5yy_shift_ofst}[S]$ equal $\text{fts5YY_SHIFT_USE_DFLT}$. ** (Implementation note: $\text{fts5YY_SHIFT_USE_DFLT}$ is chosen so that ** $\text{fts5YY_SHIFT_USE_DFLT}+X$ will be out of range for all possible lookaheads X. ** Hence only tests (1) and (2) need to be evaluated.) *** ** The formulas above are for computing the action when the lookahead is ** a terminal symbol. If the lookahead is a non-terminal (as occurs after ** a reduce action) then the $\text{fts5yy_reduce_ofst}[]$ array is used in place of ** the $\text{fts5yy_shift_ofst}[]$ array and $\text{fts5YY_REDUCE_USE_DFLT}$ is used in place of ** $\text{fts5YY_SHIFT_USE_DFLT}$. *** ** The following are the tables generated in this section: *** ** $\text{fts5yy_action}[]$ A single table containing all actions. ** $\text{fts5yy_lookahead}[]$ A table containing the lookahead for each entry in ** fts5yy_action . Used to detect hash collisions. ** $\text{fts5yy_shift_ofst}[]$ For each state, the offset into fts5yy_action for ** shifting terminals. ** $\text{fts5yy_reduce_ofst}[]$ For each state, the offset into fts5yy_action for ** shifting non-terminals after a reduce. ** $\text{fts5yy_default}[]$ Default action for each state. *** ***** Begin parsing tables *****
74839. ** Boolean values
74840. ** Begin constructing a new table representation in memory. This is ** the first of several action routines that get called in response ** to a CREATE TABLE statement. In particular, this routine is called ** after seeing tokens "CREATE" and "TABLE" and the table name. The isTemp ** flag is true if the table should be stored in the auxiliary database ** file instead of in the main database file. This is normally the case ** when the "TEMP" or "TEMPORARY" keyword occurs in between ** CREATE and TABLE. *** ** The new table record is initialized and put in pParse->pNewTable. ** As more of the CREATE TABLE statement is parsed, additional action ** routines will be called to add more information to this record. ** At the end of the CREATE TABLE statement, the sqlite3EndTable() routine ** is called to complete the construction of the new table record.
74841. OP_Explain for loop
74842. Rowid to delete
74843. All keys are integers
74844. tridxby ::= NOT INDEXED
74845. 3: (end_constraints && bRev && endEq)
74846. Right hand child expression
74847. 25-31
74848. nCol rounded up for alignment
74849. ** Construct a trigger step that implements an UPDATE statement and return ** a pointer to that trigger step. The parser calls this routine when it ** sees an UPDATE statement inside the body of a CREATE TRIGGER.
74850. Increment the value stored in the P2 operand of the OP_Rowid.
74851. ** This function is only called from within a pre-update-hook callback. ** It determines if the current pre-update-hook change affects the same row ** as the change stored in argument pChange. If so, it returns true. Otherwise ** if the pre-update-hook does not affect the same row as pChange, it returns ** false.
74852. Resolve the column names in all the expressions of the ** of the UPDATE statement. Also find the column index ** for each column to be updated in the pChanges array. For each ** column to be updated, make sure we have authorization to change ** that column.
74853. **comment:** The term being tested
label: test
74854. Number of elements in regKey for WITHOUT ROWID
74855. MEMORY
74856. ** Flush any buffered data to disk and clean up the PMA-writer object. ** The results of using the PMA-writer after this call are undefined. ** Return SQLITE_OK if flushing the buffered data succeeds or is not ** required. Otherwise, return an SQLite error code. *** ** Before returning, set *piEof to the offset immediately following the ** last byte written to the file.
74857. Repeat over compound terms
74858. 5 -> CONTENT
74859. iVersion

74860. Want a read-only lock?
74861. Flags associated with this schema
74862. ** Activate assert() only if SQLITE_TEST is enabled.
74863. Check that this is either a no-op (because the requested lock is ** already held), or one of the transitions that the busy-handler ** may be invoked during, according to the comment above ** sqlite3PagerSetBusyhandler().
74864. Cursor to grab old.* values from
74865. Timezone offset in minutes
74866. ** Save the positions of all cursors (except pExcept) that are open on ** the table with root-page iRoot. "Saving the cursor position" means that ** the location in the btree is remembered in such a way that it can be ** moved back to the same spot after the btree has been modified. This ** routine is called just before cursor pExcept is used to modify the ** table, for example in BtreeDelete() or BtreeInsert(). *** If there are two or more cursors on the same btree, then all such ** cursors should have their BTCF_Multiple flag set. The btreeCursor() ** routine enforces that rule. This routine only needs to be called in ** the uncommon case when pExpect has the BTCF_Multiple flag set. *** If pExpect!=NULL and if no other cursors are found on the same root-page, ** then the BTCF_Multiple flag on pExpect is cleared, to avoid another ** pointless call to this routine. *** Implementation note: This routine merely checks to see if any cursors ** need to be saved. It calls out to saveCursorsOnList() in the (unusual) ** event that cursors are in need to being saved.
74867. Length of the field
74868. True to set lock. False to clear lock
74869. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** ***** This file contains C code routines that are called by the parser ** to handle SELECT statements in SQLite.
74870. Constant used for this transform
74871. Because the parser constructs pTblName from a single identifier, ** sqlite3FixSrcList can never fail.
74872. Shared b-tree handle
74873. **comment:** ** Change the maximum number of in-memory pages that are allowed ** before attempting to recycle clean and unused pages.
label: code-design
74874. Insert the new row into the %_content table.
74875. ** Release all memory held by the SegmentWriter object passed as the ** first argument.
74876. Not previously opened
74877. synopsis: r[P2]=root iDb=P1
74878. Create a new cursor for the pragma virtual table
74879. Added by 3.3.13
74880. Bounding box coordinates
74881. 1 START:
74882. ** This routine is called from the TCL test function "translate_selftest". ** It checks that the primitives for serializing and deserializing ** characters in each encoding are inverses of each other.
74883. ** Generate or extend an 8 byte checksum based on the data in ** array aByte[] and the initial values of aIn[0] and aIn[1] (or ** initial values of 0 and 0 if aIn==NULL). *** The checksum is written back into aOut[] before returning. *** nByte must be a positive multiple of 8.
74884. ** Return the number of changes in the most recent call to sqlite3_exec().
74885. defined(_WRS_KERNEL)
74886. Close the underlying file handle
74887. Used to iterate through attached dbs
74888. Left subtree (smaller entries)
74889. The table from which records will be deleted
74890. ** CAPI3REF: Extract Metadata About A Column Of A Table ** METHOD: sqlite3 *** ^The sqlite3_table_column_metadata(X,D,T,C,...) routine returns ** information about column C of table T in database D ** on [database connection] X.)^ ^The sqlite3_table_column_metadata() ** interface returns SQLITE_OK and fills in the non-NULL pointers in ** the final five arguments with appropriate values if the specified ** column exists. ^The sqlite3_table_column_metadata() interface returns ** SQLITE_ERROR and if the specified column does not exist. ** ^If the column-name parameter to sqlite3_table_column_metadata() is a ** NULL pointer, then this routine simply checks for the existence of the ** table and returns SQLITE_OK if the table exists and SQLITE_ERROR if it ** does not. If the table name parameter T in a call to ** sqlite3_table_column_metadata(X,D,T,C,...) is NULL then the result is ** undefined behavior. *** ^The column is identified by the second, third and fourth parameters to ** this function. ^The second parameter is either the name of the database ** (i.e. "main", "temp", or an attached database) containing the specified ** table or NULL.)^ ^If it is NULL, then all attached databases are searched ** for the table using the same algorithm used by the database engine to ** resolve unqualified table references. *** ^The third and fourth parameters to this function are the table and column ** name of the desired column, respectively. *** ^Metadata is returned by writing to the memory locations passed as the 5th ** and subsequent parameters to this function. ^Any of these arguments may be ** NULL, in which case the corresponding element of metadata is omitted. *** ^</blockquote> ** <table border="1"> ** <tr> <th> Parameter <th> Output
Type <th> Description *** <tr><td> 5th <td> const char* <td> Data type ** <tr><td> 6th <td> const char* <td> Name of default collation sequence ** <tr><td> 7th <td> int <td> True if column has a NOT NULL constraint ** <tr><td> 8th <td> int <td> True if column is part of the PRIMARY KEY ** <tr><td> 9th <td> int <td> True if column is [INCREMENT] ** </table> ** </blockquote>)*** ^The memory pointed to by the character pointers returned for the ** declaration type and collation sequence is valid until the next ** call to any SQLite API function. *** ^If the specified table is actually a view, an [error code] is returned. *** ^If the specified column is "rowid", "oid" or "_rowid_" and the table ** is not a [WITHOUT ROWID] table and an ** [INTEGER PRIMARY KEY] column has been explicitly declared, then the output ** parameters are set for the explicitly declared column. ^If there is no ** [INTEGER PRIMARY KEY] column, then the outputs ** for the [rowid] are set as follows: *** <pre> ** data type: "INTEGER" ** collation sequence: "BINARY" ** not null: 0 ** primary key: 1 ** auto increment: 0 ** </pre>)*** ^This function causes all database schemas to be read from disk and ** parsed, if that has not already been done, and returns an error if ** any errors are encountered while loading the schema.
74891. Opcode: VColumn P1 P2 P3 * * * Synopsis: r[P3]=vcolumn(P2) ** Store the value of the P2-th column of ** the row of the virtual-table that the ** P1 cursor is pointing to into register P3.
74892. ***** Begin file vxworks.h *****
74893. Write the %_docszie record
74894. ** Number of entries in a hash table
74895. Number of leaf blocks added this trans
74896. Return a pointer to the buffer containing the record data for SorterRecord ** object p. Should be used as if: *** void *SRVAL(SorterRecord *p) { return (void*)&p[1]; }
74897. ** Return TRUE if the WHERE clause returns rows in ORDER BY order. ** Return FALSE if the output needs to be sorted.
74898. The new rowid
74899. Optional list of result-set columns
74900. Next function with same name
74901. P4 is a 32-bit signed integer
74902. Start of memory holding full result (or 0)
74903. Number of characters in this token
74904. ** Locate a VFS by name. If no name is given, simply return the ** first VFS on the list.
74905. ** Ensure that the SQLite statement handles required to update the ** target database object currently indicated by the iterator passed ** as the second argument are available.
74906. Make sure there is no ORDER BY or LIMIT clause on prior SELECTs. Only ** the last (right-most) SELECT in the series may have an ORDER BY or LIMIT.
74907. **comment:** ** The charMap() macro maps alphabetic characters (only) into their ** lower-case ASCII equivalent. On ASCII machines, this is just ** an upper-to-lower case map. On EBCDIC machines we also need ** to adjust the encoding. The mapping is only valid for alphabetics ** which are the only characters for which this feature is used. *** Used by keywordhash.h
label: code-design
74908. Not found. If the name we were looking for was temp.sqlite_master ** then change the name to sqlite_temp_master and try again.

74909. ** Query the BtShared.pHasContent vector. *** This function is called when a free-list leaf page is removed from the ** free-list for reuse. It returns false if it is safe to retrieve the ** page from the pager layer with the 'no-content' flag set. True otherwise.

74910. ** Connect to or create a statvfs virtual table.

74911. OP_OpenRead or OP_OpenWrite

74912. True for UPDATE that affects rowid

74913. ** Add a single OP_Explain instruction to the Vdbe to explain a simple ** count(*) query ("SELECT count(*) FROM pTab").

74914. ** Argument pIn points to the first character in what is expected to be ** a comma-separated list of SQL literals followed by a ')' character. ** If it actually is this, return a pointer to the ')'. Otherwise, return ** NULL to indicate a parse error.

74915. C-style comments

74916. if >0, constraint is part of argv to xFilter

74917. Pause time in microseconds

74918. ** json_replace(JSON, PATH, VALUE, ...) ** Replace the value at PATH with VALUE. If PATH does not already exist, ** this routine is a no-op. If JSON or PATH is malformed, throw an error.

74919. SQLITE_ENABLE_SNAPSHOT

74920. ** The interface to the code in fault.c used for identifying "benign" ** malloc failures. This is only present if SQLITE_UNTESTABLE ** is not defined.

74921. If we reach this point, the frame is valid. Return the page number ** and the new database size.

74922. Add the new session object to the linked list of session objects ** attached to database handle \$db. Do this under the cover of the db ** handle mutex.

74923. Case 3: (SELECT ... FROM ...) ** or: EXISTS(SELECT ... FROM ...) ** ** For a SELECT, generate code to put the values for all columns of ** the first row into an array of registers and return the index of ** the first register. ** ** If this is an EXISTS, write an integer 0 (not exists) or 1 (exists) ** into a register and return that register number. ** ** In both cases, the query is augmented with "LIMIT 1". Any ** preexisting limit is discarded in place of the new LIMIT 1.

74924. 150

74925. ** CAPI3REF: Device Characteristics *** The xDeviceCharacteristics method of the [sqlite3_io_methods] ** object returns an integer which is a vector of these ** bit values expressing I/O characteristics of the mass storage ** device that holds the file that the [sqlite3_io_methods] ** refers to. *** The SQLITE_IOCAP_ATOMIC property means that all writes of ** any size are atomic. The SQLITE_IOCAP_ATOMICnnn values ** mean that writes of blocks that are nnn bytes in size and ** are aligned to an address which is an integer multiple of ** nnn are atomic. The SQLITE_IOCAP_SAFE_APPEND value means ** that when data is appended to a file, the data is appended ** first then the size of the file is extended, never the other ** way around. The SQLITE_IOCAP_SEQUENTIAL property means that ** information is written to disk in the same order as calls ** to xWrite(). The SQLITE_IOCAP_POWERSAFE_OVERWRITE property means that ** after reboot following a crash or power loss, the only bytes in a ** file that were written at the application level might have changed ** and that adjacent bytes, even bytes within the same sector are ** guaranteed to be unchanged. The SQLITE_IOCAP_UNDELETABLE_WHEN_OPEN ** flag indicates that a file cannot be deleted when open. The ** SQLITE_IOCAP_IMMUTABLE flag indicates that the file is on ** read-only media and cannot be changed even by processes with ** elevated privileges.

74926. ** Estimate the total row width for a table.

74927. (319) database_kw_opt ::= DATABASE

74928. When converting from UTF-16, the maximum growth results from ** translating a 2-byte character to a 4-byte UTF-8 character. ** A single byte is required for the output string ** nul-terminator.

74929. The database the table is being created in

74930. Opcode: Add P1 P2 P3 * * * Synopsis: r[P3]=r[P1]+r[P2] ** ** Add the value in register P1 to the value in register P2 ** and store the result in register P3. ** If either input is NULL, the result is NULL.

74931. eidlist_opt

74932. **comment:** Enable the lookaside-malloc subsystem
label: code-design

74933. Memory mapped page to return

74934. Mask of operations to perform

74935. 144

74936. ** This function compares two index or table record keys in the same way ** as the sqlite3VdbeRecordCompare() routine. Unlike VdbeRecordCompare(), ** this function deserializes and compares values using the ** sqlite3VdbeSerialGet() and sqlite3MemCompare() functions. It is used ** in assert() statements to ensure that the optimized code in ** sqlite3VdbeRecordCompare() returns results with these two primitives. ** ** Return true if the result of comparison is equivalent to desiredResult. ** Return false if there is a disagreement.

74937. the LHS of the IN operator

74938. Decomposition of the WHERE clause

74939. Maximum string length

74940. 112

74941. 110

74942. Address of the output-B subroutine

74943. Number of components of the PRIMARY KEY

74944. Array of replacement values

74945. Makes OP_Jump below testable

74946. Original SQL parameter values

74947. If the database page size is not a power of two, or is greater than ** SQLITE_MAX_PAGE_SIZE, conclude that the WAL file contains no valid ** data. Similarly, if the 'magic' value is invalid, ignore the whole ** WAL file.

74948. No locking occurs in temporary files

74949. 291

74950. Number of usable bytes on a page

74951. Allowed values for sqlite3_userauth.authLevel

74952. Name of OS function that failed

74953. **comment:** ** The interface to the virtual-table mechanism defined above (back up ** to a comment remarkably similar to this one) is currently considered ** to be experimental. The interface might change in incompatible ways. ** If this is a problem for you, do not use the interface at this time. ** ** When the virtual-table mechanism stabilizes, we will declare the ** interface fixed, support it indefinitely, and remove this comment.
label: code-design

74954. refargs ::=

74955. ** 2015-06-08 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains C code to implement the TreeView debugging routines. ** These routines print a parse tree to standard output for debugging and ** analysis. ** ** The interfaces in this file is only available when compiling ** with SQLITE_DEBUG.

74956. **comment:** ** 2008 December 3 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This module implements an object we call a "RowSet". ** ** The RowSet object is a collection of rowids. Rowids ** are inserted into the RowSet in an arbitrary order. Inserts ** can be intermixed with tests to see if a given rowid has been ** previously inserted into the RowSet. ** ** After all inserts are finished, it is possible to extract the ** elements of the RowSet in sorted order. Once this extraction ** process has started, no new elements may be inserted. ** ** Hence, the primitive operations for a RowSet are: ** ** CREATE ** INSERT ** TEST ** SMALLEST ** DESTROY ** ** The CREATE and DESTROY primitives are the constructor and destructor, ** obviously. The INSERT primitive adds a new element to the RowSet. ** TEST checks to see if an element is already in the RowSet. SMALLEST ** extracts the least value from the RowSet. ** ** The INSERT primitive might allocate additional memory. Memory is ** allocated in chunks so most INSERTs do no allocation. There is an ** upper bound on the size of allocated memory. No memory is freed ** until DESTROY. ** ** The TEST primitive includes a "batch" number. The TEST primitive ** will only see elements that were inserted before the last change ** in the batch number. In other words, if an INSERT occurs between ** two TESTs where the TESTs have the same batch number, then the ** value added by the INSERT will not be visible to the second TEST. ** The initial batch number is zero, so if the very first TEST contains ** a non-zero batch number, it will see all prior INSERTs. ** ** No INSERTs may occur after a SMALLEST. An assertion will fail if ** that is

attempted. *** The cost of an INSERT is roughly constant. (Sometimes new memory ** has to be allocated on an INSERT.) The cost of a TEST with a new ** batch number is O(NlogN) where N is the number of elements in the RowSet. ** The cost of a TEST using the same batch number is O(logN). The cost ** of the first SMALLEST is O(NlogN). Second and subsequent SMALLEST ** primitives are constant time. The cost of DESTROY is O(N). *** TEST and SMALLEST may not be used by the same RowSet. This used to ** be possible, but the feature was not used, so it was removed in order ** to simplify the code.

label: code-design

74957. Linked list of coded triggers

74958. rowid <= ?

74959. Disable database changes

74960. If creating a master or main-file journal, this function will open ** a file-descriptor on the directory too. The first time unixSync() ** is called the directory file descriptor will be fsync(jed and close(jd).

74961. ** xSetOutputs callback used when the Fts5Colset object has nCol==0 (match ** against no columns at all).

74962. ** Add a docid/column/position entry to a PendingList structure. Non-zero ** is returned if the structure is sqlite3_reallocoed as part of adding ** the entry. Otherwise, zero. *** If an OOM error occurs, *pRc is set to SQLITE_NOMEM before returning. ** Zero is always returned in this case. Otherwise, if no OOM error occurs, ** it is set to SQLITE_OK.

74963. Number of user columns in table

74964. ** Swap the contents of buffer *p1 with that of *p2.

74965. The WhereLoop factory

74966. End of pWC->a[]

74967. True if pDest has a UNIQUE index

74968. Registers used by LIMIT and OFFSET

74969. Size of first block written to WAL file

74970. ** Are most of the Win32 ANSI APIs available (i.e. with certain exceptions ** based on the sub-platform)?

74971. If non-delete entry should be written

74972. Ticket #1863. To avoid creating security problems for older ** applications that relink against newer versions of SQLite, the ** ability to run load_extension is turned off by default. One ** must call either sqlite3_enable_load_extension(db) or ** sqlite3_db_config(db, SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION, 1, 0) ** to turn on extension loading.

74973. **comment:** ** This is part of the fts5_decode() debugging aid. *** Arguments pBlob/nBlob contain an "averages" record. This function ** appends a human-readable representation of record to the buffer passed ** as the second argument.

label: code-design

74974. Index into the cell pointer array

74975. Information about the ORDER BY clause

74976. OUT: Bitmask of terms to highlight

74977. The slot remains on the free-list. Reduce its size to account ** for the portion used by the new allocation.

74978. EVIDENCE-OF: R-31624-24737 RTRIM is like BINARY except that extra ** spaces at the end of either string do not change the result. In other ** words, strings will compare equal to one another as long as they ** differ only in the number of spaces at the end.

74979. SQLITE_LOCK_TRACE

74980. This call makes sure that the pager has the correct number of ** open savepoints. If the second parameter is greater than 0 and ** the sub-journal is not already open, then it will be opened here.

74981. Condition that must be true

74982. Size of TOP vector

74983. First list to merge

74984. ** Generate code that will evaluate all == and IN constraints for an ** index scan. *** For example, consider table t1(a,b,c,d,e,f) with index i1(a,b,c). ** Suppose the WHERE clause is this: a==5 AND b IN (1,2,3) AND c>5 AND c<10 ** The index has as many as three equality constraints, but in this ** example, the third "c" value is an inequality. So only two ** constraints are coded. This routine will generate code to evaluate ** a==5 and b IN (1,2,3). The current values for a and b will be stored ** in consecutive registers and the index of the first register is returned. *** In the example above nEq==2. But this subroutine works for any value ** of nEq including 0. If nEq==0, this routine is nearly a no-op. ** The only thing it does is allocate the pLevel->iMem memory cell and ** compute the affinity string. *** The nExtraReg parameter is 0 or 1. It is 0 if all WHERE clause constraints ** are == or IN and are covered by the nEq. nExtraReg is 1 if there is ** an inequality constraint (such as the "c>=5 AND c<10" in the example) that ** occurs after the nEq quality constraints. *** This routine allocates a range of nEq+nExtraReg memory cells and returns ** the index of the first memory cell in that range. The code that ** calls this routine will use that memory range to store keys for ** start and termination conditions of the loop. ** key value of the loop. If one or more IN operators appear, then ** this routine allocates an additional nEq memory cells for internal ** use. *** Before returning, *pzAff is set to point to a buffer containing a ** copy of the column affinity string of the index allocated using ** sqlite3DbMalloc(). Except, entries in the copy of the string associated ** with equality constraints that use BLOB or NONE affinity are set to ** SQLITE_AFF_BLOB. This is to deal with SQL such as the following: *** CREATE TABLE t1(a TEXT PRIMARY KEY, b); ** SELECT ... FROM t1 AS t2, t1 WHERE t1.a = t2.b; *** In the example above, the index on t1(a) has TEXT affinity. But since ** the right hand side of the equality constraint (t2.b) has BLOB/NONE affinity, ** no conversion should be attempted before using a t2.b value as part of ** a key to search the index. Hence the first byte in the returned affinity ** string in this example would be set to SQLITE_AFF_BLOB.

74985. expr ::= expr CONCAT expr

74986. Include ICU headers

74987. Bytes of data currently in memory

74988. Number of entries in aSegment[]

74989. (306) expr ::= term (OPTIMIZED OUT)

74990. Built-in typeof() function

74991. In some circumstances, we are able to run the xfer optimization ** only if the destination table is initially empty. Unless the ** SQLITE_Vacuum flag is set, this block generates code to make ** that determination. If SQLITE_Vacuum is set, then the destination ** table is always empty. *** Conditions under which the destination must be empty. *** (1) There is no INTEGER PRIMARY KEY but there are indices. ** (If the destination is not initially empty, the rowid fields ** of index entries might need to change.) *** (2) The destination has a unique index. (The xfer optimization ** is unable to test uniqueness.) *** (3) onError is something other than OE_Abort and OE_Rollback.

74992. ** A CellArray object contains a cache of pointers and sizes for a ** consecutive sequence of cells that might be held on multiple pages.

74993. Dispatch all page fetch requests to the appropriate getter method.

74994. In most cases, we should be able to acquire the lock we ** want without having to go through the ascending lock ** procedure that follows. Just be sure not to block.

74995. IN/OUT: Current offset

74996. Unimplemented version 3 methods

74997. Determine which, if any, tokens in the expression should be deferred.

74998. Ignore the error. Do not do the INSERT or UPDATE

74999. The new record

75000. ** It is already known that pMem contains an unterminated string. ** Add the zero terminator.

75001. file desc to assoc this lock with

75002. ** Invoke the conflict handler for the change that the changeset iterator ** currently points to. *** Argument eType must be either CHANGESET_DATA or CHANGESET_CONFLICT. ** If argument pbReplace is NULL, then the type of conflict handler invoked ** depends solely on eType, as follows: *** eType value Value passed to xConflict ** ----- ** CHANGESET_DATA CHANGESET_NOTFOUND ** CHANGESET_CONFLICT CHANGESET_CONSTRAINT *** Or, if pbReplace is not NULL, then an attempt is made to find an existing ** record with the same primary key as the record about to be deleted, updated ** or inserted. If such a record can be found, it is available to the conflict ** handler as the "conflicting" record. In this case the type of conflict ** handler invoked is as follows: *** eType value PK Record found? Value passed to xConflict ** ----- ** CHANGESET_DATA Yes CHANGESET_DATA ** CHANGESET_DATA No CHANGESET_NOTFOUND ** CHANGESET_CONFLICT Yes CHANGESET_CONFLICT ** CHANGESET_CONFLICT No CHANGESET_CONSTRAINT *** If pbReplace is not NULL, and a record with a matching PK is found, and ** the conflict handler function returns SQLITE_CHANGESET_REPLACE, *pbReplace ** is set to non-zero

before returning SQLITE_OK. *** If the conflict handler returns SQLITE_CHANGESET_ABORT, SQLITE_ABORT is ** returned. Or, if the conflict handler returns an invalid value, ** SQLITE_MISUSE. If the conflict handler returns SQLITE_CHANGESET OMIT, ** this function returns SQLITE_OK.

75003. ***** CUSTOM AUXILIARY FUNCTIONS *** Virtual table

implementations may overload SQL functions by implementing ** the sqlite3_module.xFindFunction() method.

75004. OP_OpenWrite cursors for the ONEPASS opt
75005. ** Array apCell[] contains pointers to nCell b-tree page cells. The ** szCell[] array contains the size in bytes of each cell. This function ** replaces the current contents of page pPg with the contents of the cell ** array. *** Some of the cells in apCell[] may currently be stored in pPg. This ** function works around problems caused by this by making a copy of any ** such cells before overwriting the page data. *** The MemPage.nFree field is invalidated by this function. It is the ** responsibility of the caller to set it correctly.

75006. Total number of phrase instances

75007. Last leaf page number in segment

75008. Size of buffer zTerm

75009. Name of the module for the virtual table

75010. If one value is a string and the other is a blob, the string is less. ** If both are strings, compare using the collating functions.

75011. Calculate the average document length for this FTS5 table

75012. 80

75013. Additional values that can be added to the sync_flags argument of ** sqlite3WalFrames():

75014. Lock the PENDING_LOCK byte if we need to acquire a PENDING lock or ** a SHARED lock. If we are acquiring a SHARED lock, the acquisition of ** the PENDING_LOCK byte is temporary.

75015. Size of aIndex[]

75016. Cursor number to use

75017. Open transactions both databases. The *-oal file is opened or ** created at this point.

75018. ** Argument pLevel describes a strategy for scanning table pTab. This ** function appends text to pStr that describes the subset of table ** rows scanned by the strategy in the form of an SQL expression. *** For example, if the query: ** ** SELECT * FROM t1 WHERE a=1 AND b>2; ** ** is run and there is an index on (a, b), then this function returns a ** string similar to: *** "a=? AND b>?"

75019. How many seg-readers to advance

75020. ** Zero the contents of the buffer object. But do not free the associated ** memory allocation.

75021. IN/OUT: Tokens in phrase of *paPoslist

75022. Level to add scanstatus() entry for

75023. Free any outstanding Savepoint structures.

75024. OUT: Write the answer here

75025. The SELECT statement being coded.

75026. ** An instance of the following object is used to record information about ** how to process the DISTINCT keyword, to simplify passing that information ** into the selectInnerLoop() routine.

75027. If there is no type specified, columns have the default affinity ** 'BLOB'.

75028. **comment:** Read and write the xxx_rowid table

label: code-design

75029. Check if any samples from the aBest[] array should be pushed ** into IndexSample.a[] at this point.

75030. **comment:** *** CAPI3REF: Mutex Handle *** The mutex module within SQLite defines [sqlite3_mutex] to be an ** abstract type for a mutex object. The SQLite core never looks ** at the internal representation of an [sqlite3_mutex]. It only ** deals with pointers to the [sqlite3_mutex] object. *** Mutexes are created using [sqlite3_mutex_alloc()].

label: code-design

75031. ** Return the name of a wildcard parameter. Return NULL if the index ** is out of range or if the wildcard is unnamed. *** The result is always UTF-8.

75032. The complete select statement being coded

75033. Set to update db->lastRowid

75034. Sync the db file if currently doing an incremental checkpoint

75035. 188

75036. Pointer to Fts5Global object

75037. ** Finalize the statement handle at pCsr->pStmt. *** Or, if that statement handle is one created by fts3CursorSeekStmt(), ** and the Fts3Table.pSeekStmt slot is currently NULL, save the statement ** pointer there instead of finalizing it.

75038. ** Convert zNum to a 64-bit signed integer. zNum must be decimal. This ** routine does *not* accept hexadecimal notation. *** If the zNum value is representable as a 64-bit two's-complement ** integer, then write that value into *pNum and return 0. *** If zNum is exactly 9223372036854775808, return 2. This special ** case is broken out because while 9223372036854775808 cannot be a ** signed 64-bit integer, its negative -9223372036854775808 can be. *** If zNum is too big for a 64-bit integer and is not ** 9223372036854775808 or if zNum contains any non-numeric text, ** then return 1. *** length is the number of bytes in the string (bytes, not characters). ** The string is not necessarily zero-terminated. The encoding is ** given by enc.

75039. 1490

75040. Low-level mutex interface

75041. expr ::= CAST LP expr AS typetoken RP

75042. Look up the table being altered.

75043. ** Allocate or return the aggregate context for a user function. A new ** context is allocated on the first call. Subsequent calls return the ** same context that was returned on prior calls.

75044. The module for this tokenizer

75045. ** Read the last entry (most recently pushed) from the hint blob *pHint ** and then remove the entry. Write the two values read to *piAbsLevel and ** *pnInput before returning. *** If no error occurs, return SQLITE_OK. If the hint blob in *pHint does ** not contain at least two valid varints, return SQLITE_CORRUPT_VTAB.

75046. Restriction 17: If the sub-query is a compound SELECT, then it must ** use only the UNION ALL operator. And none of the simple select queries ** that make up the compound SELECT are allowed to be aggregate or distinct ** queries.

75047. True for a savepoint rollback

75048. The parse context of the CREATE TRIGGER statement

75049. 2nd parameter to sqlite3_bind out of range

75050. **comment:** Opcodes that are used as the bottom of a loop (OP_Next, OP_Prev, ** OP_VNext, or OP_SorterNext) all jump here upon ** completion. Check to see if sqlite3_interrupt() has been called ** or if the progress callback needs to be invoked. *** This code uses unstructured "goto" statements and does not look clean. ** But that is not due to sloppy coding habits. The code is written this ** way for performance, to avoid having to run the interrupt and progress ** checks on every opcode. This helps sqlite3_step() to run about 1.5% ** faster according to "valgrind --tool=cachegrind".

label: code-design

75051. Shared memory associated with this inode

75052. Expr. to advance to next matching row

75053. Root page of table

75054. ** This is called to code the required FOR EACH ROW triggers for an operation ** on table pTab. The operation to code triggers for (INSERT, UPDATE or DELETE) ** is given by the op parameter. The tr_tm parameter determines whether the ** BEFORE or AFTER triggers are coded. If the operation is an UPDATE, then ** parameter pChanges is passed the list of columns being modified. *** If there are no triggers that fire at the specified time for the specified ** operation on pTab, this function is a no-op. *** The reg argument is the address of the first in an array of registers ** that contain the values substituted for the new.* and old.* references ** in the trigger program. If N is the number of columns in table pTab ** (a copy of pTab->nCol), then registers are populated as follows: *** Register Contains ** ----- ** reg+0 OLD.rowid ** reg+1 OLD.* value of left-most column of pTab ** ** reg+N OLD.* value of right-most column of pTab ** reg+N+1 NEW.rowid ** reg+N+2 OLD.* value of left-most column of pTab ** ** reg+N+N+1 NEW.* value of right-most column of pTab ** ** For ON DELETE triggers, the registers containing the NEW.* values will ** never be accessed by the trigger program, so they are not allocated or ** populated by the caller (there is no data to populate them with anyway). ** Similarly, for ON INSERT triggers the values stored in the OLD.* registers ** are never accessed, and so are not allocated by the caller. So, for an ** ON INSERT trigger, the value passed to this function as

parameter reg ** is not a readable register, although registers (reg+N) through ** (reg+N+N+1) are. *** Parameter orconf is the default conflict resolution algorithm for the ** trigger program to use (REPLACE, IGNORE etc.). Parameter ignoreJump ** is the instruction that control should jump to if a trigger program ** raises an IGNORE exception.

75055. **comment:** ** The SQLITE_AFF_MASK values masks off the significant bits of an ** affinity value.

label: code-design

75056. INSERT_CONTENT

75057. ***** sqlite3_value_***** The following routines extract information from a Mem or sqlite3_value ** structure.

75058. ** This is a magic string that appears at the beginning of every ** SQLite database in order to identify the file as a real database. *** You can change this value at compile-time by specifying a ** -DSQLITE_FILE_HEADER="..." on the compiler command-line. The ** header must be exactly 16 bytes including the zero-terminator so ** the string itself should be 15 characters long. If you change ** the header, then your custom library will not be able to read ** databases generated by the standard tools and the standard tools ** will not be able to read databases created by your custom library.

75059. one of TK_DELETE, TK_INSERT, TK_UPDATE

75060. Built-in coalesce() or ifnull()

75061. Amount of payload held locally, not on overflow

75062. ** Deallocate and destroy a parser. Destructors are called for ** all stack elements before shutting the parser down. *** If the fts5YYPARSEFREE NEVERNULL macro exists (for example because it ** is defined in a %include section of the input grammar) then it is ** assumed that the input pointer is never NULL.

75063. List of names of LHS of the assignment

75064. ** CAPI3REF: Authorizer Action Codes *** The [sqlite3_set_authorizer()] interface registers a callback function ** that is invoked to authorize certain SQL statement actions. The ** second parameter to the callback is an integer code that specifies ** what action is being authorized. These are the integer action codes that ** the authorizer callback may be passed. *** These action code values signify what kind of operation is to be ** authorized. The 3rd and 4th parameters to the authorization ** callback function will be parameters or NULL depending on which of these ** codes is used as the second parameter. ^The 5th parameter to the ** authorizer callback is the name of the database ("main", "temp", ** etc.) if applicable.^ ^The 6th parameter to the authorizer callback ** is the name of the inner-most trigger or view that is responsible for ** the access attempt or NULL if this access attempt is directly from ** top-level SQL code.

75065. Fall through for the RTREE_EQ case

75066. End if(affinity_ok)

75067. ** Equivalent to: *** char *fts5EntryKey(Fts5HashEntry *pEntry){ return zKey; }

75068. Check the table schemas match

75069. New object to return

75070. ** Implementation of the instr() function. *** instr(haystack,needle) finds the first occurrence of needle ** in haystack and returns the number of previous characters plus 1, ** or 0 if needle does not occur within haystack. *** If both haystack and needle are BLOBS, then the result is one more than ** the number of bytes in haystack prior to the first occurrence of needle, ** or 0 if needle never occurs in haystack.

75071. The database file

75072. Verify that the database still has the same name as it did when ** it was originally opened.

75073. 7: (start_constraints && startEq && bRev)

75074. Iterator object

75075. ** Return a pointer to the name of a variable in the given VList that ** has the value iVal. Or return a NULL if there is no such variable in ** the list

75076. Available backtrace slots

75077. **comment:** ** Exactly one of the following macros must be defined in order to ** specify which memory allocation subsystem to use. ***
SQLITE_SYSTEM_MALLOC // Use normal system malloc() ** SQLITE_WIN32_MALLOC // Use Win32 native heap API ** SQLITE_ZERO_MALLOC // Use a stub allocator that always fails ** SQLITE_MEMDEBUG // Debugging version of system malloc() *** On Windows, if the SQLITE_WIN32_MALLOC_VALIDATE macro is defined and the ** assert() macro is enabled, each call into the Win32 native heap subsystem ** will cause HeapValidate to be called. If heap validation should fail, an ** assertion will be triggered. *** If none of the above are defined, then set SQLITE_SYSTEM_MALLOC as ** the default.

label: code-design

75078. Index in p->aIndex[]

75079. Pointer to blob containing changeset

75080. Index in sqlite3.aDb[] of database holding pTab

75081. The shared memory being locked

75082. Input data

75083. Fill in the header.

75084. Create the SELECT statement to read keys from data_xxx

75085. ** CAPI3REF: Generate A Changeset From A Session Object *** Obtain a changeset containing changes to the tables attached to the ** session object passed as the first argument. If successful, ** set *ppChangeset to point to a buffer containing the changeset ** and *pnChangeset to the size of the changeset in bytes before returning ** SQLITE_OK. If an error occurs, set both *ppChangeset and *pnChangeset to ** zero and return an SQLite error code. *** A changeset consists of zero or more INSERT, UPDATE and/or DELETE changes, ** each representing a change to a single row of an attached table. An INSERT ** change contains the values of each field of a new database row. A DELETE ** contains the original values of each field of a deleted database row. An ** UPDATE change contains the original values of each field of an updated ** database row along with the updated values for each updated non-primary-key ** column. It is not possible for an UPDATE change to represent a change that ** modifies the values of primary key columns. If such a change is made, it ** is represented in a changeset as a DELETE followed by an INSERT. *** Changes are not recorded for rows that have NULL values stored in one or ** more of their PRIMARY KEY columns. If such a row is inserted or deleted, ** no corresponding change is present in the changesets returned by this ** function. If an existing row with one or more NULL values stored in ** PRIMARY KEY columns is updated so that all PRIMARY KEY columns are non-NUL, ** only an INSERT is appears in the changeset. Similarly, if an existing row ** with non-NUL PRIMARY KEY values is updated so that one or more of its ** PRIMARY KEY columns are set to NULL, the resulting changeset contains a ** DELETE change only. *** The contents of a changeset may be traversed using an iterator created ** using the [sqlite3changeset_start()] API. A changeset may be applied to ** a database with a compatible schema using the [sqlite3changeset_apply()] ** API. *** Within a changeset generated by this function, all changes related to a ** single table are grouped together. In other words, when iterating through ** a changeset or when applying a changeset to a database, all changes related ** to a single table are processed before moving on to the next table. Tables ** are sorted in the same order in which they were attached (or auto-attached) ** to the sqlite3_session object. The order in which the changes related to ** a single table are stored is undefined.

*** Following a successful call to this function, it is the responsibility of ** the caller to eventually free the buffer that *ppChangeset points to using ** [sqlite3_free()]. *** <h3>Changeset Generation</h3> *** Once a table has been attached to a session object, the session object ** records the primary key values of all new rows inserted into the table. ** It also records the original primary key and other column values of any ** deleted or updated rows. For each unique primary key value, data is only ** recorded once - the first time a row with said primary key is inserted, ** updated or deleted in the lifetime of the session. **** There is one exception to the previous paragraph: when a row is inserted, ** updated or deleted, if one or more of its primary key columns contain a ** NULL value, no record of the change is made. *** The session object therefore accumulates two types of records - those ** that consist of primary key values only (created when the user inserts ** a new record) and those that consist of the primary key values and the ** original values of other table columns (created when the users deletes ** or updates a record). *** When this function is called, the requested changeset is created using ** both the accumulated records and the current contents of the database ** file. Specifically: *** ** For each record generated by an insert, the database is queried ** for a row with a matching primary key. If one is found, an INSERT ** change is added to the changeset. If no such row is found, no change ** is added to the changeset. ** For each record generated by an update or delete, the database is ** queried for a row with a matching primary key. If such a row is ** found and one or more of the non-primary key fields have been ** modified from their original values, an UPDATE change is added to ** the changeset. Or, if no such row is found in the table, a DELETE ** change is added to the changeset. If there is a row with a matching ** primary key in the database, but all fields contain their original ** values, no change is added to the changeset. ** ** This means, amongst other things, that if a row is inserted and then later ** deleted while a session object is active, neither the insert nor the delete ** will be present in the changeset. Or if a row is deleted and then later a ** row with the same primary key values inserted while a session object is ** active, the resulting changeset will contain an UPDATE change instead of ** a DELETE and an INSERT. *** When a session object is disabled (see the [sqlite3session_enable()] API), ** it does not accumulate records when rows are inserted, updated or deleted. ** This may appear to have some counter-intuitive effects if a single row ** is written to more than once during a session. For example, if a row ** is inserted while a session object is enabled, then later deleted while ** the same session object is disabled, no INSERT record will appear in the ** changeset, even though the delete took

place while the session was disabled. ** Or, if one field of a row is updated while a session is disabled, and ** another field of the same row is updated while the session is enabled, the ** resulting changeset will contain an UPDATE change that updates both fields.

75086. Page being freed. May be NULL.

75087. The 6th parameter to db->xAuth callbacks

75088. VACUUM

75089. A sorting index used to test for distinctness

75090. Local copy of initial highlight-mask

75091. yymsp[-1].minor.yy0-overwrite-yymsp[0].minor.yy0

75092. ** Obtain a reference to an r-tree node.

75093. Reinstall the LIKE and GLOB functions. The variant of LIKE ** used will be case sensitive or not depending on the RHS.

75094. SQLITE_AFF_NUMERIC

75095. **comment:** Set the magic to VDBE_MAGIC_RUN sooner rather than later.

label: code-design

75096. Figure out the best index to use to search a pragma virtual table. ** ** There are not really any index choices. But we want to encourage the ** query planner to give == constraints on as many hidden parameters as ** possible, and especially on the first hidden parameter. So return a ** high cost if hidden parameters are unconstrained.

75097. If the page being moved is dirty and has not been saved by the latest ** savepoint, then save the current contents of the page into the ** sub-journal now. This is required to handle the following scenario: ** ** BEGIN; ** <journal page X, then modify it in memory> ** SAVEPOINT one; ** <Move page X to location Y> ** ROLLBACK TO one; ** ** If page X were not written to the sub-journal here, it would not ** be possible to restore its contents when the "ROLLBACK TO one" ** statement were processed. ** ** subjournalPage() may need to allocate space to store pPg->pjno into ** one or more savepoint bitvecs. This is the reason this function ** may return SQLITE_NOMEM.

75098. Number of unixShm objects pointing to this

75099. ** Make sure the TEMP database is open and available for use. Return ** the number of errors. Leave any error messages in the pParse structure.

75100. Records are always returned in ascending order of (name, path). ** If this will satisfy the client, set the orderByConsumed flag so that ** SQLite does not do an external sort.

75101. Last insert rowid (sqlite3.lastRowid)

75102. ** Return the full pathname of the underlying database file. Return ** an empty string if the database is in-memory or a TEMP database. ** ** The pager filename is invariant as long as the pager is ** open so it is safe to access without the BtShared mutex.

75103. Map from parent key columns to child table columns

75104. key_opt ::= KEY expr

75105. The record flows over onto one or more overflow pages. In ** this case the whole cell needs to be parsed, a buffer allocated ** and accessPayload() used to retrieve the record into the ** buffer before VdbeRecordCompare() can be called. ** ** If the record is corrupt, the xRecordCompare routine may read ** up to two varints past the end of the buffer. An extra 18 ** bytes of padding is allocated at the end of the buffer in ** case this happens.

75106. 1270

75107. ** The page that pCur currently points to has just been modified in ** some way. This function figures out if this modification means the ** tree needs to be balanced, and if so calls the appropriate balancing ** routine. Balancing routines are: ** ** balance_quick() ** balance_deeper() ** balance_nonroot()

75108. Loop through the returned columns. Set nStr to the number of bytes of ** space required to store a copy of each column name, including the ** nul-terminator byte.

75109. True for first term on page

75110. ** 2008 June 13 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file contains definitions of global variables and constants.

75111. **comment:** ** hwtime.h contains inline assembler code for implementing ** high-performance timing routines.

label: code-design

75112. ***** End of sqlite3session.h *****

75113. ** Unlock a file region.

75114. Check if this is to be a periodic sample. If so, add it.

75115. Step 5. If a usable page buffer has still not been found, ** attempt to allocate a new one.

75116. IF => ID

75117. Bytes of data

75118. ** Count the number of fullsyncs and normal syncs. This is used to test ** that syncs and fullsyncs are occurring at the right times.

75119. lastErrno set by seekAndRead

75120. ** Object used to represent a single file opened and mmapmed to provide ** shared memory. When multiple threads all reference the same ** log-summary, each thread has its own winFile object, but they all ** point to a single instance of this object. In other words, each ** log-summary is opened only once per process. ** ** winShmMutexHeld() must be true when creating or destroying ** this object or while reading or writing the following fields: ** ** nRef ** pNext ** ** The following fields are read-only after the object is created: ** ** fid ** zFilename ** ** Either winShmNode.mutex must be held or winShmNode.nRef==0 and ** winShmMutexHeld() is true when reading or writing any other field ** in this structure. **

75121. Size of buffer containing root node

75122. Contents of table pTab

75123. Term to scan doclist of

75124. Size of buffer at aNode (or 0)

75125. Number of dirty pages as a percentage of the configured cache size

75126. If the INSERT statement included an IDLIST term, then make sure ** all elements of the IDLIST really are columns of the table and ** remember the column indices. ** ** If the table has an INTEGER PRIMARY KEY column and that column ** is named in the IDLIST, then record in the ipkColumn variable ** the index into IDLIST of the primary key column. ipkColumn is ** the index of the primary key as it appears in IDLIST, not as ** is appears in the original table. (The index of the INTEGER ** PRIMARY KEY in the original table is pTab->iPKey.)

75127. ** The CHECK_PAGE macro takes a PgHdr* as an argument. If SQLITE_CHECK_PAGES ** is defined, and NDEBUG is not defined, an assert() statement checks ** that the page is either dirty or still matches the calculated page-hash.

75128. create_vtab ::= createkw VIRTUAL TABLE ifnotexists nm dbnm USING nm

75129. **comment:** ** If there are currently more than nMaxPage pages allocated, try ** to recycle pages to reduce the number allocated to nMaxPage.

label: code-design

75130. True if checksums in WAL are big-endian

75131. The condition (pIter->eLock!=eLock) in the following if(...) ** statement is a simplification of: ** ** (eLock==WRITE_LOCK || pIter->eLock==WRITE_LOCK) ** ** since we know that if eLock==WRITE_LOCK, then no other connection ** may hold a WRITE_LOCK on any table in this file (since there can ** only be a single writer).

75132. **comment:** ** Initialize the temporary index cursor just opened as a sorter cursor. ** ** Usually, the sorter module uses the value of (pCsr->pKeyInfo->nField) ** to determine the number of fields that should be compared from the ** records being sorted. However, if the value passed as argument nField ** is non-zero and the sorter is able to guarantee a stable sort, nField ** is used instead. This is used when sorting records for a CREATE INDEX ** statement. In this case, keys are always delivered to the sorter in ** order of the primary key, which happens to be make up the final part ** of the records being sorted. So if the sort is stable, there is never ** any reason to compare PK fields and they can be ignored for a small ** performance boost. ** ** The sorter can guarantee a stable sort when running in single-threaded ** mode, but not in multi-threaded mode. ** ** SQLITE_OK is returned if successful, or an SQLite error code otherwise.

label: code-design

75133. **comment:** Term to query for

label: code-design

75134. 30

75135. Return the new journal mode

75136. The escape character

75137. OUT: Set to true if dequoting required

75138. cmd ::= ATTACH database_kw_opt expr AS expr key_opt
75139. Used to be a conditional
75140. SQL functions can be expensive. So try to move constant functions ** out of the inner loop, even if that means an extra OP_Copy.
75141. ** Processing is determine by the affinity parameter: ** ** SQLITE_AFF_INTEGER: ** SQLITE_AFF_REAL: ** SQLITE_AFF_NUMERIC: ** Try to convert pRec to an integer representation or a ** floating-point representation if an integer representation ** is not possible. Note that the integer representation is ** always preferred, even if the affinity is REAL, because ** an integer representation is more space efficient on disk. ** ** SQLITE_AFF_TEXT: ** Convert pRec to a text representation. ** ** SQLITE_AFF_BLOB: ** No-op. pRec is unchanged.
75142. The page cache
75143. Sorter task to read from
75144. Column of table to populate aTerm for
75145. Return the total number of pages stored in the cache
75146. First element already in pEntry
75147. REPLACE_CONFIG
75148. Name of fts5 table
75149. Number of bytes of data space
75150. True if initialized
75151. Value for "leaves_end_block" field
75152. (276) ecmd ::= explain cmdx SEMI
75153. Write the next key to the output.
75154. bMemstat
75155. Value of %_segdir.leaves_end_block
75156. Requested snippet length (in tokens)
75157. Find the iHash'th table
75158. End for loop over the format string
75159. Use this buffer for space if required
75160. Number of memory cells required
75161. side-effects-ok
75162. If WHERE_OR_SUBCLAUSE is set, index cursor number ** If WHERE_USE_LIMIT, then the limit amount
75163. 0x04
75164. The record-size field is a 2 byte varint and the record ** fits entirely on the main b-tree page.
75165. Cursor from which column data is extracted
75166. varint containing prefix size
75167. ** Read the contents of frame iRead from the wal file into buffer pOut ** (which is nOut bytes in size). Return SQLITE_OK if successful, or an ** error code otherwise.
75168. defined(SQLITE_TEST) || defined(SQLITE OMIT RANDOMNESS)
75169. ** Compiling and using WAL mode requires several APIs that are only ** available in Windows platforms based on the NT kernel.
75170. Make sure the locking sequence is correct
75171. Send data to a co-routine
75172. Cursor that data will be read from
75173. ** Erase the eponymous virtual table instance associated with ** virtual table module pMod, if it exists.
75174. OUT: Created tokenizer cursor
75175. Version 3.8.11 and later
75176. **comment:** ** Write a 64-bit big-endian integer value to the buffer aBuf[].
 label: code-design
75177. ** This routine is called by the parser to add a new term to the ** end of a growing FROM clause. The "p" parameter is the part of ** the FROM clause that has already been constructed. "p" is NULL ** if this is the first term of the FROM clause. pTable and pDatabase ** are the name of the table and database named in the FROM clause term. ** pDatabase is NULL if the database name qualifier is missing - the ** usual case. If the term has an alias, then pAlias points to the ** alias token. If the term is a subquery, then pSubquery is the ** SELECT statement that the subquery encodes. The pTable and ** pDatabase parameters are NULL for subqueries. The pOn and pUsing ** parameters are the content of the ON and USING clauses. ** ** Return a new SrcList which encodes is the FROM with the new ** term added.
75178. Can only happen following on OOM
75179. ** A total of nLeaf leaf pages of data has just been flushed to a level-0 ** segment. This function updates the write-counter accordingly and, if ** necessary, performs incremental merge work. ** ** If an error occurs, set the Fts5Index.rc error code. If an error has ** already occurred, this function is a no-op.
75180. ** VDBE Calling Convention ** ----- ** Example: ** ** For the following INSERT statement: ** ** CREATE TABLE t1(a, b INTEGER PRIMARY KEY, c); ** INSERT INTO t1 VALUES(1, 2, 3.1); ** ** Register (x): 2 (type integer) ** Register (x+1): 1 (type integer) ** Register (x+2): NULL (type NULL) ** Register (x+3): 3.1 (type real)
75181. Data for current page of this level
75182. ** pE is a pointer to an expression which is a single term in the ** ORDER BY of a compound SELECT. The expression has not been ** name resolved. ** ** At the point this routine is called, we already know that the ** ORDER BY term is not an integer index into the result set. That ** case is handled by the calling routine. ** ** Attempt to match pE against result set columns in the left-most ** SELECT statement. Return the index i of the matching column, ** as an indication to the caller that it should sort by the i-th column. ** The left-most column is 1. In other words, the value returned is the ** same integer value that would be used in the SQL statement to indicate ** the column. ** ** If there is no match, return 0. Return -1 if an error occurs.
75183. ** This function is a complex assert() that verifies the following ** properties of the blocked connections list: ** ** 1) Each entry in the list has a non-NULL value for either ** pUnlockConnection or pBlockingConnection, or both. ** ** 2) All entries in the list that share a common value for ** xUnlockNotify are grouped together. ** ** 3) If the argument db is not NULL, then none of the entries in the ** blocked connections list have pUnlockConnection or pBlockingConnection ** set to db. This is used when closing connection db.
75184. Cells to delete
75185. End attempt to optimize using an index
75186. now insert a "." before the last / character
75187. "SELECT * FROM %_segdir WHERE level = ? ORDER BY ..."
75188. ** Set up a raw page so that it looks like a database page holding ** no entries.
75189. The index (1-based) of the term out of range
75190. ** CAPI3REF: Cancel Automatic Extension Loading ** ** ^The [sqlite3_cancel_auto_extension(X)] interface unregisters the ** initialization routine X that was registered using a prior call to ** [sqlite3_auto_extension(X)]. ^The [sqlite3_cancel_auto_extension(X)] ** routine returns 1 if initialization routine X was successfully ** unregistered and it returns 0 if X was not on the list of initialization ** routines.
75191. seltablist ::= stl_prefix nm dbnm LP exprlist RP as on_opt using_opt
75192. The subroutine that manifests the view might be a one-time routine, ** or it might need to be rerun on each iteration because it ** encodes a correlated subquery.
75193. ** Write a varint with value iVal into the buffer at aBuf. Return the ** number of bytes written.
75194. Need to invoke convertCompoundSelectToSubquery()
75195. ** Return true if the area covered by p2 is a subset of the area covered ** by p1. False otherwise.
75196. True if the loop has UNIQUE NOT NULL columns
75197. If this is an AUTOINCREMENT table, look up the sequence number in the ** sqlite_sequence table and store it in memory cell regAutoinc.
75198. Size of array at aElem
75199. ** An open write-ahead log file is represented by an instance of the ** following object.
75200. xCreate
75201. ** Figure out what version of the code to use. The choices are ** ** SQLITE_MUTEX OMIT No mutex logic. Not even stubs. The ** mutexes implementation cannot be overridden ** at start-time. ** ** SQLITE_MUTEX_NOOP For single-threaded applications. No ** mutual exclusion is provided. But this **

implementation can be overridden at ** start-time. *** ** SQLITE_MUTEX_PTHREADS For multi-threaded applications on Unix. *** ** SQLITE_MUTEX_W32 For multi-threaded applications on Win32.

75202. ** Return the sector size in bytes of the underlying block device for ** the specified file. This is almost always 512 bytes, but may be ** larger for some devices. *** ** SQLite code assumes this function cannot fail. It also assumes that ** if two files are created in the same file-system directory (i.e. ** a database and its journal file) that the sector size will be the ** same for both.

75203. **comment:** The following is just a sanity check to make sure SQLite has ** been compiled correctly. It is important to run this code, but ** we don't want to run it too often and soak up CPU cycles for no ** reason. So we run it once during initialization.

label: code-design

75204. ** Page number PAGER_MJ_PGNO is never used in an SQLite database (it is ** reserved for working around a windows/posix incompatibility). It is ** used in the journal to signify that the remainder of the journal file ** is devoted to storing a master journal name - there are no more pages to ** roll back. See comments for function writeMasterJournal() in pager.c ** for details.

75205. The expired flag was set on the VDBE before the first call ** to sqlite3_step(). For consistency (since sqlite3_step() was ** called), set the database error in this case as well.

75206. If a truth probability is specified using the likelihood() hints, ** then use the probability provided by the application.

75207. Release the locks.

75208. Check if the current entries really are a phrase match

75209. No key specified. Use the key from the main database

75210. 1130

75211. OUT: Selected leaf page

75212. **comment:** ** CAPI3REF: Add A Changeset To A Changegroup *** ** Add all changes within the changeset (or patchset) in buffer pData (size ** nData bytes) to the changegroup. *** ** If the buffer contains a patchset, then all prior calls to this function ** on the same changegroup object must also have specified patchsets. Or, if ** the buffer contains a changeset, so must have the earlier calls to this ** function. Otherwise, SQLITE_ERROR is returned and no changes are added ** to the changegroup. *** ** Rows within the changeset and changegroup are identified by the values in ** their PRIMARY KEY columns. A change in the changeset is considered to ** apply to the same row as a change already present in the changegroup if ** the two rows have the same primary key. *** ** Changes to rows that do not already appear in the changegroup are ** simply copied into it. Or, if both the new changeset and the changegroup ** contain changes that apply to a single row, the final contents of the ** changegroup depends on the type of each change, as follows: *** ** <table border=1 style="margin-left:8ex;margin-right:8ex"> ** <tr><th style="white-space:pre">Existing Change </th> ** <th style="white-space:pre">New Change </th> ** <th>Output Change *** <tr><td>INSERT <td>INSERT <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td>INSERT <td>UPDATE <td> ** The INSERT change remains in the changegroup. The values in the ** INSERT change are modified as if the row was inserted by the ** existing change and then updated according to the new change. ** <tr><td>DELETE <td> ** The existing INSERT is removed from the changegroup. The DELETE is ** not added. ** <tr><td>UPDATE <td>INSERT <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td>UPDATE <td> ** The existing UPDATE remains within the changegroup. It is amended ** so that the accompanying values are as if the row was updated once ** by the existing change and then again by the new change. ** <tr><td>DELETE <td> ** The existing UPDATE is replaced by the new DELETE within the ** changegroup. ** <tr><td>DELETE <td>INSERT <td> ** If one or more of the column values in the row inserted by the ** new change differ from those in the row deleted by the existing ** change, the existing DELETE is replaced by an UPDATE within the ** changegroup. Otherwise, if the inserted row is exactly the same ** as the deleted row, the existing DELETE is simply discarded. ** <tr><td>DELETE <td>UPDATE <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td>DELETE <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td> ** If the new changeset contains changes to a table that is already present ** in the changegroup, then the number of columns and the position of the ** primary key columns for the table must be consistent. If this is not the ** case, this function fails with SQLITE_SCHEMA. If the input changeset ** appears to be corrupt and the corruption is detected, SQLITE_CORRUPT is ** returned. Or, if an out-of-memory condition occurs during processing, this ** function returns SQLITE_NOMEM. In all cases, if an error occurs the ** final contents of the changegroup is undefined. *** ** If no error occurs, SQLITE_OK is returned.

label: code-design

75213. ** Array of lists of free blocks according to the block size ** for smaller chunks, or a hash on the block size for larger ** chunks.

75214. GetVersionEx() is current

75215. If there is no open transaction, then mark this as a special ** "transaction savepoint".

75216. expr ::= expr COLLATE ID|STRING

75217. Error message

75218. ** Convert an SQL-style quoted string into a normal string by removing ** the quote characters. The conversion is done in-place. If the ** input does not begin with a quote character, then this routine ** is a no-op. *** ** Examples: *** ** "abc" becomes abc ** 'xyz' becomes xyz ** [pqr] becomes pqr ** `mno` becomes mno

75219. List of active savepoints

75220. If the current node of layer iLayer contains zero keys, or if adding ** the key to it will not cause it to grow to larger than nNodeSize ** bytes in size, write the key here.

75221. ** Assign expression b to lvalue a. A second, no-op, version of this macro ** is provided when SQLITE OMIT_EXPLAIN is defined. This allows the code ** in sqlite3Select() to assign values to structure member variables that ** only exist if SQLITE OMIT_EXPLAIN is not defined without polluting the ** code with #ifndef directives.

75222. Jump here to start the next IN combination

75223. 129

75224. Advance the first pCsr->nAdvance entries in the apSegment[] array ** forward. Then sort the list in order of current term again.

75225. Size of segment just written

75226. First page of the overflow chain

75227. xDestroy - Drop a table

75228. ** Set the most recent error code and error string for the sqlite ** handle "db". The error code is set to "err_code". *** ** If it is not NULL, string zFormat specifies the format of the ** error string in the style of the printf functions: The following ** format characters are allowed: *** ** %s Insert a string ** %z A string that should be freed after use ** %d Insert an integer ** %T Insert a token ** %S Insert the first element of a SrcList ** ** zFormat and any string tokens that follow it are assumed to be ** encoded in UTF-8. *** ** To clear the most recent error for sqlite handle "db", sqlite3Error ** should be called with err_code set to SQLITE_OK and zFormat set ** to NULL.

75229. 1200

75230. ** Structures used by the virtual table interface

75231. ** Initialize all term iterators in the pNear object. If any term is found ** to match no documents at all, return immediately without initializing any ** further iterators. *** ** If an error occurs, return an SQLite error code. Otherwise, return ** SQLITE_OK. It is not considered an error if some term matches zero ** documents.

75232. Size of record in bytes

75233. Only the first 8 bytes (above) are zeroed by pager.c when a new page ** is allocated. All fields that follow must be initialized before use

75234. SQLITE_ECEL_* flags

75235. 5: (start_constraints && !startEq && bRev)

75236. The xColumn method simply returns the corresponding column from ** the PRAGMA.

75237. If pCur->pKeyInfo is not NULL, then the caller that opened this cursor ** expected to open it on an index b-tree. Otherwise, if pKeyInfo is ** NULL, the caller expects a table b-tree. If this is not the case, ** return an SQLITE_CORRUPT error. *** ** Earlier versions of SQLite assumed that this test could not fail ** if the root page was already loaded when this function was called (i.e. ** if pCur->iPage>=0). But this is not so if the database is corrupted ** in such a way that page pRoot is linked into a second b-tree table ** (or the freelist).

75238. Output flags returned to SQLite core

75239. ** This function is called before generating code to update or delete a ** row contained in table pTab. If the operation is a DELETE, then ** parameter aChange is passed a NULL value. For an UPDATE, aChange points ** to an array of size N, where N is the number of columns in table pTab. ** If the i'th column is not modified by the UPDATE, then the corresponding ** entry in the aChange[] array is set to -1. If the column is modified, ** the value is 0 or greater. Parameter

chngRowid is set to true if the ** UPDATE statement modifies the rowid fields of the table. *** If any foreign key processing will be required, this function returns ** non-zero. If there is no foreign key related processing, this function ** returns zero. *** For an UPDATE, this function returns 2 if: *** *** There are any FKs for which pTab is the child and the parent table, or *** the UPDATE modifies one or more parent keys for which the action is ** not "NO ACTION" (i.e. is CASCADE, SET DEFAULT or SET NULL). *** Or, assuming some other foreign key processing is required, 1.

75240. **comment:** No memory allocation is ever used on mem1. Prove this using ** the following assert(). If the assert() fails, it indicates a ** memory leak and a need to call sqlite3VdbeMemRelease(&mem1).

label: code-design

75241. Size of regions

75242. Address of array containing child table row

75243. A result set

75244. ** 2007 June 22 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file implements a tokenizer for fts3 based on the ICU library.

75245. The szMalloc field holds the correct memory allocation size

75246. Name of the file (UTF-8)

75247. p->flags holds EP_Collate and p->pLeft->flags does not. And ** p->x.pSelect cannot. So if p->x.pLeft exists, it must hold at ** least one EP_Collate. Thus the following two ALWAYS.

75248. **comment:** ** In addition to its current configuration, have the Fts3MultiSegReader ** passed as the 4th argument also scan the doclist for term zTerm/nTerm. ** ** SQLITE_OK is returned if no error occurs, otherwise an SQLite error code.

label: code-design

75249. ** This function starts a write transaction on the WAL. *** A read transaction must have already been started by a prior call ** to sqlite3WalBeginReadTransaction(). *** If another thread or process has written into the database since ** the read transaction was started, then it is not possible for this ** thread to write as doing so would cause a fork. So this routine ** returns SQLITE_BUSY in that case and no write transaction is started. *** There can only be a single writer active at a time.

75250. Append to this SrcList. NULL creates a new SrcList

75251. Which tables are currently available

75252. ** CAPI3REF: Online Backup Object *** The sqlite3_backup object records state information about an ongoing ** online backup operation. ^The sqlite3_backup object is created by ** a call to [sqlite3_backup_init()] and is destroyed by a call to ** [sqlite3_backup_finish()]. *** See Also: [Using the SQLite Online Backup API]

75253. ** Code an OP_TableLock instruction for each table locked by the ** statement (configured by calls to sqlite3TableLock()).

75254. The associated database

75255. ** Search the free-list on page pPg for space to store a cell nByte bytes in ** size. If one can be found, return a pointer to the space and remove it ** from the free-list. *** If no suitable space can be found on the free-list, return NULL. ** ** This function may detect corruption within pPg. If corruption is ** detected then *pRc is set to SQLITE_CORRUPT and NULL is returned. ** ** Slots on the free list that are between 1 and 3 bytes larger than nByte ** will be ignored if adding the extra space to the fragmentation count ** causes the fragmentation count to exceed 60.

75256. ** Advance iterator pIter to the next entry. ** ** This version of fts5SegIterNext() is only used by reverse iterators.

75257. Make the JSON in p the result of the SQL function.

75258. ** Move the iterator passed as the only argument to the previous entry.

75259. Update the sqlite_sequence table by storing the content of the ** maximum rowid counter values recorded while inserting into ** autoincrement tables.

75260. porter rule condition: (m > 1 and (*S or *T))

75261. Append the table key to the end of the index. For WITHOUT ROWID ** tables (when pPk!=0) this will be the declared PRIMARY KEY. For ** normal tables (when pPk==0) this will be the rowid.

75262. ** In the amalgamation, the parse.c file generated by lemon and the ** tokenize.c file are concatenated. In that case, sqlite3RunParser() ** has access to the the size of the yyParser object and so the parser ** engine can be allocated from stack. In that case, only the ** sqlite3ParserInit() and sqlite3ParserFinalize() routines are invoked ** and the sqlite3ParserAlloc() and sqlite3ParserFree() routines can be ** omitted.

75263. pExpr is original. Make a new entry in pAggInfo->aFunc[]

75264. **comment:** A malloc must have failed

label: code-design

75265. **comment:** This routine is never used in single-threaded mode

label: requirement

75266. OP_Found will use an unpacked key

75267. Bytes buffered before opening the file

75268. ** Return a pointer to a subexpression of pVector that is the i-th ** column of the vector (numbered starting with 0). The caller must ** ensure that i is within range. *** If pVector is really a scalar (and "scalar" here includes subqueries ** that return a single column!) then return pVector unmodified. ** ** pVector retains ownership of the returned subexpression. ** ** If the vector is a (SELECT ...) then the expression returned is ** just the expression for the i-th term of the result set, and may ** not be ready for evaluation because the table cursor has not yet ** been positioned.

75269. Payload to be inserted

75270. Return segments in order from oldest to newest.

75271. ***** Continuing where we left off in pragma.c *****

75272. ***** Begin file dbstat.c *****

75273. The FTS3 table

75274. Page number array for hash table

75275. OUT: Final checksum value output

75276. ***** End of legacy.c *****

75277. ** Add a new module argument to pTable->azModuleArg[]. ** The string is not copied - the pointer is stored. The ** string will be freed automatically when the table is ** deleted.

75278. First register in OLD.* array

75279. Create a record for this index entry as it should appear after ** the insert or update. Store that record in the aRegIdx[ix] register

75280. ***** End %syntax_error code *****

75281. OUT: Number of backfilled frames in WAL

75282. Left side of comparison

75283. Assuming the wal-index file was successfully mapped, populate the ** page number array and hash table entry.

75284. Tokenizer for this table

75285. Total bytes of data (incl. structure)

75286. First in linked list of saved aux-data

75287. Check that the pointer pOld points to a valid, non-free block.

75288. Next sample to test

75289. Register that is true if RHS contains NULL values

75290. If the first attempt to open the database file fails and the bRetry ** flag is set, this means that the db was not opened because it seemed ** to be a wal-mode db. But, this may have happened due to an earlier ** RBU vacuum operation leaving an old wal file in the directory. ** If this is the case, it will have been checkpointed and deleted ** when the handle was closed and a second attempt to open the ** database may succeed.

75291. 1. first try to open/create the file ** 2. if that fails, and this is a lock file (not-conch), try creating ** the parent directories and then try again. ** 3. if that fails, try to open the file read-only ** otherwise return BUSY (if lock file) or CANTOPEN for the conch file

75292. The next block of code is equivalent to: ** ** pIter += getVarint(pIter, (u64*)&pInfo->nKey); ** ** The code is inlined to avoid a function call.

75293. ** The following 256 byte lookup table is used to support SQLites built-in ** equivalents to the following standard library functions: ** ** isspace() 0x01 ** isalpha() 0x02 ** isdigit() 0x04 ** isalnum() 0x06 ** isxdigit() 0x08 ** toupper() 0x20 ** SQLite identifier character 0x40 ** Quote character 0x80 ** ** Bit 0x20 is set if the mapped character requires translation to upper ** case. i.e. if the character is a lower-case ASCII character. ** If x is a lower-case ASCII character, then its upper-case equivalent ** is (x - 0x20). Therefore toupper() can be implemented as: ** ** (x & ~map[x]&0x20) ** ** The equivalent of

tolower() is implemented using the sqlite3UpperToLower[] ** array. tolower() is used more often than toupper() by SQLite. *** Bit 0x40 is set if the character is non-alphanumeric and can be used in an ** SQLite identifier. Identifiers are alphanumeric, "_", "\$", and any ** non-ASCII UTF character. Hence the test for whether or not a character is ** part of an identifier is 0x46.

75294. VM to add scanstatus() to

75295. ** This function is used to create the text of expressions of the form: ** name=<constant1> OR name=<constant2> OR ... ** ** If argument zWhere is NULL, then a pointer string containing the text ** "name=<constant>" is returned, where <constant> is the quoted version ** of the string passed as argument zConstant. The returned buffer is ** allocated using sqlite3DbMalloc(). It is the responsibility of the ** caller to ensure that it is eventually freed. *** If argument zWhere is not NULL, then the string returned is ** "<where> OR name=<constant>", where <where> is the contents of zWhere. ** In this case zWhere is passed to sqlite3DbFree() before returning. **

75296. Mask of FTS5_TOKENIZE_* flags

75297. ** Append SegReader object pNew to the end of the pCsr->apSegment[] array.

75298. Page number

75299. True if WAL file exists

75300. 2,3

75301. Assert that operands are never NULL

75302. columnname ::= nm typetoken

75303. The SrcList to be enlarged

75304. **comment:** ** Return TRUE if the WHERE clause term pTerm is of a form where it ** could be used with an index to access pSrc, assuming an appropriate ** index existed.

label: code-design

75305. Update the in-memory representation of all UNIQUE indices by converting ** the final rowid column into one or more columns of the PRIMARY KEY.

75306. EOF when i equals or exceeds this value

75307. ** Advance iterator pIter to the next entry. *** If an error occurs, Fts5Index.rc is set to an appropriate error code. It ** is not considered an error if the iterator reaches EOF. If an error has ** already occurred when this function is called, it is a no-op.

75308. Deallocate page if true

75309. ** For some Windows sub-platforms, the _beginthreadex() / _endthreadex() ** functions are not available (e.g. those not using MSVC, Cygwin, etc).

75310. Opcode: Found P1 P2 P3 P4 ** Synopsis: key=r[P3@P4] *** If P4==0 then register P3 holds a blob constructed by MakeRecord. If ** P4>0 then register P3 is the first of P4 registers that form an unpacked ** record. *** Cursor P1 is on an index btree. If the record identified by P3 and P4 ** is a prefix of any entry in P1 then a jump is made to P2 and ** P1 is left pointing at the matching entry. *** This operation leaves the cursor in a state where it can be ** advanced in the forward direction. The Next instruction will work, ** but not the Prev instruction. *** See also: NotFound, NoConflict, NotExists. SeekGe

75311. A connection with the read-uncommitted flag set will never try to ** obtain a read-lock using this function. The only read-lock obtained ** by a connection in read-uncommitted mode is on the sqlite_master ** table, and that lock is obtained in BtreeBeginTrans().

75312. If pWalker->eCode is 2 then any term of the expression that comes from ** the ON or USING clauses of a left join disqualifies the expression ** from being considered constant.

75313. ** End of highlight() implementation. *****

75314. Zero-length table names are allowed

75315. IMP: R-51971-34154

75316. Total leaves written to level 0

75317. Create new databases in format 1

75318. ** Page type flags. An ORed combination of these flags appear as the ** first byte of on-disk image of every BTree page.

75319. The WHERE processing context

75320. OUT: Allocated Wal handle

75321. RHS of LIKE/GLOB ends with wildcard

75322. Now update the actual sibling pages. The order in which they are updated ** is important, as this code needs to avoid disrupting any page from which ** cells may still to be read. In practice, this means: ** ** (1) If cells are moving left (from apNew[iPg] to apNew[iPg-1]) ** then it is not safe to update page apNew[iPg] until after ** the left-hand sibling apNew[iPg-1] has been updated. ** ** (2) If cells are moving right (from apNew[iPg] to apNew[iPg+1]) ** then it is not safe to update page apNew[iPg] until after ** the right-hand sibling apNew[iPg+1] has been updated. ** ** If neither of the above apply, the page is safe to update. *** The iPg value in the following loop starts at nNew-1 goes down ** to 0, then back up to nNew-1 again, thus making two passes over ** the pages. On the initial downward pass, only condition (1) above ** needs to be tested because (2) will always be true from the previous ** step. On the upward pass, both conditions are always true, so the ** upwards pass simply processes pages that were missed on the downward ** pass.

75323. If it did not delete it, the row-trigger may still have modified ** some of the columns of the row being updated. Load the values for ** all columns not modified by the update statement into their ** registers in case this has happened.

75324. frame == (aHash[x]+iZero)

75325. Number of tokens in phrase

75326. OUT: Parsed query structure

75327. Indicates an exclusive lock has been obtained

75328. ** If the second argument is not NULL, release any allocations associated ** with the memory cells in the p->aMem[] array. Also free the UnpackedRecord ** structure itself, using sqlite3DbFree(). *** This function is used to free UnpackedRecord structures allocated by ** the vdbeUnpackRecord() function found in vdbeapi.c.

75329. The WherePath to check

75330. Gather entropy into this buffer

75331. Record the number of outstanding lookaside allocations in schema Tables ** prior to doing any free() operations. Since schema Tables do not use ** lookaside, this number should not change.

75332. Table being modified

75333. Column to scan for

75334. all memory is allocated, proxys are created and assigned, ** switch the locking context and pMethod then return.

75335. Maximum number of cells for pNode

75336. If currently processing the PRIMARY KEY of a WITHOUT ROWID ** table, only conflict if the new PRIMARY KEY values are actually ** different from the old. *** For a UNIQUE index, only conflict if the PRIMARY KEY values ** of the matched index row are different from the original PRIMARY ** KEY values of this row before the update.

75337. **comment:** ** CAPI3REF: OS Interface Object ** ** An instance of the sqlite3_vfs object defines the interface between ** the SQLite core and the underlying operating system. The "vfs" ** in the name of the object stands for "virtual file system". See ** the [VFS | VFS documentation] for further information. *** ** The value of the iVersion field is initially 1 but may be larger in ** future versions of SQLite. Additional fields may be appended to this ** object when the iVersion value is increased. Note that the structure ** of the sqlite3_vfs object changes in the transaction between ** SQLite version 3.5.9 and 3.6.0 and yet the iVersion field was not ** modified. *** The szOsFile field is the size of the subclassed [sqlite3_file] ** structure used by this VFS. mxPathname is the maximum length of ** a pathname in this VFS. *** Registered sqlite3_vfs objects are kept on a linked list formed by ** the pNext pointer. The [sqlite3_vfs_register()] ** and [sqlite3_vfs_unregister()] interfaces manage this list ** in a thread-safe way. The [sqlite3_vfs_find()] interface ** searches the list. Neither the application code nor the VFS ** implementation should use the pNext pointer. *** The pNext field is the only field in the sqlite3_vfs ** structure that SQLite will ever modify. SQLite will only access ** or modify this field while holding a particular static mutex. ** The application should never modify anything within the sqlite3_vfs ** object once the object has been registered. *** The zName field holds the name of the VFS module. The name must ** be unique across all VFS modules. *** [[sqlite3_vfs.xOpen]] ** ^SQLite guarantees that the zFilename parameter to xOpen ** is either a NULL pointer or string obtained ** from xFullPathname() with an optional suffix added. ** ^If a suffix is added to the zFilename parameter, it will ** consist of a single "-" character followed by no more than ** 11 alphanumeric and/or "-" characters. ** ^SQLite further guarantees that ** the string will be valid and unchanged until xClose() is ** called. Because of the previous sentence, ** the [sqlite3_file] can safely store a pointer to the ** filename if it needs to remember the filename for some reason. ** If the zFilename parameter to xOpen is a NULL pointer then xOpen ** must invent its own temporary name for the file. ^Whenever the ** xFilename parameter is NULL it will also be the case that the ** flags parameter will include [SQLITE_OPEN_DELETEONCLOSE]. *** The flags argument to xOpen() includes all bits set in ** the flags argument to [sqlite3_open_v2()]. Or if [sqlite3_open()] ** or [sqlite3_open16()] is used, then flags includes at least ** [SQLITE_OPEN_READWRITE] | [SQLITE_OPEN_CREATE]. ** If xOpen() opens a file read-only then it sets *pOutFlags to ** include [SQLITE_OPEN_READONLY]. Other bits in *pOutFlags

may be set. *** ^^(SQLite will also add one of the following flags to the xOpen() ** call, depending on the object being opened: *** ** [SQLITE_OPEN_MAIN_DB] ** [SQLITE_OPEN_MAIN_JOURNAL] ** [SQLITE_OPEN_TEMP_DB] ** [SQLITE_OPEN_TEMP_JOURNAL] ** [SQLITE_OPEN_TRANSIENT_DB] ** [SQLITE_OPEN_SUBJOURNAL] ** [SQLITE_OPEN_MASTER_JOURNAL] ** [SQLITE_OPEN_WAL] **)*** The file I/O implementation can use the object type flags to ** change the way it deals with files. For example, an application ** that does not care about crash recovery or rollback might make ** the open of a journal file a no-op. Writes to this journal would ** also be no-ops, and any attempt to read the journal would return ** SQLITE_IOERR. Or the implementation might recognize that a database ** file will be doing page-aligned sector reads and writes in a random ** order and set up its I/O subsystem accordingly. *** SQLite might also add one of the following flags to the xOpen method: *** ** [SQLITE_OPEN_DELETEONCLOSE] ** [SQLITE_OPEN_EXCLUSIVE] ** *** The [SQLITE_OPEN_DELETEONCLOSE] flag means the file should be ** deleted when it is closed. ^The [SQLITE_OPEN_DELETEONCLOSE] ** will be set for TEMP databases and their journals, transient ** databases, and subjournals. *** ^The [SQLITE_OPEN_EXCLUSIVE] flag is always used in conjunction ** with the [SQLITE_OPEN_CREATE] flag, which are both directly ** analogous to the O_EXCL and O_CREAT flags of the POSIX open() ** API. The SQLITE_OPEN_EXCLUSIVE flag, when paired with the ** SQLITE_OPEN_CREATE, is used to indicate that file should always ** be created, and that it is an error if it already exists. *** It is <i>not</i> used to indicate the file should be opened ** for exclusive access. *** ^At least szOsFile bytes of memory are allocated by SQLite ** to hold the [sqlite3_file] structure passed as the third ** argument to xOpen. The xOpen method does not have to ** allocate the structure; it should just fill it in. Note that ** the xOpen method must set the sqlite3_file.pMethods to either ** a valid [sqlite3_io_methods] object or to NULL. xOpen must do ** this even if the open fails. SQLite expects that the sqlite3_file.pMethods ** element will be valid after xOpen returns regardless of the success ** or failure of the xOpen call. *** [[sqlite3_vfs.xAccess]] ** ^The flags argument to xAccess() may be [SQLITE_ACCESS_EXISTS] ** to test for the existence of a file, or [SQLITE_ACCESS_READWRITE] to ** test whether a file is readable and writable, or [SQLITE_ACCESS_READ] ** to test whether a file is at least readable. The file can be a ** directory. *** ^SQLite will always allocate at least mxPathname+1 bytes for the ** output buffer xFullPathname. The exact size of the output buffer ** is also passed as a parameter to both methods. If the output buffer ** is not large enough, [SQLITE_CANTOPEN] should be returned. Since this is ** handled as a fatal error by SQLite, vfs implementations should endeavor ** to prevent this by setting mxPathname to a sufficiently large value. *** ^The xRandomness(), xSleep(), xCurrentTime(), and xCurrentTimeInt64() ** interfaces are not strictly a part of the filesystem, but they are ** included in the VFS structure for completeness. ** The xRandomness() function attempts to return nBytes bytes ** of good-quality randomness into zOut. The return value is ** the actual number of bytes of randomness obtained. ** The xSleep() method causes the calling thread to sleep for at ** least the number of microseconds given. ^The xCurrentTime() ** method returns a Julian Day Number for the current date and time as ** a floating point value. *** ^The xCurrentTimeInt64() method returns, as an integer, the Julian ** Day Number multiplied by 86400000 (the number of milliseconds in ** a 24-hour day). *** ^SQLite will use the xCurrentTimeInt64() method to get the current ** date and time if that method is available (if iVersion is 2 or ** greater and the function pointer is not NULL) and will fall back ** to xCurrentTime() if xCurrentTimeInt64() is unavailable. *** ^The xSetSystemCall(), xGetSystemCall(), and xNestSystemCall() interfaces ** are not used by the SQLite core. These optional interfaces are provided ** by some VFSes to facilitate testing of the VFS code. By overriding ** system calls with functions under its control, a test program can ** simulate faults and error conditions that would otherwise be difficult ** or impossible to induce. The set of system calls that can be overridden ** varies from one VFS to another, and from one version of the same VFS to the ** next. Applications that use these interfaces must be prepared for any ** or all of these interfaces to be NULL or for their behavior to change ** from one release to the next. Applications must not attempt to access ** any of these methods if the iVersion of the VFS is less than 3.

label: code-design

75338. Select the child node which will be enlarged the least if pCell ** is inserted into it. Resolve ties by choosing the entry with ** the smallest area.

75339. Aux data pointer

75340. ** Return TRUE if the database file is opened read-only. Return FALSE ** if the database is (in theory) writable.

75341. Remove the slot from the free-list. Update the number of ** fragmented bytes within the page.

75342. Local cache of pPage->hdrOffset

75343. Cursor to iterate through level(s)

75344. The main DB, main journal, WAL file and master journal are never ** automatically deleted. Nor are they ever temporary files.

75345. List of available buffers

75346. Number of szAtom sized blocks in zPool

75347. Or, if aOut==0, here

75348. **comment:** Used only to suppress a compiler warning

label: code-design

75349. Index of parent of each node

75350. sqlite3Fts5Parser_ENGINEALWAYSSTACK

75351. A VACUUM cannot change the pagesize of an encrypted database.

75352. ** Configure SQL variable iVar so that binding a new value to it signals ** to sqlite3_reoptimize() that re-preparing the statement may result ** in a better query plan.

75353. The root page of the b-tree now contains no cells. The only sibling ** page is the right-child of the parent. Copy the contents of the ** child page into the parent, decreasing the overall height of the ** b-tree structure by one. This is described as the "balance-shallower" ** sub-algorithm in some documentation. *** If this is an auto-vacuum database, the call to copyNodeContent() ** sets all pointer-map entries corresponding to database image pages ** for which the pointer is stored within the content being copied. *** It is critical that the child page be defragmented before being ** copied into the parent, because if the parent is page 1 then it will ** be smaller than the child due to the database header, and so all the ** free space needs to be up front.

75354. Functions to support testing and debugging.

75355. ** CAPI3REF: SQL Function Context Object *** The context in which an SQL function executes is stored in an ** sqlite3_context object. ^A pointer to an sqlite3_context object ** is always first parameter to [application-defined SQL functions]. ** The application-defined SQL function implementation will pass this ** pointer through into calls to [sqlite3_result_int | sqlite3_result()], ** [sqlite3_aggregate_context()], [sqlite3_user_data()], ** [sqlite3_context_db_handle()], [sqlite3_get_auxdata()], ** and/or [sqlite3_set_auxdata()].

75356. IN terms are only valid for sorting in the ORDER BY LIMIT ** optimization, and then only if they are actually used ** by the query plan

75357. tab1 and tab2 may not be the same table

75358. The name of the table/view the trigger applies to

75359. Compute the complete text of the CREATE statement

75360. Return code from subprocedures

75361. Load the statistics from the sqlite_stat4 table.

75362. Restrictions (22) and (24)

75363. 36

75364. ** chng_addr_N: ** regRowid = idx(rowid) // STAT34 only ** stat_push(P, regChng, regRowid) // 3rd parameter STAT34 only ** Next csr ** if !eof(csr) goto next_row;

75365. Copy of Fts5Config.eDetail

75366. 2:save off p0<<21 | p1<<14 | p2<<7 | p3 (masked)

75367. ** Close an iterator opened by an earlier call to sqlite3Fts5IndexQuery().

75368. **comment:** ** Some Microsoft compilers lack this definition.

label: test

75369. Size of buffer a[] in bytes

75370. **comment:** ** Set the value of the Pager.sectorSize variable for the given ** pager based on the value returned by the xSectorSize method ** of the open database file. The sector size will be used ** to determine the size and alignment of journal header and ** master journal pointers within created journal files. *** For temporary files the effective sector size is always 512 bytes. *** Otherwise, for non-temporary files, the effective sector size is ** the value returned by the xSectorSize() method rounded up to 32 if ** it is less than 32, or rounded down to MAX_SECTOR_SIZE if it ** is greater than MAX_SECTOR_SIZE. *** If the file has the SQLITE_IOCAP_POWERSAFE_OVERWRITE property, then set ** the effective sector size to its minimum value (512). The purpose of ** pPager->sectorSize is to define the "blast radius" of bytes that ** might change if a crash occurs while writing to a single byte in ** that range. But with POWERSAFE_OVERWRITE, the blast radius is zero ** (that is what POWERSAFE_OVERWRITE means), so we minimize the sector ** size. For backwards compatibility of the rollback journal file format, ** we cannot reduce the effective sector size below 512.

label: code-design

75371. Fields in sorter record

75372. Verify that the version number on the WAL format is one that ** are able to understand

75373. ** Enable or disable the session object passed as the first argument.
75374. Separate the left and the right query from one another
75375. Index Name NULL
75376. ** 2001 September 22 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This is the header file for the generic hash-table implementation ** used in SQLite. We've modified it slightly to serve as a standalone ** hash table implementation for the full-text indexing module. **

75377. ** The following functions: ** ** sqlite3VdbeSerialType() ** sqlite3VdbeSerialTypeLen() ** sqlite3VdbeSerialLen() ** sqlite3VdbeSerialPut() **
sqlite3VdbeSerialGet() ** ** encapsulate the code that serializes values for storage in SQLite ** data and index records. Each serialized value consists of a ** 'serial-type' and a blob of data. The serial type is an 8-byte unsigned ** integer, stored as a varint. ** ** In an SQLite index record, the serial type is stored directly before ** the blob of data that it corresponds to. In a table record, all serial ** types are stored at the start of the record, and the blobs of data at ** the end. Hence these functions allow the caller to handle the ** serial-type and data blob separately. ** ** The following table describes the various storage classes for data: ** ** serial type bytes of data type ** ----- ** 0 0 NULL ** 1 1 signed integer ** 2 2 signed integer ** 3 3 signed integer ** 4 4 signed integer ** 5 6 signed integer ** 6 8 signed integer ** 7 8 IEEE float ** 8 0 Integer constant 0 ** 9 0 Integer constant 1 ** 10,11 reserved for expansion ** N>=12 and even (N-12)/2 BLOB ** N>=13 and odd (N-13)/2 text ** ** The 8 and 9 types were added in 3.3.0, file format 4. Prior versions ** of SQLite will not understand those serial types.

75378. True if next rowid is first in doclist
75379. ** This function is purely an internal test. It does not contribute to ** FTS functionality, or even the integrity-check, in any way. ** ** Instead, it tests that the same set of pgno/rowid combinations are ** visited regardless of whether the doclist-index identified by parameters ** iSegid/iLeaf is iterated in forwards or reverse order.

75380. P5
75381. ** Subqueries stores the original database, table and column names for their ** result sets in ExprList.a[].zSpan, in the form "DATABASE.TABLE.COLUMN". ** Check to see if the zSpan given to this routine matches the zDb, zTab, ** and zCol. If any of zDb, zTab, and zCol are NULL then those fields will ** match anything.

75382. szMmap
75383. Add a segment iterator for the current contents of the hash table.
75384. Cursor used to read input data
75385. 123456789 123456789 123
75386. colsetlist ::= STRING
75387. True to scan from zTerm to EOF
75388. ***** sqlite3_pcache Methods *****
75389. same as TK_SLASH, in1, in2, out3
75390. OUT: Error number if error occurs
75391. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains C code routines that are called by the parser ** to handle INSERT statements in SQLite.

75392. Rowid for current entry
75393. Check if the temp table already contains this key. If so, ** the row has already been included in the result set and ** can be ignored (by jumping past the Gosub below). Otherwise, ** insert the key into the temp table and proceed with processing ** the row. ** ** Use some of the same optimizations as OP_RowSetTest: If iSet ** is zero, assume that the key cannot already be present in ** the temp table. And if iSet is -1, assume that there is no ** need to insert the key into the temp table, as it will never ** be tested for.

75394. New virtual expression
75395. Opcode: ReopenIdx P1 P2 P3 P4 P5 ** Synopsis: root=P2 iDb=P3 ** ** The ReopenIdx opcode works exactly like ReadOpen except that it first ** checks to see if the cursor on P1 is already open with a root page ** number of P2 and if it is this opcode becomes a no-op. In other words, ** if the cursor is already open, do not reopen it. ** ** The ReopenIdx opcode may only be used with P5==0 and with P4 being ** a P4_KEYINFO object. Furthermore, the P3 value must be the same as ** every other ReopenIdx or OpenRead for the same cursor number. ** ** See the OpenRead opcode documentation for additional information.

75396. ELSE
75397. The sqlite_statN table does not exist. Create it. Note that a ** side-effect of the CREATE TABLE statement is to leave the rootpage ** of the new table in register pParse->regRoot. This is important ** because the OpenWrite opcode below will be needing it.

75398. OUT: New iterator
75399. Size of shared-memory regions
75400. True to set the USESEEKRESULT flag on OP_[Idx]Insert
75401. Commit by zeroing journal header
75402. At this point, any non-zero iOrderByCol values indicate that the ** ORDER BY column expression is identical to the iOrderByCol'th ** expression returned by SELECT statement pSub. Since these values ** do not necessarily correspond to columns in SELECT statement pParent, ** zero them before transferring the ORDER BY clause. ** ** Not doing this may cause an error if a subsequent call to this ** function attempts to flatten a compound sub-query into pParent ** (the only way this can happen is if the compound sub-query is ** currently part of pSub->pSrc). See ticket [d11a6e908f].

75403. Load b.apCell[] with pointers to all cells in pOld. If pOld ** contains overflow cells, include them in the b.apCell[] array ** in the correct spot. ** ** Note that when there are multiple overflow cells, it is always the ** case that they are sequential and adjacent. This invariant arises ** because multiple overflows can only occur when inserting divider ** cells into a parent on a prior balance, and divider cells are always ** adjacent and are inserted in order. There is an assert() tagged ** with "NOTE 1" in the overflow cell insertion loop to prove this ** invariant. ** ** This must be done in advance. Once the balance starts, the cell ** offset section of the btree page will be overwritten and we will no ** long be able to find the cells if a pointer to each cell is not saved ** first.

75404. ** Add an entry to the Fts5SFinder.aFirst[] array. Grow the array if ** necessary. Return SQLITE_OK if successful, or SQLITE_NOMEM if an ** error occurs.
75405. 08..0f

75406. Statement number (or 0 if has not opened stmt)
75407. Load the contents of the %_config table
75408. If one value is NULL, it is less than the other. If both values ** are NULL, return 0.
75409. moved CSE2 up
75410. Function to call when busy
75411. First character of input text
75412. Move to the next entry that matches the configured constraints.
75413. Code an OP_Expire. For an ATTACH statement, set P1 to true (expire this ** statement only). For DETACH, set it to false (expire all existing ** statements).
75414. buffer in which to construct conch name
75415. Index of column in second table
75416. A fake index object for the primary key
75417. Column to match on.
75418. Offset to next unparsed byte of the header
75419. ** The MSVC CRT on Windows CE may not have a localtime() function. ** So define a substitute.
75420. SQLITE OMIT_SCHEMA_VERSION_PRAGMAS
75421. Return no rowids earlier than this
75422. pkChng!=0 does not mean that the rowid has changed, only that ** it might have changed. Skip the conflict logic below if the rowid ** is unchanged.
75423. Write the header for the new UPDATE change. Same as the original.
75424. SQLITE_PREPARE_* flags
75425. **comment:** If the default value for the new column was specified with a ** literal NULL, then set pDflt to 0. This simplifies checking ** for an SQL NULL default below.
label: code-design
75426. ** This interface is used by the fts5vocab module.

75427. For UNIQUE indexes, verify that only one entry exists with the ** current key. The entry is unique if (1) any column is NULL ** or (2) the next entry has a different key

75428. First allowable cell index

75429. The SnippetIter object has just been initialized. The first snippet ** candidate always starts at offset 0 (even if this candidate has a ** score of 0.0).

75430. Declare the table schema to SQLite.

75431. Index name

75432. FTS cursor handle

75433. ** Truncate an open file to a specified size

75434. Make sure cursor numbers have been assigned to all entries in ** the FROM clause of the SELECT statement.

75435. If there was a master journal and this routine will return success, ** see if it is possible to delete the master journal.

75436. constructor for a JsonEachCursor object for json_each().

75437. A complex statement machine used to detect the end of a CREATE TRIGGER ** statement. This is the normal case.

75438. If a transaction is open, roll it back. This also ensures that if ** any database schemas have been modified by an uncommitted transaction ** they are reset. And that the required b-tree mutex is held to make ** the pager rollback and schema reset an atomic operation.

75439. (301) resolvetype ::= raisetype (OPTIMIZED OUT)

75440. True to use in-memory sub-journals

75441. Parent key index for this FK

75442. Use the foreground thread for this operation

75443. True if this is a VIRTUAL table

75444. VDBE cursor for the canonical data source

75445. Disables and re-enables match

75446. ** Remove entries from the sqlite_statN tables (for N in (1,2,3)) ** after a DROP INDEX or DROP TABLE command.

75447. This block codes the top of loop only. The complete loop is the ** following pseudocode (template 3): *** C: yield X, at EOF goto D ** insert the select result into <table> from R..R+n ** goto C ** D: ...

75448. Index key to query for

75449. ** Implementation of the xSetAuxdata() method.

75450. True to omit this if found

75451. ***** Begin file mem3.c *****

75452. ** Print the SQL that was used to generate a VDBE program.

75453. Constructor for the json_each virtual table

75454. ** Mark the VDBE as one that can only be run multiple times.

75455. ** Append position iPos to the position list being accumulated in buffer ** pBuf, which must be already be large enough to hold the new data. ** The previous position written to this list is *piPrev. *piPrev is set ** to iPos before returning.

75456. ** This function performs the parts of the "close file" operation ** common to all locking schemes. It closes the directory and file ** handles, if they are valid, and sets all fields of the unixFile ** structure to 0. *** It is *not* necessary to hold the mutex when this routine is called, ** even on VxWorks. A mutex will be acquired on VxWorks by the ** vxworksReleaseFileId() routine.

75457. Add the new entry to the end of the cache

75458. Name of the database to use internally

75459. ** This function is called when flushing a leaf page that contains no ** terms at all to disk.

75460. Object for pair-wise doclist merging

75461. Tables used by best possible plan

75462. 0

75463. Next registered VFS

75464. This routine is not called unless a write-transaction has already ** been started. The journal file may or may not be open at this point. ** It is never called in the ERROR state.

75465. Unpacked index key

75466. IMP: R-38219-53002

75467. VFS containing this as the xDelete method

75468. now attempt to get the exclusive lock range

75469. Information about the record format

75470. Main database structure

75471. No term was written to this page.

75472. Start of range is constrained

75473. ** Return the current term.

75474. ** Free the RbuObjIter.azTblCol[] and RbuObjIter.abTblPk[] arrays allocated ** by an earlier call to rbuObjIterCacheTableInfo().

75475. ** Given an expression list (which is really the list of expressions ** that form the result set of a SELECT statement) compute appropriate ** column names for a table that would hold the expression list. *** All column names will be unique. *** Only the column names are computed. Column.zType, Column.zColl, ** and other fields of Column are zeroed. *** Return SQLITE_OK on success. If a memory allocation error occurs, ** store NULL in *paCol and 0 in *pnCol and return SQLITE_NOMEM. *** The only guarantee that SQLite makes about column names is that if the ** column has an AS clause assigning it a name, that will be the name used. ** That is the only documented guarantee. However, countless applications ** developed over the years have made baseless assumptions about column names ** and will break if those assumptions changes. Hence, use extreme caution ** when modifying this routine to avoid breaking legacy. *** See Also: generateColumnNames()

75476. **comment:** The cell should normally be sized correctly. However, when moving a ** malformed cell from a leaf page to an interior page, if the cell size ** wanted to be less than 4 but got rounded up to 4 on the leaf, then size ** might be less than 8 (leaf-size + pointer) on the interior node. Hence ** the term after the || in the following assert().

label: code-design

75477. ** File control method. For custom operations on an rbuVfs-file.

75478. Use the standard inner loop.

75479. Mask of snippet terms to highlight

75480. ** Extend level iLvl so that there is room for at least nExtra more ** segments.

75481. cmd ::= PRAGMA nm dbnm LP minus_num RP

75482. 0x50..0x5F

75483. ***** Include vxworks.h in the middle of sqliteInt.h *****

75484. ** 2014 August 30 **** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** OVERVIEW *** The RBU extension requires that the RBU update be packaged as an ** SQLite database. The tables it expects to find are described in ** sqlite3rbu.h. Essentially, for each table xyz in the target database ** that the user wishes to write to, a corresponding data_xyz table is ** created in the RBU database and populated with one row for each row to ** update, insert or delete from the target table. *** The update proceeds in three stages: *** 1) The database is updated. The modified database pages are written ** to a *-oal file. A *-oal file is just like a *-wal file, except ** that it is named "<database>-oal" instead of "<database>-wal". ** Because regular SQLite clients do not look for file named ** "<database>-oal", they go on using the original database in ** rollback mode while the *-oal file is being generated. *** During this stage RBU does not update the database by writing ** directly to the target tables. Instead it creates "impostor" ** tables using the SQLITE_TESTCTRL_IMPOSTER interface that it uses ** to update each b-tree individually. All updates required by each ** b-tree are completed before moving on to the next, and all ** updates are done in sorted key order. *** 2) The "<database>-oal" file is moved to the equivalent "<database>-wal" ** location using a call to rename(2). Before doing this the RBU ** module takes an EXCLUSIVE lock on the database file, ensuring ** that there are no other active readers. *** Once the EXCLUSIVE lock is released, any other database readers ** detect the new *-wal file and read the database in wal mode. At ** this point they see the new version of the database - including ** the updates made as part of the RBU update. *** 3) The new *-wal file is checkpointed. This proceeds in the same way ** as a regular database checkpoint, except that a single frame is ** checkpointed each time sqlite3rbu_step() is called. If the RBU ** handle is closed before the entire *-wal file is checkpointed, ** the

checkpoint progress is saved in the RBU database and the ** checkpoint can be resumed by another RBU client at some point in ** the future. *** POTENTIAL PROBLEMS *** The rename() call might not be portable. And RBU is not currently ** syncing the directory after renaming the file. *** When state is saved, any commit to the *-oal file and the commit to ** the RBU update database are not atomic. So if the power fails at the ** wrong moment they might get out of sync. As the main database will be ** committed before the RBU update database this will likely either just ** pass unnoticed, or result in SQLITE_CONSTRAINT errors (due to UNIQUE ** constraint violations). *** If some client does modify the target database mid RBU update, or some ** other error occurs, the RBU extension will keep throwing errors. It's ** not really clear how to get out of this state. The system could just ** by delete the RBU update database and *-oal file and have the device ** download the update again and start over. *** At present, for an UPDATE, both the new.* and old.* records are ** collected in the rbu_xyz table. And for both UPDATES and DELETES all ** fields are collected. This means we're probably writing a lot more ** data to disk when saving the state of an ongoing update to the RBU ** update database than is strictly necessary. ***

75485. out2: P2 is an output

75486. The JSON parser that contains the TARGET

75487. Join type mask

75488. 1 byte past end of aAll

75489. porter rule condition: (*v*)

75490. Number of tokens desired for snippet

75491. True if sorter record includes seq. no.

75492. int

75493. Set the output variables before returning.

75494. First argument to sqlite3_result_...()

75495. ** This constant should already be defined (in the "WinNT.h" SDK file).

75496. Index in mem3.aPool[] of next free chunk

75497. Set variable iNext to the next available segdir index at level iLevel.

75498. xBestIndex - Determine search strategy

75499. Where to jump when NULLs seen in step 2

75500. Require a write-lock on the table to perform this operation

75501. Size of sibling on the right

75502. COLLATE, AS, or UNLIKELY

75503. Create a new thread

75504. insert_cmd ::= REPLACE

75505. If the pager is in CACHEMOD state, then there must be a copy of this ** page in the pager cache. In this case just update the pager cache, ** not the database file.

The page is left marked dirty in this case. *** An exception to the above rule: If the database is in no-sync mode ** and a page is moved during an incremental vacuum then the page may ** not be in the pager cache. Later: if a malloc() or IO error occurs ** during a Movepage() call, then the page may not be in the cache ** either. So the condition described in the above paragraph is not ** assertable. *** If in WRITER_DBMOD, WRITER_FINISHED or OPEN state, then we update the ** pager cache if it exists and the main file. The page is then marked ** not dirty. Since this code is only executed in PAGER_OPEN state for ** a hot-journal rollback, it is guaranteed that the page-cache is empty ** if the pager is in OPEN state. *** Ticket #1171: The statement journal might contain page content that is ** different from the page content at the start of the transaction. ** This occurs when a page is changed prior to the start of a statement ** then changed again within the statement. When rolling back such a ** statement we must not write to the original database unless we know ** for certain that original page contents are synced into the main rollback ** journal. Otherwise, a power loss might leave modified data in the ** database file without an entry in the rollback journal that can ** restore the database to its original form. Two conditions must be ** met before writing to the database files. (1) the database must be ** locked. (2) we know that the original page content is fully synced ** in the main journal either because the page is not in cache or else ** the page is marked as needSync==0. *** 2008-04-14: When attempting to vacuum a corrupt database file, it ** is possible to fail a statement on a database that does not yet exist. ** Do not attempt to write if database file has never been opened.

75506. Number of UChar elements in pInput

75507. SQL statement to delete rows

75508. 108

75509. ** Erase column-cache entry number i

75510. ** Process a pragma statement. *** PRAGMA [schema].jid [= value] ** ** The identifier might also be a string. The value is a string, and ** identifier, or a number. If minusFlag is true, then the value is ** a number that was preceded by a minus sign. ** ** If the left side is "database.id" then pId1 is the database name ** and pId2 is the id. If the left side is just "id" then pId1 is the ** id and pId2 is any empty string.

75511. ***** Include fts3Int.h in the middle of fts3.c *****

75512. Version 3.8.7 and later

75513. ** Check a unixFile that is a database. Verify the following: ** ** (1) There is exactly one hard link on the file ** (2) The file is not a symbolic link ** (3) The file has not been renamed or unlinked ** ** Issue sqlite3_log(SQLITE_WARNING,...) messages if anything is not right.

75514. Primary key declaration for imposter

75515. OUT: Number of tokens returned before this one

75516. Index to write to

75517. full

75518. ** Information held in the "sqlite3" database connection object and used ** to manage user authentication.

75519. Context of the fixation

75520. Code constant expressions that where factored out of inner loops

75521. **comment:** The temp-database schema is allocated differently from the other schema ** objects (using sqliteMalloc() directly, instead of sqlite3BtreeSchema()). ** So it needs to be freed here. Todo: Why not roll the temp schema into ** the same sqliteMalloc() as the one that allocates the database ** structure?

label: code-design

75522. Name of attached database (or NULL)

75523. 274

75524. Tell the code in notify.c that the connection no longer holds any ** locks and does not require any further unlock-notify callbacks.

75525. Current FTS5 configuration

75526. Number of pages on the freelist initially

75527. ** Compare the term that the Fts3SegReader object passed as the first argument ** points to with the term specified by arguments zTerm and nTerm. ** ** If the pSeg iterator is already at EOF, return 0. Otherwise, return ** -ve if the pSeg term is less than zTerm/nTerm, 0 if the two terms are ** equal, or +ve if the pSeg term is greater than zTerm/nTerm.

75528. rbu VFS shim methods

75529. 214

75530. ** The yDbMask datatype for the bitmask of all attached databases.

75531. ** Threading interface

75532. For looping through segments

75533. String or blob

75534. Value inserted into rank column

75535. Number of entries in aArg[]

75536. Database to detach at end of vacuum

75537. True if iterator is "rowid DESC"

75538. ** Execute as much of a VDBE program as we can. ** This is the core of sqlite3_step().

75539. OP_ResultRow called with zero columns

75540. 2

75541. defined(InterlockedCompareExchange)

75542. Write high-water mark here

75543. Associated index cursor from which to read

75544. Opcode: IdxDelete P1 P2 P3 *** Synopsis: key=r[P2@P3] *** The content of P3 registers starting at register P2 form ** an unpacked index key. This opcode removes that entry from the ** index opened by cursor P1.

75545. The table trigger pTrigger is attached to

75546. If this call is to check if a *-wal file associated with an RBU target ** database connection exists, and the RBU update is in RBU_STAGE_OAL, ** the following special handling is activated: *** a) if the *-wal file does exist, return SQLITE_CANTOPEN. This ** ensures that the RBU extension never tries to update a database ** in wal mode, even if the first page of the database file has ** been damaged. *** b) if the *-wal file does not exist, claim that it does anyway, ** causing SQLite to call xOpen() to open it. This call will also ** be intercepted (see the rbuVfsOpen() function) and the *-oal ** file opened instead.

75547. ** Token pToken is an incrementally loaded token that is part of a ** multi-token phrase. Advance it to the next matching document in the ** database and populate output variable *p with the details of the new ** entry. Or, if the iterator has reached EOF, set *pbEof to true. *** If an error occurs, return an SQLite error code. Otherwise, return ** SQLITE_OK.

75548. SegmentNode handle

75549. synopsis: r[P2..P3]=NULL

75550. zNum is less than 9223372036854775808 so it fits

75551. For client use (any custom purpose)

75552. If the next character is a digit, this is a floating point ** number that begins with ". ". Fall thru into the next case

75553. Fall thru

75554. ** Scan through the expression pExpr. Replace every reference to ** a column in table number iTable with a copy of the iColumn-th ** entry in pEList. (But leave references to the ROWID column ** unchanged.) *** This routine is part of the flattening procedure. A subquery ** whose result set is defined by pEList appears as entry in the ** FROM clause of a SELECT such that the Vdbe cursor assigned to that ** FORM clause entry is iTable. This routine makes the necessary ** changes to pExpr so that it refers directly to the source table ** of the subquery rather the result set of the subquery.

75555. fts3.c

75556. A bitfield type for use inside of structures. Always follow with :N where ** N is the number of bits.

75557. 1070

75558. Current state number

75559. Check that the leaf contains at least one term, and that it is equal ** to or larger than the split-key in zIdxTerm. Also check that if there ** is also a rowid pointer within the leaf page header, it points to a ** location before the term.

75560. Update allocator performance statistics.

75561. iReg is a temp register that needs to be freed

75562. Generates a warning - but it always works

75563. The cell index area

75564. Idx of column in child table

75565. Mark the database file just opened as an RBU target database. If ** this call returns SQLITE_NOTFOUND, then the RBU vfs is not in use. ** This is an error.

75566. Non-NULL for UPDATE operations

75567. afp style keeps a reference to the db path in the filePath field ** of the struct

75568. Minimum number of pages reserved

75569. ** Resize the block of memory pointed to by p to n bytes. If the ** resize fails, set the mallocFailed flag in the connection object.

75570. Table this step applies to

75571. 81

75572. List of triggers stored in pSchema

75573. 10's digit

75574. Replace references to this table

75575. Number of ORDER BY terms satisfied by indices

75576. Parameter number to bind to

75577. Check for mismatched parenthesis

75578. ** 2009 November 25 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used to insert the values of host parameters ** (aka "wildcards") into the SQL text output by sqlite3_trace(). *** The Vdbe parse-tree explainer is also found here.

75579. read the existing conch file

75580. Needed for various definitions...

75581. 22

75582. A single constraint term

75583. ** Currently, SQLite never calls the LockFileEx function without wanting the ** call to fail immediately if the lock cannot be obtained.

75584. ** CAPI3REF: Write Data Into A BLOB Incrementally ** METHOD: sqlite3_blob *** ^^(This function is used to write data into an open [BLOB handle] from a ** caller-supplied buffer. N bytes of data are copied from the buffer Z ** into the open BLOB, starting at offset iOffset.)^ *** ^^(On success, sqlite3_blob_write() returns SQLITE_OK. ** Otherwise, an [error code] or an [extended error code] is returned.)^ *** ^^(Unless SQLITE_MISUSE is returned, this function sets the ** [database connection] error code and message accessible via ** [sqlite3_errcode()] and [sqlite3_errmsg()] and related functions. *** ^^(If the [BLOB handle] passed as the first argument was not opened for ** writing (the flags parameter to [sqlite3_blob_open()] was zero), ** this function returns [SQLITE_READONLY]. *** This function may only modify the contents of the BLOB; it is ** not possible to increase the size of a BLOB using this API. ** ^If offset iOffset is less than N bytes from the end of the BLOB, ** [SQLITE_ERROR] is returned and no data is written. The size of the ** BLOB (and hence the maximum value of N+iOffset) can be determined ** using the [sqlite3_blob_bytes()] interface. ^If N or iOffset are less ** than zero [SQLITE_ERROR] is returned and no data is written. *** ^^(An attempt to write to an expired [BLOB handle] fails with an ** error code of [SQLITE_ABORT]. ^Writes to the BLOB that occurred ** before the [BLOB handle] expired are not rolled back by the ** expiration of the handle, though of course those changes might ** have been overwritten by the statement that expired the BLOB handle ** or by other independent statements. *** This routine only works on a [BLOB handle] which has been created ** by a prior successful call to [sqlite3_blob_open()] and which has not ** been closed by [sqlite3_blob_close()]. Passing any other pointer in ** to this routine results in undefined and probably undesirable behavior. *** See also: [sqlite3_blob_read()].

75585. We have found a candidate table and column. Check to see if that ** table and column is common to every term in the OR clause

75586. **comment:** ** Value used to signify the end of an position-list. This is safe because ** it is not possible to have a document with 2^{31} terms.
label: documentation

75587. Count the number of columns that will be added to the index ** and used to match WHERE clause constraints

75588. Mask of phrases already covered

75589. Estimated number of rows generated by this path

75590. TRANSACTION

75591. ** Delete the file at zPath. If the dirSync argument is true, fsync() ** the directory after deleting the file.

75592. Write the hint values into the %_stat table for the next incr-merger

75593. ***** End of the no-op lock implementation *****

75594. The right-hand side (RHS) of the IN operator

75595. fstat() info for database file

75596. Database file holding the shared memory

75597. 1190

75598. Figure out the type of table this step will deal with.

75599. First key value passed to hook

75600. The sqlite_stat[134] table already exists. Delete all rows.

75601. Constant expressions

75602. Load the full doclist for the phrase into memory.

75603. Open page 1 of the file for writing.

75604. True if opening an ephemeral, temporary database

75605. ** This is the xFilter interface for the virtual table. See ** the virtual table xFilter method documentation for additional ** information. ** ** There are three possible query strategies: ** ** 1. Full-text search using a MATCH operator. ** 2. A by-rowid lookup. ** 3. A full-table scan.

75606. First register in VUpdate arg array

75607. in1

75608. The write cursors opened by WHERE_ONEPASS

75609. ** Populate pCsr->aMatchinfo[] with data for the current row. The ** 'matchinfo' data is an array of 32-bit unsigned integers (C type u32).

75610. X==Ei (form A) or just Ei (form B)

75611. Array used to calculate phrase freq.

75612. Index of loop to report on

75613. IN/OUT: SegmentWriter handle

75614. Root page of index

75615. ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This header file defines the interface that the sqlite page cache ** subsystem. The page cache subsystem reads and writes a file a page ** at a time and provides a journal for rollback.

75616. Move the page to the head of the dirty list. If p->pDirtyPrev==0, ** then page p is already at the head of the dirty list and the ** following call would be a no-op. Hence the OPTIMIZATION-IF-FALSE ** tag above.

75617. EVIDENCE-OF: R-25451-61125 The SQLITE_CONFIG_URI option takes a single ** argument of type int. If non-zero, then URI handling is globally ** enabled. If the parameter is zero, then URI handling is globally ** disabled.

75618. Seg-reader cursor for this term

75619. Preserve SQL text

75620. This strategy involves a two rowid lookups on an B-Tree structures ** and then a linear search of an R-Tree node. This should be ** considered almost as quick as a direct rowid lookup (for which ** sqlite uses an internal cost of 0.0). It is expected to return ** a single row.

75621. !defined(SQLITE_RTREE_INT_ONLY)

75622. Version 3.18.0 and later

75623. Tokenizer object

75624. ** Return true if the iterator passed as the only argument is at EOF.

75625. a: p0<<14 | p2 (masked)

75626. Used by DOCID_CMP() macro

75627. Name of new table or NULL

75628. ** This routine implements a busy callback that sleeps and tries ** again until a timeout value is reached. The timeout value is ** an integer number of milliseconds passed in as the first ** argument.

75629. ** CAPI3REF: Register A Virtual Table Implementation ** METHOD: sqlite3 ** ** ^These routines are used to register a new [virtual table module] name. ** ^Module names must be registered before ** creating a new [virtual table] using the module and before using a ** preexisting [virtual table] for the module. ** ** ^The module name is registered on the [database connection] specified ** by the first parameter. ^The name of the module is given by the ** second parameter. ^The third parameter is a pointer to ** the implementation of the [virtual table module]. ^The fourth ** parameter is an arbitrary client data pointer that is passed through ** into the [xCreate] and [xConnect] methods of the virtual table module ** when a new virtual table is being created or reinitialized. ** ** ^The sqlite3_create_module_v2() interface has a fifth parameter which ** is a pointer to a destructor for the pClientData. ^SQLite will ** invoke the destructor function (if it is not NULL) when SQLite ** no longer needs the pClientData pointer. ^The destructor will also ** be invoked if the call to sqlite3_create_module_v2() fails. ** ^The sqlite3_create_module() ** interface is equivalent to sqlite3_create_module_v2() with a NULL ** destructor.

75630. !SQLITE_OS_WINCE && !SQLITE_OS_WINRT && SQLITE_WIN32_USE_UUID

75631. For an in-memory database, make sure the original page continues ** to exist, in case the transaction needs to roll back. Use pPgOld ** as the original page since it has already been allocated.

75632. aDistance array

75633. m

75634. ** PRAGMA [schema.]page_size ** PRAGMA [schema.]page_size=N ** ** The first form reports the current setting for the ** database page size in bytes. The second form sets the ** database page size value. The value can only be set if ** the database has not yet been created.

75635. Update statement to modify idx values

75636. Divider cells in pParent

75637. ** Clear any and all virtual-table information from the Table record. ** This routine is called, for example, just before deleting the Table ** record. ** ** Since it is a virtual-table, the Table structure contains a pointer ** to the head of a linked list of VTable structures. Each VTable ** structure is associated with a single sqlite3* user of the schema. ** The reference count of the VTable structure associated with database ** connection db is decremented immediately (which may lead to the ** structure being xDisconnected and free). Any other VTable structures ** in the list are moved to the sqlite3.pDisconnect list of the associated ** database connection.

75638. ** Maximum number of pages in one database file. ** ** This is really just the default value for the max_page_count pragma. ** This value can be lowered (or raised) at run-time using that the ** max_page_count macro.

75639. ** Deallocate all memory associated with a WhereAndInfo object.

75640. Absolute level to open

75641. Loop through all columns of the table being considered for snippets. ** If the iCol argument to this function was negative, this means all ** columns of the FTS3 table. Otherwise, only column iCol is considered.

75642. The expression to be fixed to one database

75643. Current size of wal-index file

75644. **comment:** ** If we get to this point, the path name should almost certainly be a purely ** relative one (i.e. not a UNC name, not absolute, and not volume relative).
label: code-design

75645. * This is the extra space for the initial size of the Win32-specific heap, * in bytes. This value may be zero.

75646. Minimum cell size is 4

75647. FILETIME structure is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (= JD 2305813.5).

75648. Look for a valid schema=? constraint. If found, change the idxNum to ** 1 and request the value of that constraint be sent to xFilter. And ** lower the cost estimate to encourage the constrained version to be ** used.

75649. If an error has occurred, mark the session object as failed.

75650. Bitmask identifying table iTab

75651. Registers to allocate

75652. New rowid

75653. **comment:** Do not code a test for this constraint
label: test

75654. True if really sorted (not just grouped)

75655. One less than frame number of aPgn[1]

75656. Do not gather statistics on views or virtual tables

75657. Delete the master journal file. This commits the transaction. After ** doing this the directory is synced again before any individual ** transaction files are deleted.

75658. Buffer to fold text into

75659. ** Convert a UTF-16 string in the native encoding into a UTF-8 string. ** Memory to hold the UTF-8 string is obtained from sqlite3_malloc and must ** be freed by the calling function. ** ** NULL is returned if there is an allocation error.

75660. xFilter

75661. Could not find an existing table or index to use as the RHS b-tree. ** We will have to generate an ephemeral table to do the job.

75662. True if the AUTOINCREMENT keyword is present

75663. xColumn

75664. ** Retry ftruncate() calls that fail due to EINTR ** ** All calls to ftruncate() within this file should be made through ** this wrapper. On the Android platform, bypassing the logic below ** could lead to a corrupt database.

75665. ** Compare two ExprList objects. Return 0 if they are identical and ** non-zero if they differ in any way. ** ** If any subelement of pB has Expr.iTable==(-1) then it is allowed ** to compare equal to an equivalent element in pA with Expr.iTable==iTab. ** ** This routine might return non-zero for equivalent ExprLists. The ** only consequence will be disabled optimizations. But this routine ** must never return 0 if the two ExprList objects are different, or ** a malfunction will result. ** ** Two NULL pointers are considered to be the same. But a NULL pointer ** always differs from a non-NUL pointer.

75666. Base class. Must be first

75667. A-overwrites-OP

75668. Size decreases

75669. All threads share a single random number generator. ** This structure is the current state of the generator.

75670. Initial root node

75671. add_column_fullname ::= fullname

75672. ** 2005 February 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file contains C code routines that used to generate VDBE code ** that implements the ALTER TABLE command.

75673. The table we are inserting into

75674. ** The page handle passed as the first argument refers to a dirty page ** with a page number other than iNew. This function changes the page's ** page number to iNew and sets the value of the PgHdr.flags field to ** the value passed as the third parameter.

75675. ** The names of the following types declared in vdbeInt.h are required ** for the VdbeOp definition.

75676. ** Functions generated by lemon from fts5parse.y.

75677. vtabargtoken ::= ANY

75678. During the search for the best function definition, this procedure ** is called to test how well the function passed as the first argument ** matches the request for a function with nArg arguments in a system ** that uses encoding enc. The value returned indicates how well the ** request is matched. A higher value indicates a better match. ** ** If nArg is -1 that means to only return a match (non-zero) if p->nArg ** is also -1. In other words, we are searching for a function that ** takes a variable number of arguments. ** ** If nArg is -2 that means that we are searching for any function ** regardless of the number of arguments it uses, so return a positive ** match score for any ** ** The returned value is always between 0 and 6, as follows: ** ** 0: Not a match. ** 1: UTF8/16 conversion required and function takes any number of arguments. ** 2: UTF16 byte order change required and function takes any number of args. ** 3: encoding matches and function takes any number of arguments ** 4: UTF8/16 conversion required - argument count matches exactly ** 5: UTF16 byte order conversion required - argument count matches exactly ** 6: Perfect match: encoding and argument count match exactly. ** ** If nArg==(-2) then any function with a non-null xSFunc is ** a perfect match and any function with xSFunc NULL is ** a non-match.

75679. Column number in the sorting index

75680. Determine if doclists may be loaded from disk incrementally. This is ** possible if the bOptOk argument is true, the FTS doclists will be ** scanned in forward order, and the phrase consists of ** MAX_INCR_PHRASE_TOKENS or fewer tokens, none of which are are "^first" ** tokens or prefix tokens that cannot use a prefix-index.

75681. Open the sqlite_stat[134] tables for writing.

75682. Database handle session is attached to

75683. 630

75684. Verify property (1)

75685. Ordinary Step 3, for the case where FALSE and NULL are distinct

75686. Pointer to int

75687. (321) kwcolumn_opt ::=

75688. Name of table that the key points to (aka: Parent)

75689. **comment:** ** CAPI3REF: One-Step Query Execution Interface ** METHOD: sqlite3_exec() interface is a convenience wrapper around ** [sqlite3_prepare_v2()], [sqlite3_step()], and [sqlite3_finalize()], ** that allows an application to run multiple statements of SQL ** without having to use a lot of C code. ** ** ^The sqlite3_exec() interface runs zero or more UTF-8 encoded, ** semicolon-separate SQL statements passed into its 2nd argument, ** in the context of the [database connection] passed in as its 1st ** argument. ^If the callback function of the 3rd argument to ** sqlite3_exec() is not NULL, then it is invoked for each result row ** coming out of the evaluated SQL statements. ^The 4th argument to ** sqlite3_exec() is relayed through to the 1st argument of each ** callback invocation. ^If the callback pointer to sqlite3_exec() ** is NULL, then no callback is ever invoked and result rows are ** ignored. ** ** ^If an error occurs while evaluating the SQL statements passed into ** sqlite3_exec(), then execution of the current statement stops and ** subsequent statements are skipped. ^If the 5th parameter to sqlite3_exec() ** is not NULL then any error message is written into memory obtained ** from [sqlite3_malloc()] and passed back through the 5th parameter. ** To avoid memory leaks, the application should invoke [sqlite3_free()] ** on error message strings returned through the 5th parameter of ** sqlite3_exec() after the error message string is no longer needed. ** ^If the 5th parameter to sqlite3_exec() is not NULL and no errors ** occur, then sqlite3_exec() sets the pointer in its 5th parameter to ** NULL before returning. ** ** ^If an sqlite3_exec() callback returns non-zero, the sqlite3_exec() ** routine returns SQLITE_ABORT without invoking the callback again and ** without running any subsequent SQL statements. ** ** ^The 2nd argument to the sqlite3_exec() callback function is the ** number of columns in the result. ^The 3rd argument to the sqlite3_exec() ** callback is an array of pointers to strings obtained as if from ** [sqlite3_column_text()], one for each column. ^If an element of a ** result row is NULL then the corresponding string pointer for the ** sqlite3_exec() callback is a NULL pointer. ^The 4th argument to the ** sqlite3_exec() callback is an array of pointers to strings where each ** entry represents the name of corresponding result column as obtained ** from [sqlite3_column_name()]. ** ** ^If the 2nd parameter to sqlite3_exec() is a NULL pointer, a pointer ** to an empty string, or a pointer that contains only whitespace and/or ** SQL comments, then no SQL statements are evaluated and the database ** is not changed. ** ** Restrictions: ** ** ** The application must ensure that the 1st parameter to sqlite3_exec() ** is a valid and open [database connection]. ** The application must not close the [database connection] specified by ** the 1st parameter to sqlite3_exec() while sqlite3_exec() is running. ** The application must not modify the SQL statement text passed into ** the 2nd parameter of sqlite3_exec() while sqlite3_exec() is running. **

label: code-design

75690. ** Declarations used for tracing the operating system interfaces.

75691. **comment:** The term to be analyzed

label: code-design

75692. Last UPDATE used (for PK b-tree updates only), or NULL.

75693. ** Add a new element to the end of an expression list. If pList is ** initially NULL, then create a new expression list. ** ** If a memory allocation error occurs, the entire list is freed and ** NULL is returned. If non-NUL is returned, then it is guaranteed ** that the new entry was successfully appended.

75694. Pretend we are in EXCLUSIVE mode

75695. ***** GENERATED CODE STARTS HERE (mkportersteps.tcl)

75696. 308

75697. sqlite3_step() verifies this

75698. True if we need to scan in reverse order

75699. ***** Include keywordhash.h in the middle of tokenize.c *****

75700. ***** Include fts3_hash.h in the middle of fts3Int.h *****

75701. This error condition is now caught prior to reaching this function

75702. Offset of pFd to begin writing at

75703. defined(SQLITE_ENABLE_UPDATE_DELETE_LIMIT)

75704. Number of entries in a[]

75705. The name context used to resolve the name

75706. SQLITE_ERROR

75707. Allocate a bitvec to use to store the set of pages rolled back

75708. Offset into aPrefix[] of the prefix string

75709. Pointer to value 2

75710. Array of child nodes

75711. ** CAPI3REF: String Comparison *** ^The [sqlite3_stricmp()] and [sqlite3_strnicmp()] APIs allow applications ** and extensions to compare the contents of two buffers containing UTF-8 ** strings in a case-independent fashion, using the same definition of "case ** independence" that SQLite uses internally when comparing identifiers.

75712. ** Do various sanity checks on a single page of a tree. Return ** the tree depth. Root pages return 0. Parents of root pages ** return 1, and so forth. *** These checks are done: ** ** 1. Make sure that cells and freeblocks do not overlap ** but combine to completely cover the page. ** 2. Make sure integer cell keys are in order. ** 3. Check the integrity of overflow pages. ** 4. Recursively call checkTreePage on all children. ** 5. Verify that the depth of all children is the same.

75713. The OR-clause broken out into subterms

75714. ** Deregister and destroy an RBU vfs created by an earlier call to ** sqlite3rbu_create_vfs().

75715. ** Possible values for the sqlite3.flags. ** ** Value constraints (enforced via assert()): ** SQLITE_FullFSync == PAGER_FULLFSYNC **
SQLITE_CkptFullFSync == PAGER_CKPT_FULLFSYNC ** SQLITE_CacheSpill == PAGER_CACHE_SPILL

75716. Make a copy of the entire SELECT statement that defines the view. ** This will force all the Expr.token.z values to be dynamically ** allocated rather than point to the input string - which means that ** they will persist after the current sqlite3_exec() call returns.

75717. adjust the sign of significand

75718. Table column number

75719. 660

75720. If MEM_Dyn is set then Mem.xDel!=0. ** Mem.xDel might not be initialized if MEM_Dyn is clear.

75721. Database structure for this iterator

75722. ** Only set the lastErrno if the error code is a real error and not ** a normal expected return code of SQLITE_BUSY or SQLITE_OK

75723. Note: the calls to BtreeKeyFetch() and DataFetch() below assert() ** that both the BtShared and database handle mutexes are held.

75724. ** FTS maintains a separate indexes for each language-id (a 32-bit integer). ** Within each language id, a separate index is maintained to store the ** document terms, and each configured prefix size (configured the FTS ** "prefix=" option). And each index consists of multiple levels ("relative ** levels"). ** ** All three of these values (the language id, the specific index and the ** level within the index) are encoded in 64-bit integer values stored ** in the %_segdir table on disk. This function is used to convert three ** separate component values into the single 64-bit integer value that ** can be used to query the %_segdir table. *** Specifically, each language-id/index combination is allocated 1024 ** 64-bit integer level values ("absolute levels"). The main terms index ** for language-id 0 is allocate values 0-1023. The first prefix index ** (if any) for language-id 0 is allocated values 1024-2047. And so on. ** Language 1 indexes are allocated immediately following language 0. ** ** So, for a system with nPrefix prefix indexes configured, the block of ** absolute levels that corresponds to language-id iLangid and index ** iIndex starts at absolute level ((iLangid * (nPrefix+1) + iIndex) * 1024).

75725. ** Return TRUE if a KeyInfo object can be change. The KeyInfo object ** can only be changed if this is just a single reference to the object. ** ** This routine is used only inside of assert() statements.

75726. **comment:** ** Open an RBU handle to perform an RBU vacuum on database file zTarget. ** An RBU vacuum is similar to SQLite's built-in VACUUM command, except ** that it can be suspended and resumed like an RBU update. ** ** The second argument to this function identifies a database in which ** to store the state of the RBU vacuum operation if it is suspended. The ** first time sqlite3rbu_vacuum() is called, to start an RBU vacuum ** operation, the state database should either not exist or be empty ** (contain no tables). If an RBU vacuum is suspended by calling ** sqlite3rbu_close() on the RBU handle before sqlite3rbu_step() has ** returned SQLITE_DONE, the vacuum state is stored in the state database. ** The vacuum can be resumed by calling this function to open a new RBU ** handle specifying the same target and state databases. ** ** If the second argument passed to this function is NULL, then the ** name of the state database is " <database>-vacuum", where <database> ** is the name of the target database file. In this case, on UNIX, if the ** state database is not already present in the file-system, it is created ** with the same permissions as the target db is made. ** ** This function does not delete the state database after an RBU vacuum ** is completed, even if it created it. However, if the call to ** sqlite3rbu_close() returns any value other than SQLITE_OK, the contents ** of the state tables within the state database are zeroed. This way, ** the next call to sqlite3rbu_vacuum() opens a handle that starts a ** new RBU vacuum operation. ** ** As with sqlite3rbu_open(), Zipvfs users should rever to the comment ** describing the sqlite3rbu_create_vfs() API function below for ** a description of the complications associated with using RBU with ** zipvfs databases.

label: code-design

75727. ** Bits of the Pager.doNotSpill flag. See further description below.

75728. Mark the page that is about to be modified as dirty.

75729. OUT: Number of FK violations

75730. Used when p4type is P4_FUNCDEF

75731. 8x

75732. If more than one term matches the prefix, sort the Fts3HashElem ** objects in term order using qsort(). This uses the same comparison ** callback as is used when flushing terms to disk.

75733. Generating CHECK constraints or inserting into partial index

75734. **comment:** NOT USED

label: code-design

75735. ** This function is called to release all dynamic resources held by the ** merge-writer object pWriter, and if no error has occurred, to flush ** all outstanding node buffers held by pWriter to disk. ** ** If *pRc is not SQLITE_OK when this function is called, then no attempt ** is made to write any data to disk. Instead, this function serves only ** to release outstanding resources. ** ** Otherwise, if *pRc is initially SQLITE_OK and an error occurs while ** flushing buffers to disk, *pRc is set to an SQLite error code before ** returning.

75736. True if yymajor has invoked an error

75737. nBlob==0 in detail=none mode.

75738. ** Invoke the profile callback. This routine is only called if we already ** know that the profile callback is defined and needs to be invoked.

75739. ** This routine transforms the internal text encoding used by pMem to ** desiredEnc. It is an error if the string is already of the desired ** encoding, or if *pMem does not contain a string value.

75740. Update the write-counter. While doing so, set nWork.

75741. EVIDENCE-OF: R-55530-52930 In a well-formed b-tree page, there will ** always be at least one cell before the first freeblock.

75742. Space to hold MEM_Str or MEM_Blob if szMalloc>0

75743. ** 2013 November 25 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains pre-processor directives related to operating system ** detection and/or setup.

75744. Various flags. EP_* See below

75745. SQLITE_ENABLE_MEMSYS3

75746. Open the database file. If the btree is successfully opened, use ** it to obtain the database schema. At this point the schema may ** or may not be initialized.

75747. Calculate the estimated cost based on the flags set in idxFlags.

75748. ** 2007 August 28 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the C functions that implement mutexes for pthreads

75749. ** pExpr is a comparison operator. Return the type affinity that should ** be applied to both operands prior to doing the comparison.

75750. ** Return non-zero if the bit in the IntegrityCk.aPgRef[] array that ** corresponds to page iPg is already set.

75751. The selected WhereLoop object

75752. Opcode: FkIfZero P1 P2 ** ** ** Synopsis: if fkctr[P1]==0 goto P2 ** ** This opcode tests if a foreign key constraint-counter is currently zero. ** If so, jump to instruction P2. Otherwise, fall through to the next ** instruction. ** ** If P1 is non-zero, then the jump is taken if the database constraint-counter ** is zero (the one that counts deferred constraint violations). If P1 is ** zero, the jump is taken if the statement constraint-counter is zero ** (immediate foreign key constraint violations).

75753. ** Set the maximum page count for a database if mxPage is positive. ** No changes are made if mxPage is 0 or negative. ** Regardless of the value of mxPage, return the maximum page count.

75754. ** Return true if any register in the range iFrom..iTo (inclusive) ** is used as part of the column cache. ** ** This routine is used within assert() and testcase() macros only ** and does not appear in a normal build.

75755. SQL statement passed to declare_vtab()

75756. maximum depth of the parser stack
75757. Right-child page number (or 0)
75758. Pass the expected checksum down to the FTS index module. It will ** verify, amongst other things, that it matches the checksum generated by ** inspecting the index itself.
75759. Registers holding data and rowid
75760. ** Implementation of xRollback(). Discard the contents of the pending-terms ** hash-table. Any changes made to the database are reverted by SQLite.
75761. Total size of one pcache line
75762. ** There are 2 different modes of operation for a hash table: ** ** FTS3_HASH_STRING pKey points to a string that is nKey bytes long ** (including the null-terminator, if any). Case ** is respected in comparisons. ** ** FTS3_HASH_BINARY pKey points to binary data nKey bytes long. ** memcmp() is used to compare keys. ** ** A copy of the key is made if the copyKey parameter to fts3HashInit is 1.
75763. 1550
75764. ** If the last opcode is "op" and it is not a jump destination, ** then remove it. Return true if and only if an opcode was removed.
75765. xSavepoint
75766. pWal->readLock is usually set, but might be -1 if there was a ** prior error while attempting to acquire a read-lock. This cannot ** happen if the connection is actually in exclusive mode (as no xShmLock ** locks are taken in this case). Nor should the pager attempt to ** upgrade to exclusive-mode following such an error.
75767. Set TRUE at EOF. See false for more content
75768. ** Initialize the fts3 extension. If this extension is built as part ** of the sqlite library, then this function is called directly by ** SQLite. If fts3 is built as a dynamically loadable extension, this ** function is called by the sqlite3_extension_init() entry point.
75769. ** Return the SQL associated with a prepared statement
75770. ** If the current doclist-index accumulating in pWriter->aDlidx[] is large ** enough, flush it to disk and return 1. Otherwise discard it and return ** zero.
75771. Copy of BtShared.maxLocal or BtShared.maxLeaf
75772. **comment:** One of the P4_xxx constants for p4
label: code-design
75773. ** This function does all the work for both the xConnect and xCreate methods. ** These tables have no persistent representation of their own, so xConnect ** and xCreate are identical operations. ** ** argv[0]: module name ** argv[1]: database name ** argv[2]: table name ** argv[3]: first argument (tokenizer name)
75774. Scan query (fts5vocab)
75775. Names of columns for pragmas that return multi-column result ** or that return single-column results where the name of the ** result column is different from the name of the pragma
75776. ** This object is a convenience wrapper holding all information needed ** to construct WhereLoop objects for a particular query.
75777. ** A macro to discover the encoding of a database.
75778. Size of column value in tokens
75779. Setting isLeftJoin to -1 causes OP_IfNullRow opcodes to be generated for ** every reference to any result column from subquery in a join, even though ** they are not necessary. This will stress-test the OP_IfNullRow opcode.
75780. 312
75781. Populate this object
75782. Value of 'pgOffset' column
75783. Link list of trigger program steps
75784. ** The zeroblob(N) function returns a zero-filled blob of size N bytes.
75785. Invalidate all ephemeral cursor row caches
75786. Generate code to skip over the creation and initialization of the ** transient index on 2nd and subsequent iterations of the loop.
75787. ** If a Win32 native heap has been configured, this function will attempt to ** compact it. Upon success, SQLITE_OK will be returned. Upon failure, one ** of SQLITE_NOMEM, SQLITE_ERROR, or SQLITE_NOTFOUND will be returned. The ** "pnLargest" argument, if non-zero, will be used to return the size of the ** largest committed free block in the heap, in bytes.
75788. "rowid" column of stat[34] entry
75789. duplicate the FROM clause as it is needed by both the DELETE/UPDATE tree ** and the SELECT subtree.
75790. 250
75791. ASC or DESC for INTEGER PRIMARY KEY
75792. The pSubWInfo->untestedTerms flag means that this OR term ** contained one or more AND term from a notReady table. The ** terms from the notReady table could not be tested and will ** need to be tested later.
75793. Operation performed by this node
75794. Which column of the vector to return
75795. Comparison result if keys are equal
75796. Write the averages record
75797. ** Set the SegReader to point to the first docid in the doclist associated ** with the current term.
75798. Obtain a RESERVED lock on the database file. If the exFlag parameter ** is true, then immediately upgrade this to an EXCLUSIVE lock. The ** busy-handler callback can be used when upgrading to the EXCLUSIVE ** lock, but not when obtaining the RESERVED lock.
75799. TEMPORARY
75800. True if there is a doclist-index
75801. Used to iterate name contexts
75802. The routine to run as a thread
75803. ** If the following global variable points to a string which is the ** name of a directory, then that directory will be used to store ** all database files specified with a relative pathname. ** ** See also the "PRAGMA data_store_directory" SQL command.
75804. Database only autoinc table
75805. Now begin substituting subquery result set expressions for ** references to the iParent in the outer query. ** ** Example: ** ** SELECT a+5, b*10 FROM (SELECT x*3 AS a, y+10 AS b FROM t1) WHERE a>b; ** ______ subquery ______ / ** ______ outer query

/ ** ** We look at every expression in the outer query and every place we see ** "a" we substitute "x*3" and every place we see "b" we substitute "y+10".
75806. Mask of FTSS5INDEX_QUERY_X flags
75807. True if callback data is initialized
75808. automatically defined by SQLITE_ENABLE_FTS4
75809. The ONEPASS_DESIRED flags never occurs together with ORDER BY
75810. Text of the error report
75811. Number of bytes in string zName
75812. OUT: Number of phrases in query
75813. Compute collating sequences used by ** temporary tables needed to implement the compound select. ** Attach the KeyInfo structure to all temporary tables. **
** This section is run by the right-most SELECT statement only. ** SELECT statements to the left always skip this part. The right-most ** SELECT might also skip this part if it has no ORDER BY clause and ** no temp tables are required.
75814. Forward reference
75815. WITH => ID
75816. ** Allocate a new porter tokenizer. Return a pointer to the new ** tokenizer in *ppModule
75817. SQL-style comments from "--" to end of line
75818. Return the size of an allocation
75819. **comment:** ** Allocate memory that is to be used and released right away. ** This routine is similar to alloca() in that it is not intended ** for situations where the memory might be held long-term. This ** routine is intended to get memory to old large transient data ** structures that would not normally fit on the stack of an ** embedded processor.
label: code-design

75820. ** Generate code for the SELECT statement given in the p argument. *** The results are returned according to the SelectDest structure. ** See comments in sqliteInt.h for further information. *** This routine returns the number of errors. If any errors are ** encountered, then an appropriate error message is left in ** pParse->zErrMsg. *** This routine does NOT free the Select structure passed in. The ** calling function needs to do that.

75821. ** Constants used for locking

75822. Reference page

75823. ** Return TRUE if zTable is the name of the system table that stores the ** list of users and their access credentials.

75824. ** Open an RBU handle. *** Argument zTarget is the path to the target database. Argument zRbu is ** the path to the RBU database. Each call to this function must be matched ** by a call to sqlite3rbu_close(). When opening the databases, RBU passes ** the SQLITE_CONFIG_URI flag to sqlite3_open_v2(). So if either zTarget ** or zRbu begin with "file:", it will be interpreted as an SQLite ** database URI, not a regular file name. *** If the zState argument is passed a NULL value, the RBU extension stores ** the current state of the update (how many rows have been updated, which ** indexes are yet to be updated etc.) within the RBU database itself. This ** can be convenient, as it means that the RBU application does not need to ** organize removing a separate state file after the update is concluded. ** Or, if zState is non-NULL, it must be a path to a database file in which ** the RBU extension can store the state of the update. *** When resuming an RBU update, the zState argument must be passed the same ** value as when the RBU update was started. *** Once the RBU update is finished, the RBU extension does not ** automatically remove any zState database file, even if it created it. *** By default, RBU uses the default VFS to access the files on disk. To ** use a VFS other than the default, an SQLite "file:" URI containing a ** "vfs=..." option may be passed as the zTarget option. *** IMPORTANT NOTE FOR ZIPVFS USERS: The RBU extension works with all of ** SQLite's built-in VFSs, including the multiplexor VFS. However it does ** not work out of the box with zipvfs. Refer to the comment describing ** the zipvfs_create_vfs() API below for details on using RBU with zipvfs.

75825. ** The second argument is a Trigger structure allocated by the ** fkActionTrigger() routine. This function deletes the Trigger structure ** and all of its sub-components. *** The Trigger structure or any of its sub-components may be allocated from ** the lookaside buffer belonging to database handle dbMem.

75826. Current node image to append to

75827. True if next rowid is first in page

75828. Update the db file change counter via the direct-write method. The ** following call will modify the in-memory representation of page 1 ** to include the updated change counter and then write page 1 ** directly to the database file. Because of the atomic-write ** property of the host file-system, this is safe.

75829. OUT: True if there are NULL values in PK

75830. FTS5 backend object

75831. Return SQLITE_ROW

75832. same as TK_STRING, out2

75833. Previous id read from aDoclist

75834. RAISE

75835. The page of the frame to be written

75836. Used to iterate through phrase tokens

75837. Name of database file for pPager

75838. The VDBE under construction

75839. The row that the VUpdate opcode will delete: none

75840. ** Define the parameters of the hash tables in the wal-index file. There ** is a hash-table following every HASHTABLE_NPAGE page numbers in the ** wal-index. *** Changing any of these constants will alter the wal-index format and ** create incompatibilities.

75841. Append a single character

75842. Fts5Storage.aTotalSize[]

75843. Start the view context

75844. (315) foreach_clause ::=

75845. True if RHS must be unique

75846. For STRING expressions, the near cluster

75847. ***** End of fault.c *****

75848. Value to return via *paiCol

75849. If SQLITE_FTS5_ENABLE_TEST_MI is defined, assume that the file ** fts5_test_mi.c is compiled and linked into the executable. And call ** its entry point to enable the matchinfo() demo.

75850. zName

75851. Check for a "special" INSERT operation. One of the form: *** INSERT INTO xyz(xyz) VALUES('command');

75852. UPDATE + DELETE

75853. Cannot be both MEM_Null and some other type

75854. No rowid. PRIMARY KEY is the key

75855. in3

75856. ** If the TRANSLATE_TRACE macro is defined, the value of each Mem is ** printed on stderr on the way into and out of sqlite3VdbeMemTranslate().

75857. IGNORE

75858. ** This function is invoked by the vdbe to call the xDestroy method ** of the virtual table named zTab in database iDb. This occurs ** when a DROP TABLE is mentioned. *** This call is a no-op if zTab is not a virtual table.

75859. 0x01

75860. Parent frame

75861. Used to iterate through aHash[]

75862. STAT3 and STAT4 have a parameter on this routine.

75863. CONSTRAINT

75864. ** Argument pNode is an FTS5_AND node.

75865. assert(db->pPreUpdate->pUnpacked);

75866. Address of OP_FkIfZero

75867. ** Recompute pMerger->aTree[iOut] by comparing the next keys on the ** two PmaReaders that feed that entry. Neither of the PmaReaders ** are advanced. This routine merely does the comparison.

75868. xShutdown

75869. 3-byte signed integer

75870. This is the sub-WHERE clause body. First skip over ** duplicate rows from prior sub-WHERE clauses, and record the ** rowid (or PRIMARY KEY) for the current row so that the same ** row will be skipped in subsequent sub-WHERE clauses.

75871. Create an implicit NOT operator.

75872. ** json_array_length(JSON) ** json_array_length(JSON, PATH) ** ** Return the number of elements in the top-level JSON array. ** Return 0 if the input is not a well-formed JSON array.

75873. Two copies of the header content

75874. Program counter

75875. Allocate the Fts5Bm25Data object

75876. If there is no more data to be read from the buffer, read the next ** p->nBuffer bytes of data from the file into it. Or, if there are less ** than p->nBuffer bytes remaining in the PMA, read all remaining data.

75877. ** Rtree virtual table module xFilter method.

75878. Visibility of parent node

75879. ** Read an entry from the pointer map. *** This routine retrieves the pointer map entry for page 'key', writing ** the type and parent page number to *pEType and *pPgno respectively. ** An error code is returned if something goes wrong, otherwise SQLITE_OK.

75880. Remove page from cache

75881. Cursor number of the subquery

75882. SELECT may not have a LIMIT clause

75883. Beginning here are the reduction cases. A typical example ** follows: ** case 0: ** #line <lineno> <grammarfile> ** { ... } // User supplied code ** #line <lineno> <thisfile> ** break;

75884. ** Attempt to seek the file-descriptor passed as the first argument to ** absolute offset iOff, then attempt to write nBuf bytes of data from ** pBuf to it. If an error occurs, return -1 and set *piErrno. Otherwise, ** return the actual number of bytes written (which may be less than ** nBuf).

75885. End mark for a varint

75886. If RESTRICT, "SELECT RAISE(...)"

75887. Open the index sub-system

75888. ** Convert a scalar expression node to a TK_REGISTER referencing ** register iReg. The caller must ensure that iReg already contains ** the correct value for the expression.

75889. Uncommitted Hash table changes

75890. ** An instance of the following structure holds information about SQL ** functions arguments that are the parameters to the printf() function.

75891. Set the P5 operand of the OP_Program instruction to non-zero if ** recursive invocation of this trigger program is disallowed. Recursive ** invocation is disallowed if (a) the sub-program is really a trigger, ** not a foreign key action, and (b) the flag to enable recursive triggers ** is clear.

75892. Subtask context (for pKeyInfo)

75893. The pBt is no longer on the sharing list, so we can access ** it without having to hold the mutex. ** ** Clean out and delete the BtShared object.

75894. OUT: SQLite db handle

75895. Opcode: AutoCommit P1 P2 * * * ** Set the database auto-commit flag to P1 (1 or 0). If P2 is true, roll ** back any currently active btree transactions. If there are any active ** VMs (apart from this one), then a ROLLBACK fails. A COMMIT fails if ** there are active writing VMs or active VMs that use shared cache. ** ** This instruction causes the VM to halt.

75896. Has no NOT INDEXED clause

75897. **comment:** Convert the data in the temporary table into whatever form ** it is that we currently need.
label: code-design

75898. **comment:** Zero or more PragFlg_XXX values
label: code-design

75899. Number of entries in aExt[]

75900. Vdbe_DISPLAY_P4 && defined(SQLITE_ENABLE_CURSOR_HINTS)

75901. ccons ::= DEFAULT ID|INDEXED

75902. Set pFile->locktype to this value before exiting

75903. The desired encoding for the collating sequence

75904. For operators other than UNION ALL we have to make sure that ** the ORDER BY clause covers every term of the result set. Add ** terms to the ORDER BY clause as necessary.

75905. **comment:** Remove the xxx_parent entry.
label: code-design

75906. ** Form A: ** CASE x WHEN e1 THEN r1 WHEN e2 THEN r2 ... WHEN eN THEN rN ELSE y END ** ** Form B: ** CASE WHEN e1 THEN r1 WHEN e2 THEN r2 ... WHEN eN THEN rN ELSE y END ** ** Form A is can be transformed into the equivalent form B as follows: ** CASE WHEN x=e1 THEN r1 WHEN x=e2 THEN r2 ... ** WHEN x=eN THEN rN ELSE y END ** ** X (if it exists) is in pExpr->pLeft. ** Y is in the last element of pExpr->x.pList if pExpr->x.pList->nExpr is ** odd. The Y is also optional. If the number of elements in x.pList ** is even, then Y is omitted and the "otherwise" result is NULL. ** Ei is in pExpr->pList->a[i*2] and Ri is pExpr->pList->a[i*2+1]. ** ** The result of the expression is the Ri for the first matching Ei, ** or if there is no matching Ei, the ELSE term Y, or if there is ** no ELSE term, NULL.

75907. ** Write mapping (iNode->iPar) to the <rtree>_parent table.

75908. Affinity of the LHS of the IN

75909. Last page in WAL for this reader

75910. Number of docs in database

75911. ** An instance of the following type is used to iterate through the contents ** of a doclist-index record. ** ** pData: ** Record containing the doclist-index data. ** ** bEof: ** Set to true once iterator has reached EOF. ** ** iOff: ** Set to the current offset within record pData.

75912. Assert that the p1 parameter is valid. Also that if there is no open ** transaction, then there cannot be any savepoints.

75913. the WHERE clause

75914. ** Allocate and return a new expression object. If anything goes wrong (i.e. ** OOM error), leave an error code in pParse and return NULL.

75915. Checkpoint and close the log. Because an EXCLUSIVE lock is held on ** the database file, the log and log-summary files will be deleted.

75916. The table being read

75917. **comment:** The FROM clause btree term to add
label: code-design

75918. Offsets into aPoslist for current row

75919. pPager->xBusyHandler = 0;

75920. ** Append the hash of the 64-bit integer passed as the second argument to the ** hash-key value passed as the first. Return the new hash-key value.

75921. PRAGMA synchronous=OFF

75922. EVIDENCE-OF: R-61304-29449 The unlikely(X) function is ** equivalent to likelihood(X, 0.0625). ** EVIDENCE-OF: R-01283-11636 The unlikely(X) function is ** short-hand for likelihood(X,0.0625). ** EVIDENCE-OF: R-36850-34127 The likely(X) function is short-hand ** for likelihood(X,0.9375). ** EVIDENCE-OF: R-53436-40973 The likely(X) function is equivalent ** to likelihood(X,0.9375).

75923. Sizes of each column, in tokens

75924. Function used to reclaim memory

75925. ***** Begin file update.c *****

75926. Populate this structure before returning.

75927. The return address register

75928. If we are holding a PENDING lock that ought to be released, then ** release it now.

75929. ** Change the limit on the amount of the database file that may be ** memory mapped.

75930. If pTemplate is always better than p, then cause p to be overwritten ** with pTemplate. pTemplate is better than p if: ** (1) pTemplate has no more dependences than p, and ** (2) pTemplate has an equal or lower cost than p.

75931. Use ExprList.u.x.iOrderByCol

75932. All table that pExpr might refer to

75933. ColNames:

75934. ** 2015 May 08 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This is an SQLite virtual table module implementing direct access to an ** existing FTS5 index. The module may create several different types of ** tables: ** ** col: ** CREATE TABLE vocab(term, col, doc, cnt, PRIMARY KEY(term, col)); ** ** One row for each term/column combination. The value of \$doc is set to ** the number of fts5 rows that contain at least one instance of term ** \$term within column \$col. Field \$cnt is set to the total number of ** instances of term \$term in column \$col (in any row of the fts5 table). ** ** row: ** CREATE TABLE vocab(term, doc, cnt, PRIMARY KEY(term)); ** ** One row for each term in the database. The value of \$doc is set to ** the number of fts5 rows that contain at least one instance of term ** \$term. Field \$cnt is set to the total number of instances of term ** \$term in the database.

75935. ** CAPI3REF: Constants Passed To The Conflict Handler ** ** Values that may be passed as the second argument to a conflict-handler. ** ** <d> ** <d>SQLITE_CHANGESET_DATA<dd> ** The conflict handler is invoked with CHANGESET_DATA as the second argument ** when processing a DELETE or UPDATE change if a row with the required ** PRIMARY KEY fields is present in the database, but one or more other ** (non primary-key) fields modified by the update do not contain the ** expected "before" values. ** ** The conflicting row, in this case, is the database row with the matching ** primary key. ** ** <d>SQLITE_CHANGESET_NOTFOUND<dd> ** The conflict handler is invoked with CHANGESET_NOTFOUND as the second ** argument when processing a DELETE or UPDATE change if a row with the ** required PRIMARY KEY fields is not present in the database. ** ** There is no conflicting row in this case. The results of invoking the ** sqlite3changeset_conflict() API are undefined. ** ** <d>SQLITE_CHANGESET_CONFLICT<dd> ** CHANGESET_CONFLICT is passed as the second argument to the conflict ** handler while processing an INSERT change if the operation would result ** in duplicate primary key values. ** ** The conflicting row in this case is the database row with the matching ** primary key. ** ** <d>SQLITE_CHANGESET_FOREIGN_KEY<dd> ** If foreign key handling is enabled, and applying a changeset leaves the ** database in a state containing foreign key violations, the conflict ** handler is invoked with CHANGESET_FOREIGN_KEY as the second argument ** exactly once before the changeset is

committed. If the conflict handler ** returns CHANGESET OMIT, the changes, including those that caused the ** foreign key constraint violation, are committed. Or, if it returns ** CHANGESET_ABORT, the changeset is rolled back. *** No current or conflicting row information is provided. The only function ** it is possible to call on the supplied sqlite3_changeset_iter handle ** is sqlite3changeset_fk_conflicts(). ***
<dt>SQLITE_CHANGESET_CONSTRAINT<dd> ** If any other constraint violation occurs while applying a change (i.e. ** a UNIQUE, CHECK or NOT NULL constraint), the conflict handler is ** invoked with CHANGESET_CONSTRAINT as the second argument. *** There is no conflicting row in this case. The results of invoking the ** sqlite3changeset_conflict() API are undefined. *** </dl>

75936. **comment:** Initialize the output variables to -1 in case an error occurs.

label: code-design

75937. Length of string argv[1]

75938. printf("SQL: [%s]\n", zSql); fflush(stdout);

75939. OUT: Jump to this label to skip partial index

75940. Compare size with this KeyInfo

75941. Write the term for this entry to disk.

75942. Set if global data is loaded

75943. Neither sqlite3changeset_old or sqlite3changeset_new can fail if the ** argument iterator points to a suitable entry. Make sure that xValue ** is one of these to guarantee that it is safe to ignore the return ** in the code below.

75944. ** Combine two changesets together.

75945. OUT: Total number of OR nodes in expr.

75946. Object partially overlaps query region

75947. ** RECORD FORMAT: *** The following record format is similar to (but not compatible with) that ** used in SQLite database files. This format is used as part of the ** change-set binary format, and so must be architecture independent. *** Unlike the SQLite database record format, each field is self-contained - ** there is no separation of header and data. Each field begins with a ** single byte describing its type, as follows: *** 0x00: Undefined value. ** 0x01: Integer value. ** 0x02: Real value. ** 0x03: Text value. ** 0x04: Blob value. ** 0x05: SQL NULL value. *** Note that the above match the definitions of SQLITE_INTEGER, SQLITE_TEXT ** and so on in sqlite3.h. For undefined and NULL values, the field consists ** only of the single type byte. For other types of values, the type byte ** is followed by: *** Text values: ** A varint containing the number of bytes in the value (encoded using ** UTF-8). Followed by a buffer containing the UTF-8 representation ** of the text value. There is no nul terminator. *** Blob values: ** A varint containing the number of bytes in the value, followed by ** a buffer containing the value itself. *** Integer values: ** An 8-byte big-endian integer value. *** Real values: ** An 8-byte big-endian IEEE 754-2008 real value. *** Varint values are encoded in the same way as varints in the SQLite ** record format. *** CHANGESET FORMAT: *** A changeset is a collection of DELETE, UPDATE and INSERT operations on ** one or more tables. Operations on a single table are grouped together, ** but may occur in any order (i.e. deletes, updates and inserts are all ** mixed together). *** Each group of changes begins with a table header: *** 1 byte: Constant 0x54 (capital 'T') ** Varint: Number of columns in the table. ** nCol bytes: 0x01 for PK columns, 0x00 otherwise. ** N bytes: Unqualified table name (encoded using UTF-8). Null-terminated. *** Followed by one or more changes to the table. *** 1 byte: Either SQLITE_INSERT (0x12), UPDATE (0x17) or DELETE (0x09). ** 1 byte: The "indirect-change" flag. ** old.* record: (delete and update only) ** new.* record: (insert and update only) *** The "old." and "new." records, if present, are N field records in the ** format described above under "RECORD FORMAT", where N is the number of ** columns in the table. The i'th field of each record is associated with ** the i'th column of the table, counting from left to right in the order ** in which columns were declared in the CREATE TABLE statement. ** The new.* record that is part of each INSERT change contains the values ** that make up the new row. Similarly, the old.* record that is part of each ** DELETE change contains the values that made up the row that was deleted ** from the database. In the changeset format, the records that are part ** of INSERT or DELETE changes never contain any undefined (type byte 0x00) ** fields. *** Within the old.* record associated with an UPDATE change, all fields ** associated with table columns that are not PRIMARY KEY columns and are ** not modified by the UPDATE change are set to "undefined". Other fields ** are set to the values that made up the row before the UPDATE that the ** change records took place. Within the new.* record, fields associated ** with table columns modified by the UPDATE change contain the new ** values. Fields associated with table columns that are not modified ** are set to "undefined". *** PATCHSET FORMAT: *** A patchset is also a collection of changes. It is similar to a changeset, ** but leaves undefined those fields that are not useful if no conflict ** resolution is required when applying the changeset. *** Each group of changes begins with a table header: *** 1 byte: Constant 0x50 (capital 'P') ** Varint: Number of columns in the table. ** nCol bytes: 0x01 for PK columns, 0x00 otherwise. ** N bytes: Unqualified table name (encoded using UTF-8). Null-terminated. *** Followed by one or more changes to the table. *** 1 byte: Either SQLITE_INSERT (0x12), UPDATE (0x17) or DELETE (0x09). ** 1 byte: The "indirect-change" flag. ** single record: (PK fields for DELETE, PK and modified fields for UPDATE, ** full record for INSERT). *** As in the changeset format, each field of the single record that is part ** of a patchset change is associated with the correspondingly positioned ** table column, counting from left to right within the CREATE TABLE ** statement. *** For a DELETE change, all fields within the record except those associated ** with PRIMARY KEY columns are set to "undefined". The PRIMARY KEY fields ** contain the values identifying the row to delete. *** For an UPDATE change, all fields except those associated with PRIMARY KEY ** columns and columns that are modified by the UPDATE are set to "undefined". ** PRIMARY KEY fields contain the values identifying the table row to update, ** and fields associated with modified columns contain the new column values. *** The records associated with INSERT changes are in the same format as for ** changesets. It is not possible for a record associated with an INSERT ** change to contain a field set to "undefined".

75948. **comment:** ** Implementation of the sqlite3_pcache.xFetch method. *** Fetch a page by key value. ** ** Whether or not a new page may be allocated by this function depends on ** the value of the createFlag argument. 0 means do not allocate a new ** page. 1 means allocate a new page if space is easily available. 2 ** means to try really hard to allocate a new page. *** For a non-purgeable cache (a cache used as the storage for an in-memory ** database) there is really no difference between createFlag 1 and 2. So ** the calling function (pcache.c) will never have a createFlag of 1 on ** a non-purgeable cache. *** There are three different approaches to obtaining space for a page, ** depending on the value of parameter createFlag (which may be 0, 1 or 2). *** 1. Regardless of the value of createFlag, the cache is searched for a ** copy of the requested page. If one is found, it is returned. *** 2. If createFlag==0 and the page is not already in the cache, NULL is ** returned. *** 3. If createFlag is 1, and the page is not already in the cache, then ** return NULL (do not allocate a new page) if any of the following ** conditions are true: *** (a) the number of pages pinned by the cache is greater than ** PCache1.nMax, or *** (b) the number of pages pinned by the cache is greater than ** the sum of nMax for all purgeable caches, less the sum of ** nMin for all other purgeable caches, or *** 4. If none of the first three conditions apply and the cache is marked ** as purgeable, and if one of the following is true: *** (a) The number of pages allocated for the cache is already ** PCache1.nMax, or *** (b) The number of pages allocated for all purgeable caches is ** already equal to or greater than the sum of nMax for all ** purgeable caches, *** (c) The system is under memory pressure and wants to avoid ** unnecessary pages cache entry allocations *** then attempt to recycle a page from the LRU list. If it is the right ** size, return the recycled buffer. Otherwise, free the buffer and ** proceed to step 5. *** 5. Otherwise, allocate and return a new page buffer. *** There are two versions of this routine. pcache1FetchWithMutex() is ** the general case. pcache1FetchNoMutex() is a faster implementation for ** the common case where pGroup->mutex is NULL. The pcache1Fetch() wrapper ** invokes the appropriate routine.

label: code-design

75949. Register storing resulting

75950. 21

75951. Generate code to this VDBE

75952. Search for this index expression

75953. transtype ::= DEFERRED

75954. ** Called during virtual module initialization to register UDF ** fts5_decode() with SQLite

75955. ** Use standard C library malloc and free on non-Apple systems. ** Also used by Apple systems if SQLITE_WITHOUT_ZONEMALLOC is defined.

75956. x>EXPR or x>=EXPR constraint

75957. xCurrentTimeInt64 (version 2)

75958. Allocate the sqlite data structure

75959. Number of bytes to allocate

75960. Size of doclist

75961. Original value of pNew->wsFlags

75962. **comment:** Update the Fts5PageWriter.term field.

label: code-design

75963. The cursor to be moved

75964. Encoding did not change

75965. Size of the journal file in bytes

75966. !defined(_OS_COMMON_H_)

75967. ** INSERT OR REPLACE a record into the %_data table.
75968. WHERE rowid ..
75969. File descriptor for sub-journal
75970. The iterator to advance by one position
75971. For use by extension VFS
75972. In an UPDATE operation, if this index is the PRIMARY KEY index ** of a WITHOUT ROWID table and there has been no change the ** primary key, then no collision is possible. The collision detection ** logic below can all be skipped.
75973. synopsis: if fkctr[P1]==0 goto P2
75974. **comment:** NB: The sqlite3OpcodeName() function is implemented by code created ** by the mkopcodeh.awk and mkopcodec.awk scripts which extract the ** information from the vdbe.c source text
label: code-design
75975. PRAGMA fullfsync=ON
75976. RO incr-blob open on %_data table
75977. Write cell size here
75978. **** This function appends an update change to the buffer (see the comments ** under "CHANGESET FORMAT" at the top of the file). An update change ** consists of: ** ** 1 byte: SQLITE_UPDATE (0x17) ** n bytes: old.* record (see RECORD FORMAT) ** m bytes: new.* record (see RECORD FORMAT) ** ** The SessionChange object passed as the third argument contains the ** values that were stored in the row when the session began (the old.* ** values). The statement handle passed as the second argument points ** at the current version of the row (the new.* values). ** ** If all of the old.* values are equal to their corresponding new.* value ** (i.e. nothing has changed), then no data at all is appended to the buffer. ** ** Otherwise, the old.* record contains all primary key values and the ** original values of any fields that have been modified. The new.* record ** contains the new values of only those fields that have been modified.
75979. ** Allocate k new pages. Reuse old pages where possible.
75980. ***** End of analyze.c *****
75981. True if second change is indirect
75982. fall thru
75983. ** Return the maximum amount of memory that has ever been ** checked out since either the beginning of this process ** or since the most recent reset.
75984. First token in coalesced phrase instance
75985. Set nDestTruncate to the final number of pages in the destination ** database. The complication here is that the destination page ** size may be different to the source page size. ** ** If the source page size is smaller than the destination page size, ** round up. In this case the call to sqlite3OsTruncate() below will ** fix the size of the file. However it is important to call ** sqlite3PagerTruncateImage() here so that any pages in the ** destination file that lie beyond the nDestTruncate page mark are ** journalled by PagerCommitPhaseOne() before they are destroyed ** by the file truncation.
75986. Position of last value written
75987. Byte offset of end of current token
75988. Like SRT_Fifo, but unique results only
75989. Opcode: DecrJumpZero P1 P2 * * * * * Synopsis: if (--r[P1]==0 goto P2) ** Register P1 must hold an integer. Decrement the value in P1 ** and jump to P2 if the new value is exactly zero.
75990. IMPLEMENTATION-OF: R-57228-12904 Invoking sqlite3_finalize() on a NULL ** pointer is a harmless no-op.
75991. ** Allocate a new segment-id for the structure pStruct. The new segment ** id must be between 1 and 65335 inclusive, and must not be used by ** any currently existing segment. If a free segment id cannot be found, ** SQLITE_FULL is returned. ** ** If an error has already occurred, this function is a no-op. 0 is ** returned in this case.
75992. Fall through into OP_NeedsTable
75993. term ::= STRING
75994. ** Additional bit values that can be ORed with an affinity without ** changing the affinity. ** ** The SQLITE_NOTNULL flag is a combination of NULLEQ and JUMPIFNULL. ** It causes an assert() to fire if either operand to a comparison ** operator is NULL. It is added to certain comparison operators to ** prove that the operands are always NOT NULL.
75995. DB Page containing root of this index
75996. True if this is the first term written
75997. Index in pSrc->a[] of first new slot
75998. (299) tconcomma ::=
75999. The Windows errno from the last I/O error
76000. The token that holds the name of the view
76001. Construct an expression node for a unary postfix operator
76002. ** One of the cells in node pNode is guaranteed to have a 64-bit ** integer value equal to iRowid. Return the index of this cell.
76003. Multi-segment-reader handle
76004. ** Invoke SQLITE_FCNTL_MMAP_SIZE based on the current value of szMmap.
76005. ** Try to provide a memory barrier operation, needed for initialization ** and also for the xShmBarrier method of the VFS in cases when SQLite is ** compiled without mutexes (SQLITE_THREADS=0).
76006. Integer value of left operand
76007. Previous in LRU list of unpinned pages
76008. Free allocated structure
76009. ** Set the Pager.xGet method for the appropriate routine used to fetch ** content from the pager.
76010. See the tool/mkopcodeh.tcl script for details
76011. Number of bytes to read
76012. Number of child nodes
76013. Write the record
76014. **comment:** Sort entries into the forest on the first test of a new batch. ** To save unnecessary work, only do this when the batch number changes.
label: code-design
76015. Deprecated
76016. COLLATE
76017. IN/OUT: The WHERE clause to add to
76018. pArg
76019. SQLITE_UPDATE, DELETE or INSERT
76020. Pointer to allocation
76021. ***** Begin file date.c *****
76022. Number of memory cells used so far
76023. CREATE => nothing
76024. Select a sub-task to sort and flush the current list of in-memory ** records to disk. If the sorter is running in multi-threaded mode, ** round-robin between the first (pSorter->nTask-1) tasks. Except, if ** the background thread from a sub-tasks previous turn is still running, ** skip it. If the first (pSorter->nTask-1) sub-tasks are all still busy, ** fall back to using the final sub-task. The first (pSorter->nTask-1) ** sub-tasks are preferred as they use background threads - the final ** sub-task uses the main thread.
76025. pPager->pFirst = 0;
76026. Shorthand for pTerm->eOperator
76027. restore the original locking context and pMethod then close it
76028. SQLITE_OS_UNIX && defined(SQLITE_MUTEX_PTHREADS)
76029. ** Iterator pIter currently points to the first rowid in a doclist. This ** function sets the iterator up so that iterates in reverse order through ** the doclist.
76030. neardist_opt ::= COMMA STRING
76031. ** Return true if a memory cell is not marked as invalid. This macro ** is for use inside assert() statements only.
76032. This is one term of an OR-optimization using the PRIMARY KEY of a ** WITHOUT ROWID table. No need for a separate index
76033. The "B" part of a "A AS B" phrase. zName is the "A"

76034. The new freeblock is at the beginning of the cell content area, *** so just extend the cell content area rather than create another *** freelist entry
76035. ** An SQL parser context. A copy of this structure is passed through ** the parser and down into all the parser action routine in order to ** carry around information that is global to the entire parse. *** The structure is divided into two parts. When the parser and code ** generate call themselves recursively, the first part of the structure ** is constant but the second part is reset at the beginning and end of ** each recursion. *** The nTableLock and aTableLock variables are only used if the shared-cache ** feature is enabled (if sqlite3Tsd()->useSharedData is true). They are ** used to store the set of table-locks required by the statement being ** compiled. Function sqlite3TableLock() is used to add entries to the ** list.
76036. ** Open an in-memory journal file.
76037. On OS X on an msdos filesystem, the inode number is reported ** incorrectly for zero-size files. See ticket #3260. To work ** around this problem (we consider it a bug in OS X, not SQLite) ** we always increase the file size to 1 by writing a single byte ** prior to accessing the inode number. The one byte written is ** an ASCII 'S' character which also happens to be the first byte ** in the header of every SQLite database. In this way, if there ** is a race condition such that another thread has already populated ** the first page of the database, no damage is done.
76038. ** Decrement the BtShared.nRef counter. When it reaches zero, ** remove the BtShared structure from the sharing list. Return ** true if the BtShared.nRef counter reaches zero and return ** false if it is still positive.
76039. True to enable triggers
76040. ** This routine sets the value to be returned by subsequent calls to ** sqlite3_changes() on the database handle 'db'.
76041. Acts as query when has one argument
76042. ** This function merges two sorted lists into a single sorted list. *** aLeft[] and aRight[] are arrays of indices. The sort key is ** aContent[aLeft[]] and aContent[aRight[]]. Upon entry, the following ** is guaranteed for all J < K: *** aContent[aLeft[J]] < aContent[aLeft[K]] ** aContent[aRight[J]] < aContent[aRight[K]] *** This routine overwrites aRight[] with a new (probably longer) sequence ** of indices such that the aRight[] contains every index that appears in ** either aLeft[] or the old aRight[] and such that the second condition ** above is still met. *** The aContent[aLeft[X]] values will be unique for all X. And the ** aContent[aRight[X]] values will be unique too. But there might be ** one or more combinations of X and Y such that ** aLeft[X] != aRight[Y] && aContent[aLeft[X]] == aContent[aRight[Y]] *** When that happens, omit the aLeft[X] and use the aRight[Y] index.
76043. Switch on the character-class of the first byte ** of the token. See the comment on the CC_ defines ** above.
76044. File descriptor to write to
76045. ** This is called during initialization to register the fts5_expr() scalar ** UDF with the SQLite handle passed as the only argument.
76046. Number of active VDBEs that read and write
76047. ** Destroy the part of winFile that deals with wince locks
76048. Number of active OP_VDestroy operations
76049. try to create all the intermediate directories
76050. OE_Abort, OE_Ignore, OE_Replace, or OE_None
76051. ** Delete all cached deferred doclists. Deferred doclists are cached ** (allocated) by the sqlite3Fts3CacheDeferredDoclists() function.
76052. True if is an aggregate function
76053. Allocate new SessionTable object.
76054. Error and awaiting close
76055. Original value of pNew->u.btree.nTop
76056. FTS5_MAX_SEGMENT is currently defined as 2000. So the following ** array is 63 elements, or 252 bytes, in size.
76057. Join all threads
76058. This is really OP_Noop and OP_Explain
76059. If recursive mutexes are not available, then we have to grow ** our own. This implementation assumes that pthread_equal() ** is atomic - that it cannot be deceived into thinking self ** and p->owner are equal if p->owner changes between two values ** that are not equal to self while the comparison is taking place. *** This implementation also assumes a coherent cache - that ** separate processes cannot read different values from the same ** address at the same time. If either of these two conditions ** are not met, then the mutexes will fail and problems will result.
76060. OUT: True to retry.
76061. Get an EXCLUSIVE lock on the database file. At this point it is ** important that a RESERVED lock is not obtained on the way to the ** EXCLUSIVE lock. If it were, another process might open the ** database file, detect the RESERVED lock, and conclude that the ** database is safe to read while this process is still rolling the ** hot-journal back. *** Because the intermediate RESERVED lock is not requested, any ** other process attempting to access the database file will get to ** this point in the code and fail to obtain its own EXCLUSIVE lock ** on the database file. *** Unless the pager is in locking_mode=exclusive mode, the lock is ** downgraded to SHARED_LOCK before this function returns.
76062. End switch over the format type
76063. Assume an indexed expression can always yield a NULL
76064. ** This function retrieves the doclist for the specified term (or term ** prefix) from the database.
76065. Advance iterator ctx.iter so that it points to the first coalesced ** phrase instance at or following position iBestStart.
76066. Name of rank and rowid columns
76067. ** Insert or replace a WhereLoop entry using the template supplied. *** An existing WhereLoop entry might be overwritten if the new template ** is better and has fewer dependencies. Or the template will be ignored ** and no insert will occur if an existing WhereLoop is faster and has ** fewer dependencies than the template. Otherwise a new WhereLoop is ** added based on the template. *** If pBuilder->pOrSet is not NULL then we care about only the ** prerequisites and rRun and nOut costs of the N best loops. That ** information is gathered in the pBuilder->pOrSet object. This special ** processing mode is used only for OR clause processing. *** When accumulating multiple loops (when pBuilder->pOrSet is NULL) we ** still might overwrite similar loops with the new template if the ** new template is better. Loops may be overwritten if the following ** conditions are met: *** (1) They have the same iTab. ** (2) They have the same iSortIdx. ** (3) The template has same or fewer dependencies than the current loop ** (4) The template has the same or lower cost than the current loop
76068. ** Return True if it is possible that pIndex might be useful in ** implementing the ORDER BY clause in pBuilder. *** Return False if pBuilder does not contain an ORDER BY clause or ** if there is no way for pIndex to be useful in implementing that ** ORDER BY clause.
76069. rowid == ?
76070. Node number of parent node
76071. Array of hash slots
76072. Jump over the subroutines
76073. **comment:** Avoid memory leak when malloc fails
 label: code-design
76074. If after trying to extract new entries from the header, nHdrParsed is ** still not up to p2, that means that the record has fewer than p2 ** columns. So the result will be either the default value or a NULL.
76075. 980
76076. Opcode to add sorter record to sorter
76077. ***** Begin file trigger.c *****
76078. 1250
76079. ** Return the column declaration type (if applicable) of the i'th column ** of the result set of SQL statement pStmt.
76080. EVIDENCE-OF: R-61006-08918 If the memory pointer is not NULL then the ** alternative memory allocator is engaged to handle all of SQLites ** memory allocation needs.
76081. Set of table index masks
76082. The query is a simple term lookup that matches at most one term in ** the index. All that is required is a straight hash-lookup. *** Because the stack address of pE may be accessed via the aElem pointer ** below, the "Fts3HashElem *pE" must be declared so that it is valid ** within this entire function, not just this "else{...}" block.
76083. pIdx is not a UNIQUE index
76084. EVIDENCE-OF: R-53367-43190 If either argument to this option is ** negative, then that argument is changed to its compile-time default. *** EVIDENCE-OF: R-34993-45031 The maximum allowed mmap size will be ** silently truncated if necessary so that it does not exceed the ** compile-time maximum mmap size set by the SQLITE_MAX_MMAP_SIZE ** compile-time option.
76085. Length of database filename - dbPath
76086. ** Database Format of R-Tree Tables ** ----- ** The data structure for a single virtual r-tree table is stored in three ** native SQLite tables declared as follows. In each case, the '%' character ** in the table name is replaced with the user-supplied name of the r-tree ** table. *** CREATE

TABLE %_node(nodeno INTEGER PRIMARY KEY, data BLOB) ** CREATE TABLE %_parent(nodeno INTEGER PRIMARY KEY, parentnode INTEGER) **
CREATE TABLE %_rowid(rowid INTEGER PRIMARY KEY, nodeno INTEGER) *** The data for each node of the r-tree structure is stored in the %_node **
table. For each node that is not the root node of the r-tree, there is ** an entry in the %_parent table associating the node with its parent. ** And for each row of
data in the table, there is an entry in the %_rowid ** table that maps from the entries rowid to the id of the node that it ** is stored on. *** The root node of an r-
tree always exists, even if the r-tree table is ** empty. The nodeno of the root node is always 1. All other nodes in the ** table must be the same size as the root
node. The content of each node ** is formatted as follows: *** 1. If the node is the root node (node 1), then the first 2 bytes ** of the node contain the tree depth
as a big-endian integer. ** For non-root nodes, the first 2 bytes are left unused. *** 2. The next 2 bytes contain the number of entries currently ** stored in the
node. *** 3. The remainder of the node contains the node entries. Each entry ** consists of a single 8-byte integer followed by an even number ** of 4-byte
coordinates. For leaf nodes the integer is the rowid ** of a record. For internal nodes it is the node number of a ** child page.

76087. **comment:** Execute assert() statements to ensure that the Vdbe.apCsr[] and ** Vdbe.aMem[] arrays have already been cleaned up.

label: code-design

76088. ** Return a checksum of all entries in the FTS index that correspond to ** language id iLangid. The checksum is calculated by XORing the checksums ** of each
individual entry (see fts3ChecksumEntry()) together. *** If successful, the checksum value is returned and *pRc set to SQLITE_OK. ** Otherwise, if an error
occurs, *pRc is set to an SQLite error code. The ** return value is undefined in this case.

76089. OUT: Results of parse

76090. WITH clause that pCte belongs to

76091. Possible values for RtreeConstraint.op

76092. Mark node i of pParse as being a child of iParent. Call recursively ** to fill in all the descendants of node i.

76093. Allocate the array of TermOffset iterators.

76094. CURRENT_TIME

76095. ** Allowed flags for the 3rd parameter to sqlite3FindInIndex().

76096. int *psz

76097. Releasing a pending lock

76098. Combination of the previous two

76099. ** Attempt to automatically detect the operating system and setup the ** necessary pre-processor macros for it.

76100. Prerequisites of the pExpr->pLeft

76101. ccons ::= DEFAULT term

76102. Index cursor

76103. **comment:** ** This function is called just before writing a set of frames to the log ** file (see sqlite3WalFrames()). It checks to see if, instead of appending ** to
the current log file, it is possible to overwrite the start of the ** existing log file with the new frames (i.e. "reset" the log). If so, ** it sets pWal->hdr.mxFrame to 0.
Otherwise, pWal->hdr.mxFrame is left ** unchanged. *** SQLITE_OK is returned if no error is encountered (regardless of whether ** or not pWal-
>hdr.mxFrame is modified). An SQLite error code is returned ** if an error occurs.

label: code-design

76104. Bytes to allocate for PCache

76105. ***** Include os_common.h in the middle of mutex_w32.c *****

76106. from ::= FROM seltablist

76107. IMP: R-26835-10964

76108. xOpen - open a cursor

76109. **comment:** ** This routine attempts to flatten subqueries as a performance optimization. ** This routine returns 1 if it makes changes and 0 if no flattening occurs.

*** To understand the concept of flattening, consider the following ** query: *** SELECT a FROM (SELECT x+y AS a FROM t1 WHERE z<100) WHERE
a>5 *** The default way of implementing this query is to execute the ** subquery first and store the results in a temporary table, then ** run the outer query on
that temporary table. This requires two ** passes over the data. Furthermore, because the temporary table ** has no indices, the WHERE clause on the outer query
cannot be ** optimized. *** This routine attempts to rewrite queries such as the above into ** a single flat select, like this: *** SELECT x+y AS a FROM t1
WHERE z<100 AND a>5 *** The code generated for this simplification gives the same result ** but only has to scan the data once. And because indices might
** exist on the table t1, a complete scan of the data might be ** avoided. *** Flattening is only attempted if all of the following are true: *** (1) The subquery
and the outer query do not both use aggregates. ** (2) The subquery is not an aggregate or (2a) the outer query is not a join ** and (2b) the outer query does not
use subqueries other than the one ** FROM-clause subquery that is a candidate for flattening. (2b is ** due to ticket [2f170d73bf9abf80] from 2015-02-09.) **
*** (3) The subquery is not the right operand of a LEFT JOIN ** or (a) the subquery is not itself a join and (b) the FROM clause ** of the subquery does not
contain a virtual table and (c) the ** outer query is not an aggregate. *** (4) The subquery is not DISTINCT. *** (**) At one point restrictions (4) and (5)
defined a subset of DISTINCT ** sub-queries that were excluded from this optimization. Restriction ** (4) has since been expanded to exclude all DISTINCT
subqueries. *** (6) The subquery does not use aggregates or the outer query is not ** DISTINCT. *** (7) The subquery has a FROM clause. TODO: For
subqueries without ** A FROM clause, consider adding a FROM clause with the special ** table sqlite_once that consists of a single row containing a ** single
NULL. *** (8) The subquery does not use LIMIT or the outer query is not a join. *** (9) The subquery does not use LIMIT or the outer query does not use **
aggregates. *** (**) Restriction (10) was removed from the code on 2005-02-05 but we ** accidentally carried the comment forward until 2014-09-15. Original
** text: "The subquery does not use aggregates or the outer query ** does not use LIMIT." *** (11) The subquery and the outer query do not both have ORDER
BY clauses. *** (**) Not implemented. Subsumed into restriction (3). Was previously ** a separate restriction deriving from ticket #350. *** (13) The
subquery and outer query do not both use LIMIT. *** (14) The subquery does not use OFFSET. *** (15) The outer query is not part of a compound select or
the ** subquery does not have a LIMIT clause. ** (See ticket #2339 and ticket [02a8e81d44].) *** (16) The outer query is not an aggregate or the subquery does
** not contain ORDER BY. (Ticket #2942) This used to not matter ** until we introduced the group_concat() function. *** (17) The sub-query is not a
compound select, or it is a UNION ALL ** compound clause made up entirely of non-aggregate queries, and ** the parent query: **** is not itself part of a
compound select, ** is not an aggregate or DISTINCT query, and ** is not a join ** ** The parent and sub-query may contain WHERE clauses. Subject to **
rules (11), (13) and (14), they may also contain ORDER BY, ** LIMIT and OFFSET clauses. The subquery cannot use any compound ** operator other than
UNION ALL because all the other compound ** operators have an implied DISTINCT which is disallowed by ** restriction (4). *** Also, each component of
the sub-query must return the same number ** of result columns. This is actually a requirement for any compound ** SELECT statement, but all the code here
does is make sure that no ** such (illegal) sub-query is flattened. The caller will detect the ** syntax error and return a detailed message. *** (18) If the sub-
query is a compound select, then all terms of the ** ORDER BY clause of the parent must be simple references to ** columns of the sub-query. *** (19) The
subquery does not use LIMIT or the outer query does not ** have a WHERE clause. *** (20) If the sub-query is a compound select, then it must not use ** an
ORDER BY clause. Ticket #3773. We could relax this constraint ** somewhat by saying that the terms of the ORDER BY clause must ** appear as unmodified
result columns in the outer query. But we ** have other optimizations in mind to deal with that case. *** (21) The subquery does not use LIMIT or the outer
query is not ** DISTINCT. (See ticket [752e1646fc].) *** (22) The subquery is not a recursive CTE. *** (23) The parent is not a recursive CTE, or the sub-
query is not a ** compound query. This restriction is because transforming the ** parent to a compound query confuses the code that handles ** recursive queries
in multiSelect(). *** (24) The subquery is not an aggregate that uses the built-in min() or ** or max() functions. (Without this restriction, a query like: **
"SELECT x FROM (SELECT max(y), x FROM t1)" would not necessarily ** return the value X for which Y was maximal.) *** In this routine, the "p"
parameter is a pointer to the outer query. ** The subquery is p->pSrc->a[ifFrom]. isAgg is true if the outer query ** uses aggregates and subqueryIsAgg is true if
the subquery uses aggregates. *** If flattening is not attempted, this routine is a no-op and returns 0. ** If flattening is attempted this routine returns 1. *** All
of the expression analysis must occur on both the outer query and ** the subquery before this routine runs.

label: code-design

76110. ** Implement a memory barrier or memory fence on shared memory. ** All loads and stores begun before the barrier must complete before ** any load or store
begun after the barrier.

76111. ** Return a pointer to the data for the specified page.

76112. ** The proxy locking method is a "super-method" in the sense that it ** opens secondary file descriptors for the conch and lock files and ** it uses proxy, dot-file,
AFP, and flock() locking methods on those ** secondary files. For this reason, the division that implements ** proxy locking is located much further down in the
file. But we need ** to go ahead and define the sqlite3_io_methods and finder function ** for proxy locking here. So we forward declare the I/O methods.

76113. Bytes of text in zName

76114. File descriptor open on db file

76115. If pLast is NULL at this point, then the last rowid for this doclist ** lies on the page currently indicated by the iterator. In this case ** pIter->iLeafOffset is already
set to point to the position-list size ** field associated with the first relevant rowid on the page. *** Or, if pLast is non-NULL, then it is the page that contains the
last ** rowid. In this case configure the iterator so that it points to the ** first rowid on this page.

76116. ** Return the number of connections to the BtShared object accessed by ** the Btree handle passed as the only argument. For private caches ** this is always 1.
For shared caches it may be 1 or greater.

76117. The BLOB encoding of the document size

76118. Next value in trailing position list

76119. Language id for tokenizer

76120. The "docid <= ?" constraint, if any

76121. If pToRelease is not zero than pPayload points into the data area ** of pToRelease. Make sure pToRelease is still writeable.

76122. ** Generate code that will do an analysis of a single table in ** a database. If pOnlyIdx is not NULL then it is a single index ** in pTab that should be analyzed.

76123. Opcode: NewRowid P1 P2 P3 * * *** Synopsis: r[P2]=rowid ** ** Get a new integer record number (a.k.a "rowid") used as the key to a table. ** The record number is not previously used as a key in the database ** table that cursor P1 points to. The new record number is written ** written to register P2. ** ** If P3>0 then P3 is a register in the root frame of this VDBE that holds ** the largest previously generated record number. No new record numbers are ** allowed to be less than this value. When this value reaches its maximum, ** an SQLITE_FULL error is generated. The P3 register is updated with the ** generated record number. This P3 mechanism is used to help implement the ** AUTOINCREMENT feature.

76124. Only the first key on the page may ==maxKey

76125. XISTSAVEPOINTERSECTTRIGGERREFERENCESCONSTRAINTOFFSETTEMPORARY

76126. Number of columns in the result

76127. Both sides of the comparison are columns. If one has numeric ** affinity, use that. Otherwise use no affinity.

76128. Number of entries in aEquiv[]

76129. "rank" function. Populated on demand from vtab.xColumn().

76130. ***** End of the posix advisory lock implementation *****

76131. A pure NULL might have other flags, such as MEM_Static, MEM_Dyn, ** MEM_Ephem, MEM_Cleared, or MEM_Subtype

76132. Array of mmap'd *-shm regions

76133. ** For the entry that cursor pCur is point to, return as ** many bytes of the key or data as are available on the local ** b-tree page. Write the number of available bytes into *pAmt. ** ** The pointer returned is ephemeral. The key/data may move ** or be destroyed on the next call to any Btree routine, ** including calls from other threads against the same cache. ** Hence, a mutex on the BtShared should be held prior to calling ** this routine. ** ** These routines is used to get quick access to key and data ** in the common case where no overflow pages are used.

76134. Query result set. Req'd for DISTINCT

76135. (314) plus_num ::= INTEGER|FLOAT

76136. ** zSql is a zero-terminated string of UTF-8 SQL text. Return the number of ** bytes in this text up to but excluding the first character in ** a host parameter. If the text contains no host parameters, return ** the total number of bytes in the text.

76137. The specific table containing the indexed database

76138. If not NULL, bind only if true

76139. ** Create a tokenizer cursor to tokenize an input buffer. The caller ** is responsible for ensuring that the input buffer remains valid ** until the cursor is closed (using the xClose() method).

76140. Estimated cost of using this index

76141. ** Masks used for mem5.aCtrl[] elements.

76142. The cursor on the LHS of the term

76143. ** Locate the table identified by *p. ** ** This is a wrapper around sqlite3LocateTable(). The difference between ** sqlite3LocateTable() and this function is that this function restricts ** the search to schema (p->pSchema) if it is not NULL. p->pSchema may be ** non-NULL if it is part of a view or trigger program definition. See ** sqlite3FixSrcList() for details.

76144. Zero tells OP_Found to use a composite key

76145. If either the iTab or iSortIdx values for two WhereLoop are different ** then those WhereLoops need to be considered separately. Neither is ** a candidate to replace the other.

76146. Size of header before each frame in wal

76147. !defined(SQLITE_VDBEINT_H)

76148. Database to be rekeyed

76149. Evaluate subqueries as coroutines

76150. Extra bytes at the end

76151. At this point the transaction is committed but the write lock ** is still held on the file. If there is a size limit configured for ** the persistent journal and the journal file currently consumes more ** space than that limit allows for, truncate it now. There is no need ** to sync the file following this operation.

76152. expr ::= CASE case_operand case_explist case_else END

76153. Value for "level" field (absolute level)

76154. Mask of usable tables

76155. True to allow FTS4-only syntax

76156. Generate code to handle the case of A>B

76157. Output buffer

76158. Absolute index of input segments

76159. ***** Interface to code in fts5_vocab.c.

76160. ** Make sure the BtCursor* given in the argument has a valid ** BtCursor.info structure. If it is not already valid, call ** btreeParseCell() to fill it in. ** ** BtCursor.info is a cache of the information in the current cell. ** Using this cache reduces the number of calls to btreeParseCell().

76161. ***** End of table.c *****

76162. **comment:** ***** The cases of the switch statement above this line should all be indented ** by 6 spaces. But the left-most 6 spaces have been removed to improve the ** readability. From this point on down, the normal indentation rules are ** restored. *****

76163. ** Generate the text of a WHERE expression which can be used to select all ** temporary triggers on table pTab from the sqlite_temp_master table. If ** table pTab has no temporary triggers, or is itself stored in the ** temporary database, NULL is returned.

76164. EVIDENCE-OF: R-59310-51205 The "reserved space" size in the 1-byte ** integer at offset 20 is the number of bytes of space at the end of ** each page to reserve for extensions. ** ** EVIDENCE-OF: R-37497-42412 The size of the reserved region is ** determined by the one-byte unsigned integer found at an offset of 20 ** into the database file header.

76165. ***** End of fts5.c *****

76166. The original expression

76167. Range of content

76168. Set pStmt to the compiled version of: ** ** SELECT max(level) FROM %Q.%q_segdir' WHERE level BETWEEN ? AND ? ** ** (1024 is actually the value of macro FTS3_SEGDIR_PREFIXLEVEL_STR).

76169. Blocks allocated for leaf nodes

76170. Bytes of zTitle to save. Includes '\0' and padding

76171. ** CAPI3REF: Database Connection Status ** METHOD: sqlite3 *** ^This interface is used to retrieve runtime status information ** about a single [database connection]. ^The first argument is the ** database connection object to be interrogated. ^The second argument ** is an integer constant, taken from the set of ** [SQLITE_DBSTATUS options], that ** determines the parameter to interrogate. The set of ** [SQLITE_DBSTATUS options] is likely ** to grow in future releases of SQLite. ** ** ^The current value of the requested parameter is written into *pCur ** and the highest instantaneous value is written into *pHiwtr. ^If ** the resetFlg is true, then the highest instantaneous value is ** reset back down to the current value. ** ** ^The sqlite3_db_status() routine returns SQLITE_OK on success and a ** non-zero [error code] on failure. ** ** See also: [sqlite3_status()] and [sqlite3_stmt_status()].

76172. The phrase the token belongs to

76173. Term/rowid iterator object

76174. Do FK constraint checks.

76175. Number of times table has been busy

76176. CAST
76177. **
76178. ERROR: unterminated delta
76179. aSpace1
76180. Snippet ellipsis text - "..."
76181. ** 2014 May 31 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Interfaces to extend FTSS. Using the interfaces defined in this file, ** FTSS may be extended with: *** ** custom tokenizers, and *** ** custom auxiliary functions.
76182. Page number to clear
76183. table_options ::=
76184. xFileControl
76185. Current db structure (or NULL)
76186. The posix advisory locking structure
76187. Score of this snippet
76188. **comment:** Compilers may complain that mem1.u.i is potentially uninitialized. ** We could initialize it, as shown here, to silence those complaints. ** But in fact, mem1.u.i will never actually be used uninitialized, and doing ** the unnecessary initialization has a measurable negative performance ** impact, since this routine is a very high runner. And so, we choose ** to ignore the compiler warnings and leave this variable uninitialized.
label: code-design
76189. Segment "age".
76190. Array of nArg CREATE VIRTUAL TABLE args
76191. ** Close a handle opened by an earlier call to sqlite3Fts5StorageOpen().
76192. Set and get the suggested cache-size for the specified pager-cache. ** ** If no global maximum is configured, then the system attempts to limit ** the total number of pages cached by purgeable pager-caches to the sum ** of the suggested cache-sizes.
76193. Trailing values to read from sorter
76194. ** Attempt to locate an element of the hash table pH with a key ** that matches pKey,nKey. Return the data for this element if it is ** found, or NULL if there is no match.
76195. sharedCacheEnabled
76196. Global context (one per db handle)
76197. Include hidden columns in output
76198. Create a new auxiliary function
76199. Number of memory cells for child frame
76200. Equal to pPage->aData
76201. If the hint read from the %_stat table is not empty, check if the ** last entry in it specifies a relative level smaller than or equal ** to the level identified by the block above (if any). If so, this ** iteration of the loop will work on merging at the hinted level.
76202. The aColumn[] value for the sPk index
76203. 183
76204. OUT: Inverse of input
76205. Arguments to the unlock callback
76206. Black magic: If the SQLITE_WriteSchema flag is set, then consider ** the schema loaded, even if errors occurred. In this situation the ** current sqlite3_prepare() operation will fail, but the following one ** will attempt to compile the supplied statement against whatever subset ** of the schema was loaded before the error occurred. The primary ** purpose of this is to allow access to the sqlite_master table ** even when its contents have been corrupted.
76207. Statement journals spill to disk when their size exceeds the following ** threshold (in bytes). 0 means that statement journals are created and ** written to disk immediately (the default behavior for SQLite versions ** before 3.12.0). -1 means always keep the entire statement journal in ** memory. (The statement journal is also always held entirely in memory ** if journal_mode=MEMORY or if temp_store=MEMORY, regardless of this ** setting.)
76208. The node from which to extract the ID
76209. If the record number will change, set register regNewRowid to ** contain the new value. If the record number is not being modified, ** then regNewRowid is the same register as regOldRowid, which is ** already populated.
76210. ** The following two macros - READ_UTF8 and WRITE_UTF8 - have been copied ** from the sqlite3 source file utf.c. If this file is compiled as part ** of the amalgamation, they are not required.
76211. Name of the pragma
76212. ** Populate the plIdx->aAvgEq[] array based on the samples currently ** stored in pIdx->aSample[].
76213. Limit the length of the LIKE or GLOB pattern to avoid problems ** of deep recursion and N*N behavior in patternCompare().
76214. OP_Insert: Set to update db->nChange
76215. Ensure the table name matches database name and that the table exists
76216. ** Generate code that will extract the iColumn-th column from ** table pTab and store the column value in a register. ** ** An effort is made to store the column value in register iReg. This ** is not guaranteed for GetColumn() - the result can be stored in ** any register. But the result is guaranteed to land in register iReg ** for GetColumnToReg(). ** ** There must be an open cursor to pTab in iTable when this routine ** is called. If iColumn<0 then code is generated that extracts the rowid.
76217. (312) nmnum ::= DELETE
76218. ** Convert a sorted list of elements into a binary tree. Make the tree ** as deep as it needs to be in order to contain the entire list.
76219. ** Validate that no temporary register falls within the range of ** iFirst..iLast, inclusive. This routine is only call from within assert() ** statements.
76220. 163
76221. ** Context object passed by sqlite3Stat4ProbeSetValue() through to ** valueNew(). See comments above valueNew() for details.
76222. **comment:** Extra space after the Index object
label: code-design
76223. synopsis: Move P3 to P1.rowid if needed
76224. Compute the content of the next row to insert into a range of ** registers beginning at regIns.
76225. ** Return an integer value for one of the parameters to the opcode pOp ** determined by character c.
76226. ':'
76227. Total number of pages in apHash
76228. Either the state is greater than PAGER_WRITER_CACHEMOD (a transaction ** or savepoint rollback done at the request of the caller) or this is ** a hot-journal rollback. If it is a hot-journal rollback, the pager ** is in state OPEN and holds an EXCLUSIVE lock. Hot-journal rollback ** only reads from the main journal, not the sub-journal.
76229. ** Print into memory obtained from sqlite3_malloc(). Omit the internal ** %-conversion extensions.
76230. Not a subquery
76231. ** Constraint: If you have ENABLE_COLUMN_METADATA then you must ** not define OMIT_DECLTYPE.
76232. A SELECT statement used in place of a table name
76233. ** Generate an instruction that will put the integer describe by ** text z[0..n-1] into register iMem. ** ** Expr.u.zToken is always UTF8 and zero-terminated.
76234. EVIDENCE-OF: R-65033-28449 The built-in BINARY collation compares ** strings byte by byte using the memcmp() function from the standard C ** library.
76235. If the EP_TokenOnly flag is set in the Expr.flags mask, then no ** space is allocated for the fields below this point. An attempt to ** access them will result in a segfault or malfunction. *****
76236. ** Public interfaces: *** ** sqlite3ConnectionBlocked() ** sqlite3ConnectionUnlocked() ** sqlite3ConnectionClosed() ** sqlite3_unlock_notify()
76237. ** This routine checks if there is a RESERVED lock held on the specified ** file by this or any other process. If such a lock is held, set *pResOut ** to a non-zero value otherwise *pResOut is set to zero. The return value ** is set to SQLITE_OK unless an I/O error occurs during lock checking.
76238. Write the result table here
76239. Calculate the hash-key for this change. If the primary key of the row ** includes a NULL value, exit early. Such changes are ignored by the ** session module.
76240. 420

76241. Mask of rev-order loops for(..)

76242. conchHeld < 0 is lockless

76243. ** The following macros mimic the standard library functions toupper(), isspace(), isalnum(), isdigit() and isxdigit(), respectively. The sqlite versions only work for ASCII characters, regardless of locale.

76244. ***** Begin file fts3_hash.h *****

76245. True for descending rowid order

76246. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** This file contains code for implementations of the r-tree and r*-tree ** algorithms packaged as an SQLite virtual table module.

76247. A new savepoint cannot be created if there are active write ** statements (i.e. open read/write incremental blob handles).

76248. The following function deletes the "minor type" or semantic value ** associated with a symbol. The symbol can be either a terminal ** or nonterminal. "fts5yymajor" is the symbol code, and "fts5ypmin" is ** a pointer to the value to be deleted. The code used to do the ** deletions is derived from the %destructor and/or %token_destructor ** directives of the input grammar.

76249. ** We have so far matched pBuilder->pNew->u.btree.nEq terms of the ** index pIndex. Try to match one more. ** When this function is called, pBuilder->pNew->nOut contains the ** number of rows expected to be visited by filtering using the nEq ** terms only. If it is modified, this value is restored before this ** function returns. ** If pProbe->nnum==0, that means pIndex is a fake index used for the ** INTEGER PRIMARY KEY.

76250. Correct answer

76251. OUT: Write encoded frame here

76252. Some basic sanity checking on the Btree. The list of Btrees ** connected by pNext and pPrev should be in sorted order by ** Btree.pBt value. All elements of the list should belong to ** the same connection. Only shared Btrees are on the list.

76253. The call assures that only valid opcodes are sent

76254. Different expressions in the index

76255. Argument to xUnlockNotify

76256. Maximum relative level value in db

76257. Step 4

76258. Size of term zTerm in bytes

76259. True if aRight[] contains PK fields only

76260. **comment:** ** Include the header file used to customize the compiler options for MSVC. ** This should be done first so that it can successfully prevent spurious ** compiler warnings due to subsequent content in this file and other files ** that are included by this file.
label: code-design

76261. ** Rollback all database files. If tripCode is not SQLITE_OK, then ** any write cursors are invalidated ("tripped" - as in "tripping a circuit ** breaker") and made to return tripCode if there are any further ** attempts to use that cursor. Read cursors remain open and valid ** but are "saved" in case the table pages are moved around.

76262. OUT: 0 for MATCH, or else an op2 value

76263. Size of the zAuthPW in bytes

76264. ** Configure the language id of a tokenizer cursor.

76265. OUT: Next free index at iAbsLevel+1

76266. Actually do the update of the change counter

76267. segdir.root

76268. Pointer to allocated object to return

76269. ** This function is used to serialize the contents of value pValue (see ** comment titled "RECORD FORMAT" above). ** If it is non-NULL, the serialized form of the value is written to ** buffer aBuf. *pnWrite is set to the number of bytes written before ** returning. Or, if aBuf is NULL, the only thing this function does is ** set *pnWrite. ** If no error occurs, SQLITE_OK is returned. Or, if an OOM error occurs ** within a call to sqlite3_value_text() (may fail if the db is utf-16) ** SQLITE_NOMEM is returned.

76270. First in list of all aux. functions

76271. IMP: R-57884-37496

76272. Parser context.

76273. Automatic index names generated from within sqlite3_declare_vtab() ** must have names that are distinct from normal automatic index names. ** The following statement converts "sqlite3_autoindex..." into ** "sqlite3_butoindex..." in order to make the names distinct. ** The "vtab_err.test" test demonstrates the need of this statement.

76274. Bytes to allocate segment root node

76275. **comment:** The opcode being coded
label: code-design

76276. Tail of unprocessed SQL

76277. First page of the database

76278. Number of extra registers needed

76279. ** This constant controls how often segments are merged. Once there are ** FTS3_MERGE_COUNT segments of level N, they are merged into a single ** segment of level N+1.

76280. Variables populated based on current entry.

76281. The result set of the SELECT

76282. ** Return the number of wildcards that can be potentially bound to. ** This routine is added to support DBD::SQLite.

76283. Parse context for sub-vdbe

76284. Initialize the output fields of the sqlite3_index_info structure

76285. Register Allocations

76286. Defer deleting the Table object associated with the ** subquery until code generation is ** complete, since there may still exist Expr.pTab entries that ** refer to the subquery even after flattening. Ticket #3346. *** pSubitem->pTab is always non-NULL by test restrictions and tests above.

76287. Aux. function implementation

76288. OUT: The affected (or effected) rowid

76289. Generate code into this virtual machine

76290. RELEASE => ID

76291. Invoke the xFilter method

76292. The KEEPNUL flag prevents OP_Eq from overwriting a NULL with 1 ** and prevents OP_Ne from overwriting NULL with 0. This flag ** is only used in contexts where either: ** (1) op==OP_Eq && (r[P2]==NULL || r[P2]==0) ** (2) op==OP_Ne && (r[P2]==NULL || r[P2]==1) ** Therefore it is not necessary to check the content of r[P2] for ** NULL.

76293. temp ::= TEMP

76294. A hash on key

76295. Allocated size of aRowidOffset[] array

76296. ** Return the page number for page pPg.

76297. Drop pages with this pgno or larger

76298. SQL function arguments to VXPrintf

76299. Number of tokens to shift snippet by

76300. Check that the %_docszie and %_content tables contain the expected ** number of rows.

76301. If iNext is FTS3_MERGE_COUNT, indicating that level iLevel is already ** full, merge all segments in level iLevel into a single iLevel+1 ** segment and allocate (newly freed) index 0 at level iLevel. Otherwise, ** if iNext is less than FTS3_MERGE_COUNT, allocate index iNext.

76302. Special processing for WITHOUT ROWID Tables

76303. ** Increment the reference count for page pPg.

76304. ** This function is an fts3ExprIterate() callback used by fts3BestSnippet(). ** Each invocation populates an element of the SnippetIter.aPhrase[] array.

76305. Index into input list
76306. 270
76307. 720
76308. OUT: Value from conflicting row
76309. Area memory mapped
76310. unlikely() or likelihood() function
76311. 218
76312. 1: trimleft 2: trimright 3: trim
76313. ** Return TRUE if the given julian day number is within range. *** The input is the JulianDay times 86400000.
76314. First token to include
76315. Allocated size of aVTrans
76316. Check for one of the special errors
76317. Apply context
76318. OUT: Buffer containing output changeset
76319. Next frame address
76320. (6)
76321. The pager object
76322. ** Return a pointer to a buffer owned by the sorter that contains the ** current key.
76323. fourth parameter
76324. ** This routine adds datatype and collating sequence information to ** the Table structures of all FROM-clause subqueries in a ** SELECT statement. *** Use this routine after name resolution.
76325. ***** This file contains automatically generated code ***** *** The code in this file has been automatically generated by *** sqlite/tool/mkkeywordhash.c *** The code in this file implements a function that determines whether ** or not a given identifier is really an SQL keyword. The same thing ** might be implemented more directly using a hand-written hash table. ** But by using this automatically generated code, the size of the code ** is substantially reduced. This is important for embedded applications ** on platforms with limited memory.
76326. Values allocated/calculated once only
76327. Database currently in use
76328. synopsis: affinity(r[P1@P2])
76329. A list of columns to be indexed
76330. Coalesced Instance Iterator
76331. List of frames to log
76332. -DSQLITE_DEFAULT_LOCKING_MODE=1 makes EXCLUSIVE the default locking ** mode. -DSQLITE_DEFAULT_LOCKING_MODE=0 make NORMAL the default locking ** mode. Doing nothing at all also makes NORMAL the default.
76333. Vararg list
76334. Thread ID
76335. OPFLAG_FORDELETE or zero, as appropriate
76336. Object used to lock page 1
76337. A BTREE_SINGLE database is always a temporary and/or ephemeral
76338. EVIDENCE-OF: R-38720-18127 The default setting is determined by the ** SQLITE_ALLOW_COVERING_INDEX_SCAN compile-time option, or is "on" if ** that compile-time option is omitted.
76339. ** Determine if an index pIdx on table with cursor iCur contains will ** the expression pExpr. Return true if the index does cover the ** expression and false if the pExpr expression references table columns ** that are not found in the index pIdx. *** An index covering an expression means that the expression can be ** evaluated using only the index and without having to lookup the ** corresponding table entry.
76340. Node data
76341. This case occurs after failing to recompile an sql statement. ** The error message from the SQL compiler has already been loaded ** into the database handle. This block copies the error message ** from the database handle into the statement and sets the statement ** program counter to 0 to ensure that when the statement is ** finalized or reset the parser error message is available via ** sqlite3_errmsg() and sqlite3_errcode().
76342. Last entry on the pEntry list
76343. Begin generating code.
76344. ***** Begin file pcache1.c *****
76345. ** This function is called from within the xFilter method. It initializes ** the full-text query currently stored in pCsr->pExpr. To iterate through ** the results of a query, the caller does: *** fts3EvalStart(pCsr); ** while(1){ *** fts3EvalNext(pCsr); ** if(pCsr->bEof) break; ** ... return row pCsr->iPrevId to the caller ... ** }
76346. Schema object being upated
76347. One or more of the following flags are set to indicate the validOK ** representations of the value stored in the Mem struct. *** If the MEM_Null flag is set, then the value is an SQL NULL value. ** For a pointer type created using sqlite3_bind_pointer() or ** sqlite3_result_pointer() the MEM_Term and MEM_Subtype flags are also set. *** If the MEM_Str flag is set then Mem.z points at a string representation. ** Usually this is encoded in the same unicode encoding as the main ** database (see below for exceptions). If the MEM_Term flag is also ** set, then the string is nul terminated. The MEM_Int and MEM_Real ** flags may coexist with the MEM_Str flag.
76348. ** Write the current contents of in-memory linked-list pList to a level-0 ** PMA in the temp file belonging to sub-task pTask. Return SQLITE_OK if ** successful, or an SQLite error code otherwise. *** The format of a PMA is: *** * A varint. This varint contains the total number of bytes of content ** in the PMA (not including the varint itself). *** * One or more records packed end-to-end in order of ascending keys. ** Each record consists of a varint followed by a blob of data (the ** key). The varint is the number of bytes in the blob of data.
76349. void*, int nByte, int min
76350. At this point, nFree contains the sum of the offset to the start ** of the cell-content area plus the number of free bytes within ** the cell-content area. If this is greater than the usable-size ** of the page, then the page must be corrupted. This check also ** serves to verify that the offset to the start of the cell-content ** area, according to the page header, lies within the page.
76351. Address of a cell
76352. serial type
76353. Db containing fts5 table
76354. ** Return the VFS structure for the pager.
76355. True if pointing to a row with no data
76356. **comment:** malloc.c filters out 0 byte requests
 label: code-design
76357. ** Implementation of the sqlite3_pcache.xShutdown method. ** Note that the static mutex allocated in xInit does ** not need to be freed.
76358. Tree contains a TK_COLLATE operator
76359. Close any rollback journal previously open
76360. True for unindexed columns
76361. Change the OP_OpenEphemeral coded earlier to an OP_Null ** sets the MEM_Cleared bit on the first register of the ** previous value. This will cause the OP_Ne below to always ** fail on the first iteration of the loop even if the first ** row is all NULLs.
76362. ** A main database named zName has just been opened. The following ** function returns a pointer to a buffer owned by SQLite that contains ** the name of the *.wal file this db connection will use. SQLite ** happens to pass a pointer to this buffer when using xAccess() ** or xOpen() to operate on the *.wal file.
76363. Initialize and shutdown the page cache subsystem
76364. Reverse unordered SELECTs
76365. Length in bytes of master journal name
76366. Expr.pLeft, .pRight, .u.pSelect all NULL
76367. ** Update the accumulator memory cells for an aggregate based on ** the current cursor position.
76368. If not NULL, increment this in DbFree()

76369. Jump here to continue with the next loop cycle
76370. Index of database containing table/index
76371. GROUP BY clause
76372. Decrement the count of locks against this same file. When the ** count reaches zero, close any other file descriptors whose close ** was deferred because of outstanding locks.
76373. 3
76374. ** Declare to the Vdbe that the BTree object at db->aDb[i] is used. ** ** The prepared statements need to know in advance the complete set of ** attached databases that will be used. A mask of these databases ** is maintained in p->btreeMask. The p->lockMask value is the subset of ** p->btreeMask of databases that will require a lock.
76375. ** Add a record to the sorter.
76376. A general purpose loop counter
76377. **comment:** Opcode: JournalMode P1 P2 P3 * * * * * Change the journal mode of database P1 to P3. P3 must be one of the ** PAGER_JOURNALMODE_XXX values. If changing between the various rollback ** modes (delete, truncate, persist, off and memory), this is a simple ** operation. No IO is required. ** * If changing into or out of WAL mode the procedure is more complicated. ** * Write a string containing the final journal-mode to register P2.
label: code-design
76378. ** The checkProfileCallback(DB,P) macro checks to see if a profile callback ** is needed, and it invokes the callback if it is needed.
76379. Largest defined DBSTATUS
76380. True for a prefix search
76381. Instruction to open the Ephemeral table
76382. we didn't find it in the hash. h points to the first
76383. Set iSz to the expected size of file pTask->file after writing the PMA. ** This is used by an assert() statement at the end of this function.
76384. ** Set the SQLITE_INDEX_SCAN_UNIQUE flag in pIdxInfo->flags. Unless this ** extension is currently being used by a version of SQLite too old to ** support index-info flags. In that case this function is a no-op.
76385. ** Return true if the two-argument version of fts3_tokenizer() ** has been activated via a prior call to sqlite3_db_config(db, ** SQLITE_DBCONFIG_ENABLE_FTS3_TOKENIZER, 1, 0);
76386. The new offset
76387. **comment:** ** For WinCE, some API function parameters do not appear to be declared as ** volatile.
label: code-design
76388. Amount of space allocated in zText
76389. SQLite connection to register module with
76390. Current bytes of pending data
76391. ** Remove all terms smaller than zTerm/nTerm from segment iIdx in absolute ** level iAbsLevel. This may involve deleting entries from the %_segments ** table, and modifying existing entries in both the %_segments and %_segdir ** tables. ** * SQLITE_OK is returned if the segment is updated successfully. Or an ** SQLite error code otherwise.
76392. ** Routines to read or write a two- and four-byte big-endian integer values.
76393. #include "fts3_tokenizer.h"
76394. ** Empty (but do not delete) a hash table.
76395. ** Add type and collation information to a column list based on ** a SELECT statement. ** * The column list presumably came from selectColumnNamesFromExprList(). ** The column list has only names, not types or collations. This ** routine goes through and adds the types and collations. ** ** This routine requires that all identifiers in the SELECT ** statement be resolved.
76396. ** Attempt to parse the given string into a julian day number. Return ** the number of errors. ** * The following are acceptable forms for the input string: ** * * * YYYY-MM-DD HH:MM:SS.FFF +/-HH:MM ** DDDD.DD ** now ** * * In the first form, the +/-HH:MM is always optional. The fractional ** seconds extension (the ".FFF") is optional. The seconds portion ** ("::SS.FFF") is option. The year and date can be omitted as long ** as there is a time string. The time string can be omitted as long ** as there is a year and date.
76397. File path associated with error
76398. This array determines which meta meta values are preserved in the ** vacuum. Even entries are the meta value number and odd entries ** are an increment to apply to the meta value after the vacuum. ** The increment is used to increase the schema cookie so that other ** connections to the same database will know to reread the schema.
76399. ** 2013 Apr 22 ** * * The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** * * May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code for the "fts3tokenize" virtual table module. ** An fts3tokenize virtual table is created as follows: ** * * CREATE VIRTUAL TABLE <tbl> USING fts3tokenize(** <tokenizer-name>, <arg-1>, ...); ** * * The table created has the following schema: ** * * CREATE TABLE <tbl>(input, token, start, end, position) ** * * When queried, the query must include a WHERE clause of type: ** * * input = <string> ** * * The virtual table module tokenizes this <string>, using the FTS3 ** tokenizer specified by the arguments to the CREATE VIRTUAL TABLE ** statement and returns one row for each token in the result. With ** fields set as follows: ** * * input: Always set to a copy of <string> ** token: A token from the input. ** start: Byte offset of the token within the input <string>. ** end: Byte offset of the byte immediately following the end of the ** token within the input string. ** pos: Token offset of token within input. **
76400. Read and decode the next log frame.
76401. Pointer to 1 byte past end of doclist
76402. Memory cell containing record
76403. ** This function is a no-op if *pRc is other than SQLITE_OK when it is called. ** Otherwise, it advances the expression passed as the second argument to ** point to the next matching row in the database. Expressions iterate through ** matching rows in docid order. Ascending order if Fts3Cursor.bDesc is zero, ** or descending if it is non-zero. ** * * If an error occurs, *pRc is set to an SQLite error code. Otherwise, if ** successful, the following variables in pExpr are set: ** ** Fts3Expr.bEof (non-zero if EOF - there is no next row) ** Fts3Expr.iDocid (valid if bEof==0. The docid of the next row) ** * * If the expression is of type FTSQUERY_PHRASE, and the expression is not ** at EOF, then the following variables are populated with the position list ** for the phrase for the visited row: ** * * Fts3Expr.pPhrase->doclist.nList (length of pList in bytes) ** Fts3Expr.pPhrase->doclist.pList (pointer to position list) ** * * It says above that this function advances the expression to the next ** matching row. This is usually true, but there are the following exceptions: ** * * 1. Deferred tokens are not taken into account. If a phrase consists ** entirely of deferred tokens, it is assumed to match every row in ** the db. In this case the position-list is not populated at all. ** * * Or, if a phrase contains one or more deferred tokens and one or ** more non-deferred tokens, then the expression is advanced to the ** next possible match, considering only non-deferred tokens. In other ** words, if the phrase is "A B C", and "B" is deferred, the expression ** is advanced to the next row that contains an instance of "A * C", ** where "*" may match any single token. The position list in this case ** is populated as for "A * C" before returning. ** * * 2. NEAR is treated as AND. If the expression is "x NEAR y", it is ** advanced to point to the next row that matches "x AND y". ** * * See sqlite3Fts3EvalTestDeferred() for details on testing if a row is ** really a match, taking into account deferred tokens and NEAR operators.
76404. Only save cursor with this iRoot. Save all if zero
76405. Offset of record to playback
76406. Number of tables or subqueries in the FROM clause
76407. Step 7: If we reach this point, we know that the result must ** be false.
76408. 1340
76409. Bytes of space for Index object + arrays
76410. Op for vtab MATCH/LIKE/GLOB/REGEXP terms
76411. ** The following macros redefine the API routines so that they are ** redirected through the global sqlite3_api structure. ** * * This header file is also used by the loadext.c source file ** (part of the main SQLite library - not an extension) so that ** it can get access to the sqlite3_api_routines structure ** definition. But the main library does not want to redefine ** the API. So the redefinition macros are only valid if the ** SQLITE_CORE macros is undefined.
76412. ** Write the buffer for the segment node pTree and all of its peers to the ** database. Then call this function recursively to write the parent of ** pTree and its peers to the database. ** * * Except, if pTree is a root node, do not write it to the database. Instead, ** set output variables *paRoot and *pnRoot to contain the root node. ** * * If successful, SQLITE_OK is returned and output variable *piLast is ** set to the largest blockid written to the database (or zero if no ** blocks were written to the db). Otherwise, an SQLite error code is ** returned.
76413. * The initial size of the Win32-specific heap. This value may be zero.

76414. ** Estimate the logarithm of the input value to base 2.
76415. Return code from sqlite3_exec()
76416. The parser
76417. OUT: Pointer to mapping
76418. ** Resize an Index object to hold N columns total. Return SQLITE_OK ** on success and SQLITE_NOMEM on an OOM error.
76419. This was the root node. Push its first rowid up to the new root.
76420. ISNULL
76421. Parsed MATCH query string
76422. occurrences
76423. **comment:** Wildcards like ":aaa", "\$aaa" or "@aaa". Reuse the same variable ** number as the prior appearance of the same name, or if the name ** has never appeared before, reuse the same variable number
label: code-design
76424. **comment:** Hash table of all pages. The following variables may only be accessed ** when the accessor is holding the PGroup mutex.
label: code-design
76425. Syntax error
76426. ** This is a utility for setting or clearing a bit-range lock on an ** AFP filesystem. ** ** Return SQLITE_OK on success, SQLITE_BUSY on failure.
76427. Used to iterate through table columns
76428. Raw text of the SQL statement
76429. Offset into aKey[] of next data element
76430. Accumulated output
76431. The VDBE cursor used to access pIdx
76432. ** The assert_nc() macro is similar to the assert() macro, except that it ** is used for assert() conditions that are true only if it can be ** guaranteed that the database is not corrupt.
76433. ** Flush any dirty pages in the pager-cache for any attached database ** to disk.
76434. ** Return a +ve value if snapshot p1 is newer than p2. A -ve value if ** p1 is older than p2 and zero if p1 and p2 are the same snapshot.
76435. Save the positions of any open cursors. This is required in ** case they are holding a reference to an xFetch reference ** corresponding to page pgnoRoot.
76436. Return true if hasAbort==mayAbort. Or if a malloc failure occurred. ** If malloc failed, then the while() loop above may not have iterated ** through all opcodes and hasAbort may be set incorrectly. Return ** true for this case to prevent the assert() in the callers frame ** from failing.
76437. Current leaf page number
76438. Type values record decode. MUST BE LAST
76439. 0 -> MATCHINFO
76440. ***** Begin file fts3.h *****
76441. ** Allowed values of VdbeOp.p4type
76442. All backends
76443. Scan past any delimiter characters before the start of the next token. ** Return SQLITE_DONE early if this takes us all the way to the end of ** the input.
76444. A WhereTerm under consideration
76445. ** CAPI3REF: Text Encodings *** ** These constant define integer codes that represent the various ** text encodings supported by SQLite.
76446. ** Write the serialized data blob for the value stored in pMem into ** buf. It is assumed that the caller has allocated sufficient space. ** Return the number of bytes written. ** ** nBuf is the amount of space left in buf[]. The caller is responsible ** for allocating enough space to buf[] to hold the entire field, exclusive ** of the pMem->u.nZero bytes for a MEM_Zero value. ** ** Return the number of bytes actually written into buf[]. The number ** of bytes in the zero-filled tail is included in the return value only ** if those bytes were zeroed in buf[].
76447. synopsis: output=r[P1@P2]
76448. For looping over PmaReader objects
76449. 1 to write. 0 read-only
76450. **comment:** If there are still other outstanding references to the shared-btree ** structure, return now. The remainder of this procedure cleans ** up the shared-btree.
label: code-design
76451. ***** Continuing where we left off in tokenize.c *****
76452. Expression (pLeft = pRight)
76453. Because of the FTS3_NODE_PADDING bytes of padding, the following is ** safe (no risk of overread) even if the node data is corrupted.
76454. SQLITE_OMIT_VACUUM
76455. If possible, improve on the iLower estimate using (\$P:\$L).
76456. Variables set by fts3SegReaderNext(). These may be read directly ** by the caller. They are valid from the time SegmentReaderNew() returns ** until SegmentReaderNext() returns something other than SQLITE_OK ** (i.e. SQLITE_DONE).
76457. EQP output string
76458. ** Seek to the offset passed as the second argument, then read cnt ** bytes into pBuf. Return the number of bytes actually read. ** ** NB: If you define USE_PREAD or USE_PREAD64, then it might also ** be necessary to define _XOPEN_SOURCE to be 500. This varies from ** one system to another. Since SQLite does not define USE_PREAD ** in any form by default, we will not attempt to define _XOPEN_SOURCE. ** See tickets #2741 and #2681. ** ** To avoid stomping the errno value on a failed read the lastErrno value ** is set before returning.
76459. ** Implementation of the sqlite3_pcache.xTruncate method. ** ** Discard all unpinned pages in the cache with a page number equal to ** or greater than parameter iLimit. Any pinned pages with a page number ** equal to or greater than iLimit are implicitly unpinned.
76460. 1 for DESC or 0 for ASC
76461. ***** Begin file analyze.c *****
76462. Allocate the Bitvec to be tested and a linear array of ** bits to act as the reference
76463. **comment:** ** Return the device characteristics for the file. ** ** This VFS is set up to return SQLITE_IOCAP_POWERSAFE_OVERWRITE by default. ** However, that choice is controversial since technically the underlying ** file system does not always provide powersafe overwrites. (In other ** words, after a power-loss event, parts of the file that were never ** written might end up being altered.) However, non-PSOW behavior is very, ** very rare. And asserting PSOW makes a large reduction in the amount ** of required I/O for journaling, since a lot of padding is eliminated. ** Hence, while POWERSAFE_OVERWRITE is on by default, there is a file-control ** available to turn it off and URI query parameter available to turn it off.
label: code-design
76464. populated by sqlite3_create_module()
76465. Number of bytes in the key
76466. ***** End of vdbeaux.c *****
76467. Bind the current output segment id to the index-writer. This is an ** optimization over binding the same value over and over as rows are ** inserted into %_idx by the current writer.
76468. Index of languageid=? value in apVal
76469. 0xffffffff is never a valid page number
76470. Number of coordinates
76471. Has no WHERE clause
76472. **comment:** ** This function is called to handle an FTS INSERT command. In other words, ** an INSERT statement of the form: ** ** INSERT INTO fts(fts)
VALUES(\$pCmd) ** INSERT INTO fts(fts, rank) VALUES(\$pCmd, \$pVal) ** ** Argument pVal is the value assigned to column "fts" by the INSERT ** statement. This function returns SQLITE_OK if successful, or an SQLite ** error code if an error occurs. ** ** The commands implemented by this function are documented in the "Special ** INSERT Directives" section of the documentation. It should be updated if ** more commands are added to this function.
label: documentation
76473. Check for condition (a)
76474. exprlist ::=
76475. "DELETE FROM %_data ... id>=? AND id<=?"
76476. ** This is the maximum number of *** ** Columns in a table *** ** Columns in an index *** ** Columns in a view *** ** Terms in the SET clause of an UPDATE statement *** ** Terms in the result set of a SELECT statement *** ** Terms in the GROUP BY or ORDER BY clauses of a SELECT statement. *** ** Terms in the

VALUES clause of an INSERT statement *** The hard upper limit here is 32676. Most database people will ** tell you that in a well-normalized database, you usually should ** not have more than a dozen or so columns in any table. And if ** that is the case, there is no point in having more than a few ** dozen values in any of the other situations described above.

76477. SQLITE_ENABLE_STAT3_OR_STAT4

76478. Frame number associated with aPgno[0]

76479. ** Check to see if element iRowid was inserted into the rowset as ** part of any insert batch prior to iBatch. Return 1 or 0. ** If this is the first test of a new batch and if there exist entries ** on pRowSet->pEntry, then sort those entries into the forest at ** pRowSet->pForest so that they can be tested.

76480. The xOpen() operation has succeeded. Set the sqlite3_file.pMethods ** pointer and, if the file is a main database file, link it into the ** mutex protected linked list of all such files.

76481. **comment:** Pointer to block from malloc()

label: code-design

76482. ORDER BY terms to skip

76483. SQLITE_MAX_EXPR_DEPTH>0

76484. MergeEngine to initialize

76485. ** CAPI3REF: SQL Trace Hook ** METHOD: sqlite3 *** ^The sqlite3_trace_v2(D,M,X,P) interface registers a trace callback ** function X against [database connection] D, using property mask M ** and context pointer P. ^If the X callback is ** NULL or if the M mask is zero, then tracing is disabled. The ** M argument should be the bitwise OR-ed combination of ** zero or more [SQLITE_TRACE] constants. ** ^Each call to either sqlite3_trace() or sqlite3_trace_v2() overrides ** (cancels) any prior calls to sqlite3_trace() or sqlite3_trace_v2(). ** ^The X callback is invoked whenever any of the events identified by ** mask M occur. ^The integer return value from the callback is currently ** ignored, though this may change in future releases. Callback ** implementations should return zero to ensure future compatibility. ** ^A trace callback is invoked with four arguments: callback(T,C,P,X). ** ^The T argument is one of the [SQLITE_TRACE] ** constants to indicate why the callback was invoked. ** ^The C argument is a copy of the context pointer. ** The P and X arguments are pointers whose meanings depend on T. ** ^The sqlite3_trace_v2() interface is intended to replace the legacy ** interfaces [sqlite3_trace()] and [sqlite3_profile()], both of which ** are deprecated.

76486. **comment:** ** Release a unixInodeInfo structure previously allocated by findInodeInfo(). ** The mutex entered using the unixEnterMutex() function must be held ** when this function is called.

label: code-design

76487. ** Ensure that there are at least nByte bytes available in the buffer. Or, ** if there are not nByte bytes remaining in the input, that all available ** data is in the buffer. ** Return an SQLite error code if an error occurs, or SQLITE_OK otherwise.

76488. ** Function to delete compiled regexp objects. Registered as ** a destructor function with sqlite3_set_auxdata().

76489. Functions used to manage pager transactions and savepoints.

76490. This routine is only be called from within a read transaction.

76491. State 1. In this state we are expecting either a 1, indicating ** that the following integer will be a column number, or the ** start of a position list for column 0. ** The only difference between state 1 and state 2 is that if the ** integer encountered in state 1 is not 0 or 1, then we need to ** increment the column 0 "nDoc" count for this term.

76492. Frame to read

76493. OUT: New session object

76494. 100

76495. ** CAPI3REF: Create An Iterator To Traverse A Changeset *** Create an iterator used to iterate through the contents of a changeset. ** If successful, *pp is set to point to the iterator handle and SQLITE_OK ** is returned. Otherwise, if an error occurs, *pp is set to zero and an ** SQLite error code is returned. ** ^The following functions can be used to advance and query a changeset ** iterator created by this function: *** ** [sqlite3changeset_next()] ** [sqlite3changeset_op()] ** [sqlite3changeset_new()] ** [sqlite3changeset_old()] ** ** ^It is the responsibility of the caller to eventually destroy the iterator ** by passing it to [sqlite3changeset_finalize()]. The buffer containing the ** changeset (pChangeset) must remain valid until after the iterator is ** destroyed. ** ^Assuming the changeset blob was created by one of the ** [sqlite3session_changeset()], [sqlite3changeset_concat()] or ** [sqlite3changeset_invert()] functions, all changes within the changeset ** that apply to a single table are grouped together. This means that when ** an application iterates through a changeset using an iterator created by ** this function, all changes that relate to a single table are visited ** consecutively. There is no chance that the iterator will visit a change ** the applies to table X, then one for table Y, and then later on visit ** another change for table X.

76496. The UNLIKELY() function is a no-op. The result is the value ** of the first argument.

76497. Used by codeCursorHint()

76498. OUT: Size of current key in bytes

76499. ** When testing, keep a count of the number of open files.

76500. ** 2005 May 23 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains functions used to access the internal hash tables ** of user defined functions and collation sequences.

76501. rank MATCH ?

76502. ***** This section contains code for WinCE only.

76503. Jump destination or storage location

76504. case_exprlist ::= WHEN expr THEN expr

76505. Do extra initialization steps requested by the SQLITE_EXTRA_INIT ** compile-time option.

76506. One entry for each token in the phrase

76507. The following are only used if the FTSS5_SEGITER_REVERSE flag is set.

76508. **comment:** Create the rbu_tmp_xxx table and the triggers to populate it.

label: code-design

76509. Previous thread used to flush PMA

76510. Value to write the string rep. of

76511. List of function arguments

76512. SQLITE_INSERT, UPDATE or DELETE

76513. Name context for processing aggregate information

76514. Add dlidx if this many empty pages

76515. Used to iterate through schemas

76516. This happens when parsing a token or quoted phrase that contains ** no token characters at all. (e.g ... MATCH "").

76517. Note: this call could be optimized away - since the same values must ** have been requested when testing key \$P in whereEqualScanEst().

76518. Same position as prev. token

76519. Full-text table handle

76520. This just creates a place-holder record in the sqlite_master table. ** The record created does not contain anything yet. It will be replaced ** by the real entry in code generated at sqlite3EndTable(). ** ^The rowid for the new entry is left in register pParse->regRowid. ** The root page number of the new table is left in reg pParse->regRoot. ** The rowid and root page number values are needed by the code that ** sqlite3EndTable will generate.

76521. Add condition to terminate at NULLs

76522. 306

76523. ** Streaming version of sqlite3session_patchset().

76524. ** Implementation of the stat_get(P,J) SQL function. This routine is ** used to query statistical information that has been gathered into ** the Stat4Accum object by prior calls to stat_push(). The P parameter ** has type BLOB but it is really just a pointer to the Stat4Accum object. ** The content to returned is determined by the parameter J ** which is one of the STAT_GET_xxxx values defined above. ** ^The stat_get(P,J) function is not available to generic SQL. It is ** inserted as part of a manually constructed bytecode program. (See ** the callStatGet() routine below.) It is guaranteed that the P ** parameter will always be a pointer to a Stat4Accum object, never a ** NULL. ** ^If neither STAT3 nor STAT4 are enabled, then J is always ** STAT_GET_STAT1 and is hence omitted and this routine becomes ** a one-parameter function, stat_get(P), that always returns the ** stat1 table entry information.

76525. ** Generate code that will tell the VDBE the declaration types of columns ** in the result set.

76526. **comment:** ** Bitmasks used by sqlite3GetVarint(). These precomputed constants ** are defined here rather than simply putting the constant expressions ** inline in order to work around bugs in the RVT compiler. ** ** SLOT_2_0 A mask for (0x7f<<14) | 0x7f ** ** SLOT_4_2_0 A mask for (0x7f<<28) | SLOT_2_0
label: code-design

76527. The node number

76528. Grab the cookie value

76529. If MEM_Null is set, then either the value is a pure NULL (the usual ** case) or it is a pointer set using sqlite3_bind_pointer() or ** sqlite3_result_pointer(). If a pointer, then MEM_Term must also be ** set.

76530. **comment:** ** 2011-07-09 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code for the VdbeSorter object, used in concert with ** a VdbeCursor to sort large numbers of keys for CREATE INDEX statements ** or by SELECT statements with ORDER BY clauses that cannot be satisfied ** using indexes and without LIMIT clauses. ** ** The VdbeSorter object implements a multi-threaded external merge sort ** algorithm that is efficient even if the number of elements being sorted ** exceeds the available memory. ** ** Here is the (internal, non-API) interface between this module and the ** rest of the SQLite system: ** ** sqlite3VdbeSorterInit() Create a new VdbeSorter object. ** ** sqlite3VdbeSorterWrite() Add a single new row to the VdbeSorter ** object. The row is a binary blob in the ** OP_MakeRecord format that contains both ** the ORDER BY key columns and result columns ** in the case of a SELECT w/ ORDER BY, or ** the complete record for an index entry ** in the case of a CREATE INDEX. ** ** sqlite3VdbeSorterRewind() Sort all content previously added. ** Position the read cursor on the ** first sorted element. ** ** sqlite3VdbeSorterNext() Advance the read cursor to the next sorted ** element. ** ** sqlite3VdbeSorterRowkey() Return the complete binary blob for the ** row currently under the read cursor. ** ** sqlite3VdbeSorterCompare() Compare the binary blob for the row ** currently under the read cursor against ** another binary blob X and report if ** X is strictly less than the read cursor. ** Used to enforce uniqueness in a ** CREATE UNIQUE INDEX statement. ** ** sqlite3VdbeSorterClose() Close the VdbeSorter object and reclaim ** all resources. ** ** sqlite3VdbeSorterReset() Refurbish the VdbeSorter for reuse. This ** is like Close() followed by Init() only ** much faster. ** ** The interfaces above must be called in a particular order. Write() can ** only occur in between Init()/Reset() and Rewind(). Next(), Rowkey(), and ** Compare() can only occur in between Rewind() and Close()/Reset(). i.e. ** ** Init() ** for each record: Write() ** Rewind() ** Rowkey() / Compare() ** Next() ** Close() ** ** Algorithm: ** ** Records passed to the sorter via calls to Write() are initially held ** unsorted in main memory. Assuming the amount of memory used never exceeds ** a threshold, when Rewind() is called the set of records is sorted using ** an in-memory merge sort. In this case, no temporary files are required ** and subsequent calls to Rowkey(), Next() and Compare() read records ** directly from main memory. ** ** If the amount of space used to store records in main memory exceeds the ** threshold, then the set of records currently in memory are sorted and ** written to a temporary file in "Packed Memory Array" (PMA) format. ** A PMA created at this point is known as a "level-0 PMA". Higher levels ** of PMAs may be created by merging existing PMAs together - for example ** merging two or more level-0 PMAs together creates a level-1 PMA. ** ** The threshold for the amount of main memory to use before flushing ** records to a PMA is roughly the same as the limit configured for the ** page-cache of the main database. Specifically, the threshold is set to ** the value returned by "PRAGMA main.page_size" multiplied by ** that returned by "PRAGMA main.cache_size", in bytes. ** ** If the sorter is running in single-threaded mode, then all PMAs generated ** are appended to a single temporary file. Or, if the sorter is running in ** multi-threaded mode then up to (N+1) temporary files may be opened, where ** N is the configured number of worker threads. In this case, instead of ** sorting the records and writing the PMA to a temporary file itself, the ** calling thread usually launches a worker thread to do so. Except, if ** there are already N worker threads running, the main thread does the work ** itself. ** ** The sorter is running in multi-threaded mode if (a) the library was built ** with pre-processor symbol SQLITE_MAX_WORKER_THREADS set to a value greater ** than zero, and (b) worker threads have been enabled at runtime by calling ** "PRAGMA threads=N" with some value of N greater than 0. ** ** When Rewind() is called, any data remaining in memory is flushed to a ** final PMA. So at this point the data is stored in some number of sorted ** PMAs within temporary files on disk. ** ** If there are fewer than SORTER_MAX_MERGE_COUNT PMAs in total and the ** sorter is running in single-threaded mode, then these PMAs are merged ** incrementally as keys are retrieved from the sorter by the VDBE. The ** MergeEngine object, described in further detail below, performs this ** merge. ** ** Or, if running in multi-threaded mode, then a background thread is ** launched to merge the existing PMAs. Once the background thread has ** merged T bytes of data into a single sorted PMA, the main thread ** begins reading keys from that PMA while the background thread proceeds ** with merging the next T bytes of data. And so on. ** ** Parameter T is set to half the value of the memory threshold used ** by Write() above to determine when to create a new PMA. ** ** If there are more than SORTER_MAX_MERGE_COUNT PMAs in total when ** Rewind() is called, then a hierarchy of incremental-merges is used. ** First, T bytes of data from the first SORTER_MAX_MERGE_COUNT PMAs on ** disk are merged together. Then T bytes of data from the second set, and ** so on, such that no operation ever merges more than SORTER_MAX_MERGE_COUNT ** PMAs at a time. This done is to improve locality. ** ** If running in multi-threaded mode and there are more than ** SORTER_MAX_MERGE_COUNT PMAs on disk when Rewind() is called, then more ** than one background thread may be created. Specifically, there may be ** one background thread for each temporary file on disk, and one background ** thread to merge the output of each of the others to a single PMA for ** the main thread to read from.

label: code-design

76531. Length of BTM vector

76532. RELEASE

76533. ** The following version of unixSectorSize() is optimized for QNX.

76534. Opcode: Savepoint P1 * * P4 * * ** Open, release or rollback the savepoint named by parameter P4, depending ** on the value of P1. To open a new savepoint, P1==0. To release (commit) an ** existing savepoint, P1==1, or to rollback an existing savepoint P1==2.

76535. res might be negative because the table is empty. Check to ** see if this is the case.

76536. A DELETE operation. Foreign key processing is required if the ** table in question is either the child or parent table for any ** foreign key constraint.

76537. ** Shorthand for the functions above

76538. Replacement content for JNODE_REPLACE

76539. ** Close the read-only blob handle, if it is open.

76540. ** Page paths: ** ** The value of the 'path' column describes the path taken from the ** root-node of the b-tree structure to each page. The value of the ** root-node path is '/'. ** ** The value of the path for the left-most child page of the root of ** a b-tree is '000/'. (Btrees store content ordered from left to right ** so the pages to the left have smaller keys than the pages to the right.) ** The next to left-most child of the root page is ** '001', and so on, each sibling page identified by a 3-digit hex ** value. The children of the 451st left-most sibling have paths such ** as '/1c2/000/, '/1c2/001' etc. ** ** Overflow pages are specified by appending a '+' character and a ** six-digit hexadecimal value to the path to the cell they are linked ** from. For example, the three overflow pages in a chain linked from ** the left-most cell of the 450th child of the root page are identified ** by the paths: ** ** '/1c2/000+000000' // First page in overflow chain ** '/1c2/000+000001' // Second page in overflow chain ** '/1c2/000+000002' // Third page in overflow chain ** ** If the paths are sorted using the BINARY collation sequence, then ** the overflow pages associated with a cell will appear earlier in the ** sort-order than its child page: ** ** '/1c2/000' // Left-most child of 451st child of root

76541. The index being used for this loop

76542. 'l' is a single-byte character

76543. FROM clause not a virtual table

76544. 'l', [...] style quoted ids

76545. Number of arguments to the function

76546. ** The hex() function. Interpret the argument as a blob. Return ** a hexadecimal rendering as text.

76547. The table containing the blob

76548. Default return value, in case of an error

76549. **comment:** Too many digits

label: code-design

76550. Condition 3

76551. True if ORDER BY [rank|rowid] DESC

76552. nullRow[] is an OP_Record encoding of a row containing 5 NULLs

76553. The group mutex must be released before pcache1Alloc() is called. This ** is because it might call sqlite3_release_memory(), which assumes that ** this mutex is not held.

76554. If all the seg-readers are at EOF, we're finished. return SQLITE_OK.

76555. ** Generate a human-readable description of a Select object.

76556. CHECK constraint references the ROWID

76557. Write the integer values

76558. ** If the input FuncDef structure is ephemeral, then free it. If ** the FuncDef is not ephemeral, then do nothing.

76559. FTSQUERY_PHRASE expression
76560. True if pPage is a leaf of a LEAFDATA tree
76561. Index of second table in pSrc
76562. If page one was fetched successfully, and this function is not ** operating in direct-mode, make page 1 writable. When not in ** direct mode, page 1 is always held in cache and hence the PagerGet() ** above is always successful - hence the ALWAYS on rc==SQLITE_OK.
76563. Token argument. Might be NULL
76564. Value to write to hash-table slot
76565. Implied first min-heap entry
76566. 15
76567. Input iterator
76568. ** Perform a reduce action and the shift that must immediately ** follow the reduce.
76569. **comment:** ** CAPI3REF: Record A Database Snapshot ** EXPERIMENTAL ** ** ^The [sqlite3_snapshot_get(D,S,P)] interface attempts to make a ** new [sqlite3_snapshot] object that records the current state of ** schema S in database connection D. ^On success, the ** [sqlite3_snapshot_get(D,S,P)] interface writes a pointer to the newly ** created [sqlite3_snapshot] object into *P and returns SQLITE_OK. ** If there is not already a read-transaction open on schema S when this function is called, one is opened automatically. ** ** The following must be true for this function to succeed. If any of ** the following statements are false when sqlite3_snapshot_get() is ** called, SQLITE_ERROR is returned. The final value of *P is undefined ** in this case. ** ** ** The database handle must be in [autocommit mode]. ** ** Schema S of [database connection] D must be a [WAL mode] database. ** ** There must not be a write transaction open on schema S of database ** connection D. ** ** One or more transactions must have been written to the current wal ** file since it was created on disk (by any connection). This means ** that a snapshot cannot be taken on a wal mode database with no wal ** file immediately after it is first opened. At least one transaction ** must be written to it first. ** ** ** This function may also return SQLITE_NOMEM. If it is called with the ** database handle in autocommit mode but fails for some other reason, ** whether or not a read transaction is opened on schema S is undefined. ** ** The [sqlite3_snapshot] object returned from a successful call to ** [sqlite3_snapshot_get()] must be freed using [sqlite3_snapshot_free()] ** to avoid a memory leak. ** ** The [sqlite3_snapshot_get()] interface is only available when the ** SQLITE_ENABLE_SNAPSHOT compile-time option is used.
label: code-design
76570. New pX->pLeft vector
76571. If this is a scalar select that is part of an expression, then ** store the results in the appropriate memory cell and break out ** of the scan loop.
76572. Database schema version number for this file
76573. ** Return the number of changes since the database handle was opened.
76574. ** If the WHERE_GROUPBY flag is set in the mask passed to sqlite3WhereBegin(), ** the planner assumes that the specified pOrderBy list is actually a GROUP BY clause - and so any order that groups rows as required satisfies the ** request. ** ** Normally, in this case it is not possible for the caller to determine ** whether or not the rows are really being delivered in sorted order, or ** just in some other order that provides the required grouping. However, ** if the WHERE_SORTBYGROUP flag is also passed to sqlite3WhereBegin(), then ** this function may be called on the returned WhereInfo object. It returns ** true if the rows really will be sorted in the specified order, or false ** otherwise. ** ** For example, assuming: ** ** CREATE INDEX i1 ON t1(x, Y); ** ** then ** ** SELECT * FROM t1 GROUP BY x,y ORDER BY x,y; -- IsSorted() == 1 ** SELECT * FROM t1 GROUP BY y,x ORDER BY y,x; -- IsSorted() == 0
76575. **comment:** Following an incremental-merge operation, assuming that the input ** segments are not completely consumed (the usual case), they are updated ** in place to remove the entries that have already been merged. This ** involves updating the leaf block that contains the smallest unmerged ** entry and each block (if any) between the leaf and the root node. So ** if the height of the input segment b-trees is N, and input segments ** are merged eight at a time, updating the input segments at the end ** of an incremental-merge requires writing (8*(1+N)) blocks. N is usually ** small - often between 0 and 2. So the overhead of the incremental ** merge is somewhere between 8 and 24 blocks. To avoid this overhead ** dwarfing the actual productive work accomplished, the incremental merge ** is only attempted if it will write at least 64 leaf blocks. Hence ** nMinMerge. ** ** Of course, updating the input segments also involves deleting a bunch ** of blocks from the segments table. But this is not considered overhead ** as it would also be required by a crisis-merge that used the same input ** segments.
label: code-design
76576. 72
76577. Trigger this program was coded from
76578. Blob read from %_segments table
76579. Values between 128 and 16383
76580. An element of pSelect->pSrc
76581. All CHECK constraints
76582. Null-terminated URI to parse
76583. ** Allocate a new Fts5Iter object. ** ** The new object will be used to iterate through data in structure pStruct. ** If iLevel is -ve, then all data in all segments is merged. Or, if iLevel ** is zero or greater, data from the first nSegment segments on level iLevel ** is merged. ** ** The iterator initially points to the first term/rowid entry in the ** iterated data.
76584. Number of leaf pages in unit of work
76585. ** Move to the next matching rowid that occurs at or after iMatch. The ** definition of "at or after" depends on whether this iterator iterates ** in ascending or descending rowid order.
76586. ** This is called as part of an incremental checkpoint operation. Copy ** a single frame of data from the wal file into the database file, as ** indicated by the RbuFrame object.
76587. Write the first journal header to the journal file and open ** the sub-journal if necessary.
76588. FTS Table handle
76589. ***** End of rtree.c *****
76590. The FK maps to the IPK if any of the following are true: ** ** 1) There is an INTEGER PRIMARY KEY column and the FK is implicitly ** mapped to the primary key of table pParent, or ** 2) The FK is explicitly mapped to a column declared as INTEGER ** PRIMARY KEY.
76591. Object iterator for column names
76592. same as TK_ISNULL, synopsis: if r[P1]==NULL goto P2
76593. Write the rowid.
76594. No-op if we do not support a codec
76595. Initial cost estimate
76596. Reanalyze if the table is 25 times larger than the last analysis
76597. Initial space for EQP output string
76598. The cache to truncate
76599. x IN (...)
76600. Array of opcodes for sub-program
76601. table of a LEFT JOIN
76602. ** Register a profile function. The pArg from the previously registered ** profile function is returned. ** ** A NULL profile function means that no profiling is executed. A non-NULL ** profile is a pointer to a function that is invoked at the conclusion of ** each SQL statement that is run.
76603. Invoke the xNext() method of the module. There is no way for the ** underlying implementation to return an error if one occurs during ** xNext(). Instead, if an error occurs, true is returned (indicating that ** data is available) and the error code returned when xColumn or ** some other method is next invoked on the save virtual table cursor.
76604. Allocate the sqlite3_index_info structure
76605. Mark the page as clean.
76606. ** The Table structure pTable is really a VIEW. Fill in the names of ** the columns of the view in the pTable structure. Return the number ** of errors. If an error is seen leave an error message in pParse->zErrMsg.
76607. State vector for the DB fixer
76608. Search point for the leaf
76609. ** Number of malloc size increments to track.
76610. If p==0 (unmap the entire file) then there must be no outstanding ** xFetch references. Or, if p!=0 (meaning it is an xFetch reference), ** then there must be at least one outstanding.
76611. ** Clean up any resources allocated as part of the iterator object passed ** as the only argument.

76612. ** Truncate an rbuVfs-file.
76613. GetVersionEx() is deprecated
76614. ** If the virtual table pVtab supports the transaction interface (** xBegin/xRollback/xCommit and optionally xSync) and a transaction is ** not currently open, invoke the xBegin method now. ** ** If the xBegin call is successful, place the sqlite3_vtab pointer ** in the sqlite3.aVTrans array.
76615. ** Each SQL index is represented in memory by an ** instance of the following structure. ** ** The columns of the table that are to be indexed are described ** by the aiColumn[] field of this structure. For example, suppose ** we have the following table and index: ** ** CREATE TABLE Ex1(c1 int, c2 int, c3 text); ** CREATE INDEX Ex2 ON Ex1(c3,c1); ** ** In the Table structure describing Ex1, nCol==3 because there are ** three columns in the table. In the Index structure describing ** Ex2, nColumn==2 since 2 of the 3 columns of Ex1 are indexed. ** The value of aiColumn is {2, 0}. aiColumn[0]==2 because the ** first column to be indexed (c3) has an index of 2 in Ex1.aCol[]. ** The second column to be indexed (c1) has an index of 0 in ** Ex1.aCol[], hence Ex2.aiColumn[1]==0. ** ** The Index.onError field determines whether or not the indexed columns ** must be unique and what to do if they are not. When Index.onError=OE_Non, ** it means this is not a unique index. Otherwise it is a unique index ** and the value of Index.onError indicate the which conflict resolution ** algorithm to employ whenever an attempt is made to insert a non-unique ** element. ** ** While parsing a CREATE TABLE or CREATE INDEX statement in order to ** generate VDBE code (as opposed to parsing one read from an sqlite_master ** table as part of parsing an existing database schema), transient instances ** of this structure may be created. In this case the Index.tnum variable is ** used to store the address of a VDBE instruction, not a database page ** number (it cannot - the database page is not allocated until the VDBE ** program is executed). See convertToWithoutRowidTable() for details.
76616. Any unrecognized conversion type
76617. ** Add an opcode that includes the p4 value with a P4_INT64 or ** P4_REAL type.
76618. 0==closed, 1==open, 2==synced
76619. EVIDENCE-OF: R-30189-54097 For each limit category SQLITE_LIMIT_NAME ** there is a hard upper bound set at compile-time by a C preprocessor ** macro called SQLITE_MAX_NAME. (The "_LIMIT_" in the name is changed to ** "_MAX_".)
76620. Name of the common-table
76621. sqlite3WalEndReadTransaction() was not called for the previous ** transaction in locking_mode=EXCLUSIVE. So call it now. If we ** are in locking_mode=NORMAL and EndRead() was previously called, ** the duplicate call is harmless.
76622. Silently ignore database qualifiers inside CHECK constraints and ** partial indices. Do not raise errors because that might break ** legacy and because it does not hurt anything to just ignore the ** database name.
76623. Generate code into this VDBE
76624. **comment:** ** An iterator of this type is used to iterate through all objects in ** the target database that require updating. For each such table, the ** iterator visits, in order: ** ** * the table itself, ** * each index of the table (zero or more points to visit), and ** * a special "cleanup table" state. ** ** abIndexed: ** If the table has no indexes on it, abIndexed is set to NULL. Otherwise, ** it points to an array of flags nTblCol elements in size. The flag is ** set for each column that is either a part of the PK or a part of an ** index. Or clear otherwise. **
label: code-design
76625. ***** Begin file fts3_tokenizer.c *****
76626. ** *pRoot is the head of a list of free chunks of the same size ** or same size hash. In other words, *pRoot is an entry in either ** mem3.aiSmall[] or mem3.aiHash[]. ** ** This routine examines all entries on the given list and tries ** to coalesce each entries with adjacent free chunks. ** ** If it sees a chunk that is larger than mem3.iMaster, it replaces ** the current mem3.iMaster with the new larger chunk. In order for ** this mem3.iMaster replacement to work, the master chunk must be ** linked into the hash tables. That is not the normal state of ** affairs, of course. The calling routine must link the master ** chunk before invoking this routine, then must unlink the (possibly ** changed) master chunk once this routine has finished.
76627. Array of new.* values
76628. Number of pages starting at pg1 to journal
76629. Allocate readers for this expression
76630. Record consists of PK fields only
76631. Depth of node containing pCell
76632. Table Name NULL
76633. Start offset in pFile
76634. When converting from UTF-8 to UTF-16 the maximum growth is caused ** when a 1-byte UTF-8 character is translated into a 2-byte UTF-16 ** character. Two bytes are required in the output buffer for the ** nul-terminator.
76635. TUNING: If there is both an upper and lower limit and neither limit ** has an application-defined likelihood(), assume the range is ** reduced by an additional 75%. This means that, by default, an open-ended ** range query (e.g. col > ?) is assumed to match 1/4 of the rows in the ** index. While a closed range (e.g. col BETWEEN ? AND ?) is estimated to ** match 1/64 of the index.
76636. Opcode: Subtract P1 P2 P3 * * ** Synopsis: r[P3]=r[P2]-r[P1] ** ** Subtract the value in register P1 from the value in register P2 ** and store the result in register P3. ** If either input is NULL, the result is NULL.
76637. WINSHM_UNLCK, WINSHM_RDLCK, or WINSHM_WRLCK
76638. If a content=xxx option was specified, the following: ** ** 1. Ignore any compress= and uncompress= options. ** ** 2. If no column names were specified as part of the CREATE VIRTUAL ** TABLE statement, use all columns from the content table.
76639. **comment:** This function is used by SQL generated to implement the ** ALTER TABLE command. The first argument is the text of a CREATE TRIGGER ** statement. The second is a table name. The table name in the CREATE ** TRIGGER statement is replaced with the third argument and the result ** returned. This is analogous to renameTableFunc() above, except for CREATE ** TRIGGER, not CREATE INDEX and CREATE TABLE.
label: code-design
76640. SQLITE_ENABLE_MEMSYS5
76641. Values for the OP_Variable opcode.
76642. seltablist ::= stl_prefix LP seltablist RP as on_opt using_opt
76643. 6
76644. Index into pNode into which pCell is written
76645. Extract the content for the p2+1-th column. Control can only ** reach this point if aOffset[p2], aOffset[p2+1], and PC->aType[p2] are ** all valid.
76646. If any database other than TEMP is reset, then also reset TEMP ** since TEMP might be holding triggers that reference tables in the ** other database.
76647. Value of a host parameter
76648. Buffer containing serialized structure
76649. If this is a delete, decrement nPhaseOneStep by nIndex. If the DELETE ** statement below does actually delete a row, nPhaseOneStep will be ** incremented by the same amount when SQL function rbu_tmp_insert() ** is invoked by the trigger.
76650. ** Count the number of times that the LIKE operator (or GLOB which is ** just a variation of LIKE) gets called. This is used for testing ** only.
76651. Number of valid a[] entries
76652. **comment:** Maximum allowed allocation. 0 for no malloc usage
label: code-design
76653. if defined SQLITE OMIT_AUTOVACUUM
76654. iLikeRepCntr actually stores 2x the counter register number. The ** bottom bit indicates whether the search order is ASC or DESC.
76655. #include "sqlite3ext.h"
76656. Prefix character. "+" or "-" or " " or '\0'.
76657. ** Find the appropriate action for a parser given the terminal ** look-ahead token iLookAhead.
76658. An internal node.
76659. ** Erase all information in a table and add the root of the table to ** the freelist. Except, the root of the principle table (the one on ** page 1) is never added to the freelist. ** ** This routine will fail with SQLITE_LOCKED if there are any open ** cursors on the table. ** ** If AUTOVACUUM is enabled and the page at iTable is not the last ** root page in the database file, then the last root page ** in the database file is moved into the slot formerly occupied by ** iTable and that last slot formerly occupied by the last root page ** is added to the freelist instead of iTable. In this way, all ** root pages are kept at the beginning of the database file, which ** is necessary for AUTOVACUUM to work right. *piMoved is set to the ** page number that used to be the last root page in the file before ** the move. If no page gets moved, *piMoved is set to 0. ** The last root page is recorded in meta[3] and the value of ** meta[3] is updated by this procedure.
76660. Update the .nOut value of this loop
76661. File-system block size
76662. Extend the p->aAlloc[] allocation if required.

76663. ** Convert a double into a LogEst ** In other words, compute an approximation for 10*log2(x).
76664. Size of name in zName[]
76665. ** Return the number of references to the specified page.
76666. True if there are triggers or FKs or ** subqueries in the WHERE clause
76667. Status code
76668. If there is no open read-transaction on the source database, open ** one now. If a transaction is opened here, then it will be closed ** before this function exits.
76669. Compute the limit registers
76670. The Loop object
76671. ** Return true if the underlying VFS for the given pager supports the ** primitives necessary for write-ahead logging.
76672. **comment:** ** Generate a human-readable explanation of an expression list.
 label: code-design
76673. First register holding data to be sorted
76674. Index of next token to be returned
76675. Opening highlight
76676. The "docid >= ?" constraint, if any
76677. Use the SQLite core versions if this routine is part of the ** SQLite amalgamation
76678. ** Add an error message to pParse->zErrMsg and increment pParse->nErr. ** The following formatting characters are allowed: *** %s Insert a string *** %z A string that should be freed after use *** %d Insert an integer *** %T Insert a token *** %S Insert the first element of a SrcList *** This function should be used to report any error that occurs while ** compiling an SQL statement (i.e. within sqlite3_prepare()). The ** last thing the sqlite3_prepare() function does is copy the error ** stored by this function into the database handle using sqlite3Error(). ** Functions sqlite3Error() or sqlite3ErrorWithMsg() should be used ** during statement execution (sqlite3_step() etc.).
76679. ** At this point, variables are initialized as follows: *** flag_alternateform TRUE if a '#' is present. ** flag_altform2 TRUE if a '!' is present. ** flag_prefix '+' or '-' or zero ** flag_leftjustify TRUE if a '-' is present or if the ** field width was negative. ** flag_zeropad TRUE if the width began with 0. ** flag_long 1 for "l", 2 for "ll" ** width The specified field width. This is ** always non-negative. Zero is the default. ** precision The specified precision. The default ** is -1. ** xtype The class of the conversion. ** infop Pointer to the appropriate info struct.
76680. ***** End of mem1.c *****
76681. Connection
76682. Array element to populate
76683. Previous rowid value written to page
76684. with
76685. Discard pTemplate
76686. Number of pragmas: 60 on by default, 77 total.
76687. ''
76688. Number of pages on the freelist
76689. ** Set the FTS5CSR_REQUIRE_RESEEK flag on all FTS5_PLAN_MATCH cursors ** open on table pTab.
76690. A collating sequence
76691. ** PRAGMA [schema].incremental_vacuum(N) ** ** Do N steps of incremental vacuuming on a database.
76692. SELECT may not have a GROUP BY clause
76693. ** sqlite3MallocDisallow() increments the following counter. ** sqlite3MallocAllow() decrements it.
76694. Maximum length of zSql in bytes.
76695. String inserted before highlighted term
76696. This loop runs once for each entry in the blocked-connections list.
76697. Index of left-hand Fts5SegIter
76698. ** Create a temporary file name in zBuf. zBuf must be allocated ** by the calling process and must be big enough to hold at least ** pVfs->mxPathname bytes.
76699. 1
76700. ** The 'pre-update' hook registered by this module with SQLite databases.
76701. Name of target db table
76702. Base class - must be first
76703. 146
76704. No LIMIT means no OFFSET
76705. ** Release memory held by the Mem p, both external memory cleared ** by p->xDel and memory in p->zMalloc. ** ** This is a helper routine invoked by sqlite3VdbeMemRelease() in ** the unusual case where there really is memory in p that needs ** to be freed.
76706. Invalidate all incrblob cursors open on table iTable (assuming iTable ** is the root of a table b-tree - if it is not, the following call is ** a no-op).
76707. If doNot is true, then add a TK_NOT Expr-node wrapper around the ** outside of *ppExpr.
76708. If not initializing, then create a record for the new table ** in the SQLITE_MASTER table of the database. ** ** If this is a TEMPORARY table, write the entry into the auxiliary ** file instead of into the main database file.
76709. Desired maximum mmap size
76710. REFERENCES
76711. Length of aDoclist in bytes
76712. ** On recent Windows platforms, the localtime_s() function is available ** as part of the "Secure CRT". It is essentially equivalent to ** localtime_r() available under most POSIX platforms, except that the ** order of the parameters is reversed. ** ** See [http://msdn.microsoft.com/en-us/library/a442x3ye\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/a442x3ye(VS.80).aspx). ** If the user has not indicated to use localtime_r() or localtime_s() ** already, check for an MSVC build environment that provides ** localtime_s().
76713. Advance to the next term of the compound
76714. ** The following types are used as part of the implementation of the ** fts3BestSnippet() routine.
76715. Form the KeyInfo object from this ExprList
76716. a pointer to a SrcList
76717. **comment:** The following value requires a mutex to change. We skip the mutex on ** reading because (1) most platforms read a 32-bit integer atomically and ** (2) even if an incorrect value is read, no great harm is done since this ** is really just an optimization.
 label: code-design
76718. Free the contents of the apShm[] array. And the array itself.
76719. This is a utility routine used to set the ExprSpan.zStart and ** ExprSpan.zEnd values of pOut so that the span covers the complete ** range of text beginning with pStart and going to the end of pEnd.
76720. ** Deallocate a WhereClause structure. The WhereClause structure ** itself is not freed. This routine is the inverse of ** sqlite3WhereClauseInit().
76721. Opcode: IfPos P1 P2 P3 * * * Synopsis: if r[P1]>0 then r[P1]=P3, goto P2 ** ** Register P1 must contain an integer. ** If the value of register P1 is 1 or greater, subtract P3 from the ** value in P1 and jump to P2. ** ** If the initial value of register P1 is less than 1, then the ** value is unchanged and control passes through to the next instruction.
76722. Target table for DELETE, UPDATE, INSERT
76723. **comment:** 0x8000 not currently used
 label: code-design
76724. 0x01. Next integer will be a column number.
76725. 30..37 01234567
76726. OUT: pointer to doclist
76727. To assert pCache->nPage is correct
76728. Result of sqlite3_reset()
76729. OFFSET value. NULL means no offset
76730. Number of slots allocated for aPoint[]
76731. ** Query status information.

76732. ** Return TRUE if the mask of type in eType matches no bits of the type of the ** allocation p. Also return true if p==NULL. *** This routine is designed for use within an assert() statement, to ** verify the type of an allocation. For example: *** assert(sqlite3MemdebugNoType(p, MEMTYPE_LOOKASIDE));

76733. ** An instance of this structure is used to manage a pair of buffers, each ** (nElem * sizeof(u32)) bytes in size. See the MatchinfoBuffer code below ** for details.

76734. **comment:** Calculate a hash value for this term. This is the same hash checksum ** used by the fts5_hash.c module. This is not important for correct ** operation of the module, but is necessary to ensure that some tests ** designed to produce hash table collisions really do work.

label: code-design

76735. ** Reset the tokenizer cursor passed as the only argument. As if it had ** just been returned by fts3tokOpenMethod().

76736. Length of zMod in bytes

76737. IMP: R-25361-16150 This function is omitted from SQLite by default. It ** is only available if the SQLITE_SOUNDEX compile-time option is used ** when SQLite is built.

76738. 0x8

76739. database name - used by the update hook

76740. Content of the new cell

76741. Unused file descriptors to close

76742. **comment:** Make sure the new table name does not collide with an existing ** index or table name in the same database. Issue an error message if ** it does. The exception is if the statement being parsed was passed ** to an sqlite3_declare_vtab() call. In that case only the column names ** and types will be used, so there is no need to test for namespace ** collisions.

label: code-design

76743. ** Each open file is managed by a separate instance of the "Pager" structure.

76744. Array of collation sequence names for index

76745. NEED_SYNC can be set independently of WRITEABLE. This can happen, ** for example, when using the sqlite3PagerDontWrite() optimization: ** (1) Page X is journalled, and gets WRITEABLE and NEED_SEEK. ** (2) Page X moved to freelist, WRITEABLE is cleared ** (3) Page X reused, WRITEABLE is set again ** If NEED_SYNC had been cleared in step 2, then it would not be reset ** in step 3, and page might be written into the database without first ** syncing the rollback journal, which might cause corruption on a power ** loss. *** Another example is when the database page size is smaller than the ** disk sector size. When any page of a sector is journalled, all pages ** in that sector are marked NEED_SYNC even if they are still CLEAN, just ** in case they are later modified, since all pages in the same sector ** must be journalled and synced before any of those pages can be safely ** written.

76746. type of the previous token

76747. **comment:** Initialize the malloc() system and the recursive pInitMutex mutex. ** This operation is protected by the STATIC_MASTER mutex. Note that ** MutexAlloc() is called for a static mutex prior to initializing the ** malloc subsystem - this implies that the allocation of a static ** mutex must not require support from the malloc subsystem.

label: code-design

76748. The precision in %q and %Q means how many input characters to ** consume, not the length of the output... ** if(precision>=0 && precision<length) length = precision;

76749. ** cnt==0 means there was not match. cnt>1 means there were two or ** more matches. Either way, we have an error.

76750. The TK_SELECT_COLUMN Expr node: *** pLeft: pVector containing TK_SELECT. Not deleted. ** pRight: not used. But recursively deleted. ** iColumn: Index of a column in pVector ** iTable: 0 or the number of columns on the LHS of an assignment ** pLeft->iTable: First in an array of register holding result, or 0 ** if the result is not yet computed. *** sqlite3ExprDelete() specifically skips the recursive delete of ** pLeft on TK_SELECT_COLUMN nodes. But pRight is followed, so pVector ** can be attached to pRight to cause this node to take ownership of ** pVector. Typically there will be multiple TK_SELECT_COLUMN nodes ** with the same pLeft pointer to the pVector, but only one of them ** will own the pVector.

76751. Jump here (skipping the main loop body subroutine) if the ** current sub-WHERE row is a duplicate from prior sub-WHERES.

76752. COMMIT => nothing

76753. Has no LIMIT clause

76754. WRITEABLE pages must also be DIRTY

76755. Number of bits handled by each apSub[] entry.

76756. Release the reference to the new page.

76757. When reducing a lock such that other processes can start ** reading the database file again, make sure that the ** transaction counter was updated if any part of the database ** file changed. If the transaction counter is not updated, ** other connections to the same file might not realize that ** the file has changed and hence might not know to flush their ** cache. The use of a stale cache can lead to database corruption.

76758. Object completely outside of query region

76759. VDBE under construction

76760. Largest permissible value of i

76761. ** The following singleton contains the global configuration for ** the SQLite library.

76762. The last rowid in the doclist may not be on the current page. Search ** forward to find the page containing the last rowid.

76763. Size of every page in this cache

76764. ** Return the size of the prefix, in bytes, that buffer ** (pNew/<length-unknown>) shares with buffer (pOld/nOld). *** Buffer (pNew/<length-unknown>) is guaranteed to be greater ** than buffer (pOld/nOld).

76765. Offset of header in journal file

76766. 6: (start_constraints && startEq && !bRev)

76767. Opcode: Rewind P1 P2 * * * * * The next use of the Rowid or Column or Next instruction for P1 ** will refer to the first entry in the database table or index. ** If the table or index is empty, jump immediately to P2. ** If the table or index is not empty, fall through to the following ** instruction. *** This opcode leaves the cursor configured to move in forward order, ** from the beginning toward the end. In other words, the cursor is ** configured to use Next, not Prev.

76768. ** This vector defines all the methods that can operate on an ** sqlite3_file for win32.

76769. Write the current in-memory list to a PMA. When the VdbeSorterWrite() ** function flushes the contents of memory to disk, it immediately always ** creates a new list consisting of a single key immediately afterwards. ** So the list is never empty at this point.

76770. Value of %_segdir.start_block

76771. ** Release all the table locks (locks obtained via calls to ** the setSharedCacheTableLock() procedure) held by Btree object p. *** This function assumes that Btree p has an open read or write ** transaction. If it does not, then the BTS_PENDING flag ** may be incorrectly cleared.

76772. ** Structure allocated for each backup operation.

76773. **comment:** The pager this page is part of

label: documentation

76774. isOrdered for (pFrom+pWLoop)

76775. Condition 1

76776. ** Resolve an expression that was part of an ATTACH or DETACH statement. This ** is slightly different from resolving a normal SQL expression, because simple ** identifiers are treated as strings, not possible column names or aliases. *** i.e. if the parser sees: *** ATTACH DATABASE abc AS def *** it treats the two expressions as literal strings 'abc' and 'def' instead of ** looking for columns of the same name. *** This only applies to the root node of pExpr, so the statement: *** ATTACH DATABASE abc||def AS 'db2' *** will fail because neither abc or def can be resolved.

76777. Calculate the length in bytes and the checksum of zMaster

76778. 323

76779. Cell extends past end of page

76780. Each element in the hash table is an instance of the following ** structure. All elements are stored on a single doubly-linked list. *** Again, this structure is intended to be opaque, but it can't really ** be opaque because it is used by macros.

76781. True if pParent is a root-page

76782. **comment:** Balance the tree. If the entry deleted was located on a leaf page, ** then the cursor still points to that page. In this case the first ** call to balance() repairs the tree, and the if(...) condition is ** never true. *** Otherwise, if the entry deleted was on an internal node page, then ** pCur is pointing to the leaf page from which a cell was removed to ** replace the cell deleted from the internal node. This is slightly ** tricky as the leaf node may be underfull, and the internal node may ** be either under or overfull. In this case run the balancing algorithm ** on the leaf node first. If the balance proceeds far enough up the ** tree that we can be sure that any problem in the internal node has ** been corrected, so be it. Otherwise, after balancing the leaf node, ** walk the cursor up the tree to the internal node and balance it as ** well.

label: code-design

76783. ** A cursor is a pointer to a particular entry within a particular b-tree within a database file. *** The entry is identified by its MemPage and the index in ** MemPage.aCell[] of the entry. *** A single database file can be shared by two more database connections, ** but cursors cannot be shared. Each cursor is associated with a ** particular database connection identified BtCursor.pBtree.db. *** Fields in this structure are accessed under the BtShared.mutex ** found at self->pBt->mutex. *** skipNext meaning: ** eState==SKIPNEXT && skipNext>0: Next sqlite3BtreeNext() is no-op. ** eState==SKIPNEXT && skipNext<0: Next sqlite3BtreePrevious() is no-op. ** eState==FAULT: Cursor fault with skipNext as error code.
76784. Use the standard library for separate compilation
76785. Search for a deferred foreign key constraint for which this table ** is the child table. If one cannot be found, return without ** generating any VDBE code. If one can be found, then jump over ** the entire DELETE if there are no outstanding deferred constraints ** when this statement is run.
76786. Next value in position list
76787. Rollback any active transaction and free the handle structure. ** The call to sqlite3BtreeRollback() drops any table-locks held by ** this handle.
76788. ***** Begin file fts5.c *****
76789. If the p5 flag is clear, then recursive invocation of triggers is ** disabled for backwards compatibility (p5 is set if this sub-program ** is really a trigger, not a foreign key action, and the flag set ** and cleared by the "PRAGMA recursive_triggers" command is clear). *** It is recursive invocation of triggers, at the SQL level, that is ** disabled. In some cases a single trigger may generate more than one ** SubProgram (if the trigger may be executed with more than one different ** ON CONFLICT algorithm). SubProgram structures associated with a ** single trigger all have the same value for the SubProgram.token ** variable.
76790. (280) trans_opt ::= TRANSACTION nm
76791. User authentication failed
76792. If not NULL, blob handle to read node
76793. Where to send the data
76794. Initial working space
76795. Commit the transaction to the *-oal file.
76796. Number of entries allocated in a[] below
76797. The table being inserted or updated
76798. Verify ok to add new elements
76799. Delete the corresponding entry in the <rtree>_rowid table.
76800. Buffer used to build up %_docszie blob
76801. ** Generate code to drop a table.
76802. Add cells to the start of the page
76803. EVIDENCE-OF: R-14606-31564 Value is a BLOB that is (N-12)/2 bytes in ** length. ** EVIDENCE-OF: R-28401-00140 Value is a string in the text encoding and ** (N-13)/2 bytes in length.
76804. Zero the entries in the aPnfo array that correspond to frames with ** frame numbers greater than pWal->hdr.mxFrame.
76805. If this is an in-memory db, or no pages have been written to, or this ** function has already been called, it is mostly a no-op. However, any ** backup in progress needs to be restarted.
76806. 0 1 2 3 4 5 6 7 8 9
76807. ** Given a node number iNode, return the corresponding key to use ** in the Rtree.aHash table.
76808. ** A single instruction of the virtual machine has an opcode ** and as many as three operands. The instruction is recorded ** as an instance of the following structure:
76809. Number of imm. FK constraints this VM
76810. **comment:** ** Argument zFmt is a printf() style format string. This function performs ** the printf() style processing, then appends the results to buffer pBuf. *** Like sqlite3Fts5BufferAppendString(), this function ensures that the byte ** following the buffer data is set to 0x00, even though this byte is not ** included in the pBuf->n count.
76811. Blob value for "root" field
76812. ** Implementation of highlight() function.
76813. ** Create a hash table, free a hash table.
76814. ** 2004 April 13 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains routines used to translate between UTF-8, **
UTF-16, UTF-16BE, and UTF-16LE. *** Notes on UTF-8: ** Byte-0 Byte-1 Byte-2 Byte-3 Value ** 0xxxxxxxxx 00000000 00000000 0xxxxxxx ** 110yyyyy
10xxxxxxxxx 00000000 000000yy yyxxxxxxxxx ** 1110zzzz 10yyyyyy 10xxxxxxxxx 00000000 zzzzzzzz yyxxxxxxxxx ** 11110uuu 10uuuzzz 10yyyyyy 10xxxxxxxxx 000uuuuu
zzzzzzzz yyxxxxxxxxx ** *** Notes on UTF-16: (with www+1==uuuuu) *** Word-0 Word-1 Value ** 110110ww wwzzzzzy 110111yy yyxxxxxxxxx 000uuuuu
zzzzzzzz yyxxxxxxxxx ** zzzzzzzz yyxxxxxxxxx 00000000 zzzzzzzz yyxxxxxxxxx *** *** BOM or Byte Order Mark: ** 0xff 0xfe little-endian utf-16 follows ** 0xfe
0xff big-endian utf-16 follows **
76815. Number of columns forming the key
76816. For ON UPDATE, construct the next term of the WHEN clause. ** The final WHEN clause will be like this: *** WHEN NOT(old.col1 IS new.col1 AND ...
AND old.colN IS new.colN)
76817. Callback to invoke
76818. P4 is a pointer to an Expr tree
76819. Analyze all of the subexpressions.
76820. If some other connection is holding an exclusive lock, the ** requested lock may not be obtained.
76821. If the current block is not empty, and if adding this term/doclist ** to the current block would make it larger than Fts3Table.nNodeSize ** bytes, write this block out to the database.
76822. Last result returned from the iterator
76823. OE_Abort for IMMEDIATE, OE_Rollback for DEFERRED
76824. SELECT name ... WHERE rootpage = \$tnum
76825. **comment:** ** CAPI3REF: Synchronization Type Flags *** When SQLite invokes the xSync() method of an ** [sqlite3_io_methods] object it uses a combination of ** these integer values as the second argument. *** When the SQLITE_SYNC_DATAONLY flag is used, it means that the ** sync operation only needs to flush data to mass storage. Inode ** information need not be flushed. If the lower four bits of the flag ** equal SQLITE_SYNC_NORMAL, that means to use normal fsync() semantics. ** If the lower four bits equal SQLITE_SYNC_FULL, that means ** to use Mac OS X style fullsync instead of fsync(). ** ** Do not confuse the SQLITE_SYNC_NORMAL and SQLITE_SYNC_FULL flags ** with the [PRAGMA synchronous]=NORMAL and [PRAGMA synchronous]=FULL ** settings. The [synchronous pragma] determines when calls to the ** xSync VFS method occur and applies uniformly across all platforms. ** The SQLITE_SYNC_NORMAL and SQLITE_SYNC_FULL flags determine how ** energetic or rigorous or forceful the sync operations are and ** only make a difference on Mac OSX for the default SQLite code. ** (Third-party VFS implementations might also make the distinction ** between SQLITE_SYNC_NORMAL and SQLITE_SYNC_FULL, but among the ** operating systems natively supported by SQLite, only Mac OSX ** cares about the difference.)
76826. ** Read keys from pIncr->pMerger and populate pIncr->aFile[1]. The format ** of the data stored in aFile[1] is the same as that used by regular PMAs, ** except that the number-of-bytes varint is omitted from the start.
76827. Number of characters in function name
76828. Code the current SELECT statement
76829. Errors are detected by individual opcodes, with an immediate ** jumps to abort_due_to_error.
76830. Number of constraints in p
76831. ** The expression pExpr passed as the second argument to this function ** must be of type FTSQUERY_PHRASE. *** The returned value is either NULL or a pointer to a buffer containing ** a position-list indicating the occurrences of the phrase in column iCol ** of the current row. *** More specifically, the returned buffer contains 1 varint for each ** occurrence of the phrase in the column, stored using the normal (delta+2) ** compression and is terminated by either an 0x01 or 0x00 byte. For example, ** if the requested column contains "a b X c d X X" and the position-list ** for 'X' is requested, the buffer returned may contain: *** ** 0x04 0x05 0x03 0x01 or 0x04 0x05 0x03 0x00 *** This function works regardless of whether or not the phrase is deferred, ** incremental, or neither.

76832. ** Close a file. ** ** It is reported that an attempt to close a handle might sometimes ** fail. This is a very unreasonable result, but Windows is notorious ** for being unreasonable so I do not doubt that it might happen. If ** the close fails, we pause for 100 milliseconds and try again. As ** many as MX_CLOSE_ATTEMPT attempts to close the handle are made before ** giving up and returning an error.

76833. ** Set node pNode, which is part of expression pExpr, to point to the first ** match. If there are no matches, set the Node.bEof flag to indicate EOF. ** ** Return an SQLite error code if an error occurs, or SQLITE_OK otherwise. ** It is not an error if there are no matches.

76834. Patch up the ORDER BY clause

76835. Load the FTS index structure

76836. Subtype for this value

76837. ** Compute the soundex encoding of a word. ** ** IMP: R-59782-00072 The soundex(X) function returns a string that is the ** soundex encoding of the string X.

76838. 1320

76839. ** 2008 August 05 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This header file defines the interface that the sqlite page cache ** subsystem.

76840. with ::= WITH wqlist

76841. ** Return the sum of the reference counts for all pages held by pPager.

76842. Overflow chain ends prematurely

76843. Maybe another cursor on the same btree

76844. Number of bytes to lock

76845. ** Walk an expression tree. Return non-zero if the expression is constant ** or a function call with constant arguments. Return and 0 if there ** are any variables. ** ** For the purposes of this function, a double-quoted string (ex: "abc") ** is considered a variable but a single-quoted string (ex: 'abc') is ** a constant.

76846. ** Search for a term in the WHERE clause that is of the form "X <op> <expr>" ** where X is a reference to the iColumn of table iCur or of index pIdx ** if pIdx!=0 and <op> is one of the WO_xx operator codes specified by ** the op parameter. Return a pointer to the term. Return 0 if not found. ** ** If pIdx!=0 then it must be one of the indexes of table iCur. ** Search for terms matching the iColumn-th column of pIdx ** rather than the iColumn-th column of table iCur. ** ** The term returned might be Y=<expr> if there is another constraint in ** the WHERE clause that specifies that X=Y. Any such constraints will be ** identified by the WO_EQUIV bit in the pTerm->eOperator field. The ** aiCur[]/aiColumn[] arrays hold X and all its equivalents. There are 11 ** slots in aiCur[]/aiColumn[] so that means we can look for X plus up to 10 ** other equivalent values. Hence a search for X will return <expr> if X=A1 ** and A1=A2 and A2=A3 and ... and A9=A10 and A10=<expr>. ** ** If there are multiple terms in the WHERE clause of the form "X <op> <expr>" ** then try for the one with no dependencies on <expr> - in other words where ** <expr> is a constant expression of some kind. Only return entries of ** the form "X <op> Y" where Y is a column in another table if no terms of ** the form "X <op> <const-exp>" exist. If no terms with a constant RHS ** exist, try to return a term that does not use WO_EQUIV.

76847. ** Return the bitwise-OR of all Expr.flags fields in the given ** ExprList.

76848. Destination page size

76849. Number of bytes in buffer aRecord[]

76850. ** Move the open database page pDbPage to location iFreePage in the ** database. The pDbPage reference remains valid. ** ** The isCommit flag indicates that there is no need to remember that ** the journal needs to be sync()ed before database page pDbPage->pgno ** can be written to. The caller has already promised not to write to that ** page.

76851. ** This function is a no-op unless currently processing an EXPLAIN QUERY PLAN ** command, or if either SQLITE_DEBUG or SQLITE_ENABLE_STMT_SCANSTATUS was ** defined at compile-time. If it is not a no-op, a single OP_Explain opcode ** is added to the output to describe the table scan strategy in pLevel. ** ** If an OP_Explain opcode is added to the VM, its address is returned. ** Otherwise, if no OP_Explain is coded, zero is returned.

76852. Total number of calls to malloc

76853. ** Same as above, except that this implementation works for WinRT.

76854. Index in mem3.aPool[] of previous free chunk

76855. Most recent error message

76856. Bytes in pPage beyond the header

76857. **comment:** ** CAPI3REF: Obtain The Current Operation From A Changeset Iterator ** ** The pIter argument passed to this function may either be an iterator ** passed to a conflict-handler by [sqlite3changeset_apply()], or an iterator ** created by [sqlite3changeset_start()]. In the latter case, the most recent ** call to [sqlite3changeset_next()] must have returned [SQLITE_ROW]. If this ** is not the case, this function returns [SQLITE_MISUSE]. ** ** If argument pzTab is not NULL, then *pzTab is set to point to a ** nul-terminated utf-8 encoded string containing the name of the table ** affected by the current change. The buffer remains valid until either ** sqlite3changeset_next() is called on the iterator or until the ** conflict-handler function returns. If pnCol is not NULL, then *pnCol is ** set to the number of columns in the table affected by the change. If ** pbIncorrect is not NULL, then *pbIndirect is set to true (1) if the change ** is an indirect change, or false (0) otherwise. See the documentation for ** [sqlite3session_indirect()] for a description of direct and indirect ** changes. Finally, if pOp is not NULL, then *pOp is set to one of ** [SQLITE_INSERT], [SQLITE_DELETE] or [SQLITE_UPDATE], depending on the ** type of change that the iterator currently points to. ** ** If no error occurs, SQLITE_OK is returned. If an error does occur, an ** SQLite error code is returned. The values of the output variables may not ** be trusted in this case.

label: code-design

76858. %'

76859. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, append the string zStr enclosed in quotes ("") and ** with any embedded quote characters escaped to the buffer. No ** nul-terminator byte is written. ** ** If an OOM condition is encountered, set *pRc to SQLITE_NOMEM before ** returning.

76860. Not always defined in the headers as it ought to be

76861. RHS table

76862. **comment:** ** Allocate and zero the pIter->azTblCol[] and abTblPk[] arrays so that ** there is room for at least nCol elements. If an OOM occurs, store an ** error code in the RBU handle passed as the first argument.

label: code-design

76863. Number of objects currently in use

76864. Magic value read from WAL header

76865. First page of the sector pPg is located on.

76866. Statement used to fetch segdir

76867. Allocate the hash table if it has not already been allocated

76868. The opcode

76869. The score for a perfect match

76870. MATCH

76871. **comment:** Avoid a warning indicating that sqlite3Fts5ParserTrace() is unused

label: code-design

76872. nearset ::= colset COLON nearset

76873. (307) likeop ::= LIKE_KW|MATCH

76874. If the WAL file is not empty, return the number of bytes of content ** stored in each frame (i.e. the db page-size when the WAL was created).

76875. Locate the table into which we will be inserting new information.

76876. 1 to honor DESC in index. 0 to ignore.

76877. ***** Begin file walker.c *****

76878. Formulate the zContentExprlist text

76879. Number of overflow cell bodies in aCell[]

76880. nexprlist ::= expr

76881. Number of bytes of memory available to this allocator

76882. New term to write

76883. **comment:** FIXME: Be smarter about indexes that use expressions

label: code-design

76884. ** Return the N-th compile-time option string. If N is out of range, ** return a NULL pointer.
76885. ** Update the value of the change-counter at offsets 24 and 92 in ** the header and the sqlite version number at offset 96. *** This is an unconditional update. See also the pager_incr_changecounter() ** routine which only updates the change-counter if the update is actually ** needed, as determined by the pPager->changeCountDone state variable.
76886. Stop when row count reaches this
76887. ** This function is called to close the connection to the log file prior ** to switching from WAL to rollback mode. *** Before closing the log file, this function attempts to take an ** EXCLUSIVE lock on the database file. If this cannot be obtained, an ** error (SQLITE_BUSY) is returned and the log connection is not closed. ** If successful, the EXCLUSIVE lock is not released before returning.
76888. Force schema load before running
76889. ** Return some kind of integer value which is the best we can do ** at representing the value that *pMem describes as an integer. ** If pMem is an integer, then the value is exact. If pMem is ** a floating-point then the value returned is the integer part. ** If pMem is a string or blob, then we make an attempt to convert ** it into an integer and return that. If pMem represents an ** an SQL-NULL value, return 0. *** If pMem represents a string value, its encoding might be changed.
76890. Opcode: IfSmaller P1 P2 P3 * * * * * Estimate the number of rows in the table P1. Jump to P2 if that ** estimate is less than approximately $2^{**}(0.1*P3)$.
76891. Change counter
76892. p->iCoord might point to either a lower or upper bound coordinate ** in a coordinate pair. But make pCellData point to the lower bound.
76893. The FROM is a subquery
76894. ** Named semaphore locking is only available on VxWorks. ** ***** End of the named semaphore lock implementation *****

76895. The expression to be analyzed.
76896. Initialize the memory allocator
76897. Check btree cell sizes on load
76898. If !=0, WAL frame to return data from
76899. Number of elements in aSavepoint[]
76900. same as TK_GE, jump, in1, in3
76901. Advance each level to the last entry on the last page
76902. ** Insert a new value into a RowSet. ** ** The mallocFailed flag of the database connection is set if a ** memory allocation fails.
76903. A foreign key constraint
76904. List of tables
76905. Cookie value to store
76906. Backup handle
76907. ***** Include fts3.h in the middle of main.c *****
76908. An error message
76909. ** This object is the header on a single record while that record is being ** held in memory and prior to being written out as part of a PMA. ** ** How the linked list is connected depends on how memory is being managed ** by this module. If using a separate allocation for each in-memory record ** (VdbeSorter.list.aMemory==0), then the list is always connected using the ** SorterRecord.u.pNext pointers. ** ** Or, if using the single large allocation method (VdbeSorter.list.aMemory!=0), ** then while records are being accumulated the list is linked using the ** SorterRecord.u.iNext offset. This is because the aMemory[] array may ** be sqlite3Realloc(ed while records are being accumulated. Once the VM ** has finished passing records to the sorter, or when the in-memory buffer ** is full, the list is sorted. As part of the sorting process, it is ** converted to use the SorterRecord.u.pNext pointers. See function ** vdbeSorterSort() for details.
76910. OUT: Buffer containing changeset
76911. Figure out the db that the trigger will be created in
76912. Return a pointer to the allocated object to the caller. Note that ** only the pointer (the 2nd parameter) matters. The size of the object ** (given by the 3rd parameter) is never used and can be any positive ** value.
76913. ** This function creates the second imposter table used when writing to ** a table b-tree where the table has an external primary key. If the ** iterator passed as the second argument does not currently point to ** a table (not index) with an external primary key, this function is a ** no-op. ** ** Assuming the iterator does point to a table with an external PK, this ** function creates a WITHOUT ROWID imposter table named "rbu_impostor2" ** used to access that PK index. For example, if the target table is ** declared as follows: ** ** CREATE TABLE t1(a, b TEXT, c REAL, PRIMARY KEY(b, c); ** ** then the imposter table schema is: ** ** CREATE TABLE rbu_impostor2(c1 TEXT, c2 REAL, id INTEGER) WITHOUT ROWID; **
76914. Top of a loop checking foreign keys
76915. Error message codes for OP_Halt
76916. ** Object used to iterate through all "coalesced phrase instances" in ** a single column of the current row. If the phrase instances in the ** column being considered do not overlap, this object simply iterates ** through them. Or, if they do overlap (share one or more tokens in ** common), each set of overlapping instances is treated as a single ** match. See documentation for the highlight() auxiliary function for ** details. ** ** Usage is: ** ** for(rc = fts5CInstIterNext(pApi, pFts, iCol, &iter); ** (rc==SQLITE_OK && 0==fts5CInstIterEof(&iter); ** rc = fts5CInstIterNext(&iter) **){ ** printf("instance starts at %d, ends at %d\n", iter.iStart, iter.iEnd); ** } **
76917. ***** Begin file vdbesort.c *****
76918. Reload the schema of the modified table.
76919. getenv("TMP")
76920. **comment:** ** CAPI3REF: Prepared Statement Scan Status ** METHOD: sqlite3_stmt ** ** This interface returns information about the predicted and measured ** performance for pStmt. Advanced applications can use this ** interface to compare the predicted and the measured performance and ** issue warnings and/or rerun [ANALYZE] if discrepancies are found. ** ** Since this interface is expected to be rarely used, it is only ** available if SQLite is compiled using the [SQLITE_ENABLE_STMT_SCANSTATUS] ** compile-time option. ** ** The "iScanStatusOp" parameter determines which status information to return. ** The "iScanStatusOp" must be one of the [scanstatus options] or the behavior ** of this interface is undefined. ** ^The requested measurement is written into a variable pointed to by ** the "pOut" parameter. ** Parameter "idx" identifies the specific loop to retrieve statistics for. ** Loops are numbered starting from zero. ^If idx is out of range - less than ** zero or greater than or equal to the total number of loops used to implement ** the statement - a non-zero value is returned and the variable that pOut ** points to is unchanged. ** ** ^Statistics might not be available for all loops in all statements. ^In cases ** where there exist loops with no available statistics, this function behaves ** as if the loop did not exist - it returns non-zero and leave the variable ** that pOut points to unchanged. ** ** See also: [sqlite3_stmt_scanstatus_reset()]
label: code-design
76921. Allocate space for Fts3Cursor.aMatchinfo[] and Fts3Cursor.zMatchinfo.
76922. Number of iterators/phrases
76923. The HAVING clause
76924. ** Convert raw bits from the on-disk RTree record into a coordinate value. ** The on-disk format is big-endian and needs to be converted for little- ** endian platforms. The on-disk record stores integer coordinates if ** eInt is true and it stores 32-bit floating point records if eInt is ** false. a[] is the four bytes of the on-disk record to be decoded. ** Store the results in "r". ** ** There are five versions of this macro. The last one is generic. The ** other four are various architectures-specific optimizations.
76925. Round away from zero
76926. ***** End of fts3Int.h *****
76927. 540
76928. Sanity checking on other operands
76929. Current position list
76930. ** An implementation of the LockFile() API of Windows for CE
76931. free the system buffer allocated by FormatMessage
76932. Allocate space for aXRef[], aRegIdx[], and aToOpen[]. ** Initialize aXRef[] and aToOpen[] to their default values.
76933. ** json_extract(JSON, PATH, ...) ** ** Return the element described by PATH. Return NULL if there is no ** PATH element. If there are multiple PATHs, then return a JSON array ** with the result from each path. Throw an error if the JSON or any PATH ** is malformed.
76934. Number of conch taking failures
76935. WhereLoop.u.btree.pIndex is valid

76936. Extra zero bytes when MEM_Zero and MEM_Blob set
76937. 1040
76938. ** If the word ends with zFrom and xCond() is true for the stem ** of the word that precedes the zFrom ending, then change the ** ending to zTo. *** The input word *pz and zFrom are both in reverse order. zTo ** is in normal order. *** Return TRUE if zFrom matches. Return FALSE if zFrom does not ** match. Not that TRUE is returned even if xCond() fails and ** no substitution occurs.
76939. File offset to begin writing at
76940. **comment:** ** Invoke the xNext method of an Fts5ExprNode object. This macro should be ** used as if it has the same signature as the xNext() methods themselves.
label: code-design
76941. ** This function encodes a single frame header and writes it to a buffer ** supplied by the caller. A frame-header is made up of a series of ** 4-byte big-endian integers, as follows: ** 0: Page number. ** 4: For commit records, the size of the database image in pages ** after the commit. For all other records, zero. ** 8: Salt-1 (copied from the wal-header) ** 12: Salt-2 (copied from the wal-header) ** 16: Checksum-1. ** 20: Checksum-2.
76942. Context used to resolve WHERE clause
76943. Opcode: Sequence P1 P2 * * * * Synopsis: r[P2]=cursor[P1].ctr++ * * * * Find the next available sequence number for cursor P1. ** Write the sequence number into register P2. ** The sequence number on the cursor is incremented after this ** instruction.
76944. Invoke the main loop body as a subroutine
76945. ** The size of the of each page record in the journal is given by ** the following macro.
76946. ** Bitmasks for the operators on WhereTerm objects. These are all ** operators that are of interest to the query planner. An ** OR-ed combination of these values can be used when searching for ** particular WhereTerms within a WhereClause. *** Value constraints: ** WO_EQ == SQLITE_INDEX_CONSTRAINT_EQ ** WO_LT == SQLITE_INDEX_CONSTRAINT_LT ** WO_LE == SQLITE_INDEX_CONSTRAINT_LE ** WO_GT == SQLITE_INDEX_CONSTRAINT_GT ** WO_GE == SQLITE_INDEX_CONSTRAINT_GE ** WO_MATCH == SQLITE_INDEX_CONSTRAINT_MATCH
76947. ** Generate code for a single equality term of the WHERE clause. An equality ** term can be either X=expr or X IN (...). pTerm is the term to be ** coded. *** ** The current value for the constraint is left in a register, the index ** of which is returned. An attempt is made store the result in iTarget but ** this is only guaranteed for TK_ISNULL and TK_IN constraints. If the ** constraint is a TK_EQ or TK_IS, then the current value might be left in ** some other register and it is the caller's responsibility to compensate. *** ** For a constraint of the form X=expr, the expression is evaluated in ** straight-line code. For constraints of the form X IN (...) ** this routine sets up a loop that will iterate over all values of X.
76948. New element added to the pH
76949. 1-byte signed integer
76950. when_clause
76951. There are no pages on the freelist, so append a new page to the ** database image. *** Normally, new pages allocated by this block can be requested from the ** pager layer with the 'no-content' flag set. This prevents the pager ** from trying to read the pages content from disk. However, if the ** current transaction has already run one or more incremental-vacuum ** steps, then the page we are about to allocate may contain content ** that is required in the event of a rollback. In this case, do ** not set the no-content flag. This causes the pager to load and journal ** the current page content before overwriting it. *** Note that the pager will not actually attempt to load or journal ** content for any page that really does lie past the end of the database ** file on disk. So the effects of disabling the no-content optimization ** here are confined to those pages that lie between the end of the ** database image and the end of the database file.
76952. Do not allow memory allocation
76953. String on RHS of LIKE operator
76954. Number of entries in aTC[]
76955. List of values being extracted
76956. assert(db->pPreUpdate->pNewUnpacked || db->pPreUpdate->aNew);
76957. ** Context object for havingToWhereExprCb().
76958. 1280
76959. If Fts3Cursor.pMIBuffer is NULL, then this is the first time the ** matchinfo function has been called for this query. In this case ** allocate the array used to accumulate the matchinfo data and ** initialize those elements that are constant for every row.
76960. The hash table to insert into
76961. If an error occurs, we jump here
76962. OUT: gid of zFile.
76963. The table being constructed
76964. The computed column name
76965. The value returned by this function should always be the same as ** the (CellInfo.nSize) value found by doing a full parse of the ** cell. If SQLITE_DEBUG is defined, an assert() at the bottom of ** this function verifies that this invariant is not violated.
76966. This is the general case where many pages are being removed. ** It is necessary to scan the entire hash table
76967. ** Insert a new cell on pPage at cell index "i". pCell points to the ** content of the cell. *** If the cell content will fit on the page, then put it there. If it ** will not fit, then make a copy of the cell content into pTemp if ** pTemp is not null. Regardless of pTemp, allocate a new entry ** in pPage->apOverflow[] and make it point to the cell content (either ** in pTemp or the original pCell) and also record its index. ** Allocating a new entry in pPage->aCell[] implies that ** pPage->nOverflow is incremented. *** *pRC must be SQLITE_OK when this routine is called.
76968. read-write cursor
76969. ** Arguments nArg/azArg contain the string arguments passed to the xCreate ** or xConnect method of the virtual table. This function attempts to ** allocate an instance of Fts5Config containing the results of parsing ** those arguments. *** If successful, SQLITE_OK is returned and *ppOut is set to point to the ** new Fts5Config object. If an error occurs, an SQLite error code is ** returned, *ppOut is set to NULL and an error message may be left in ** *pzErr. It is the responsibility of the caller to eventually free any ** such error message using sqlite3_free().
76970. At this point, pWLoop is a candidate to be the next loop. ** Compute its cost
76971. typetoken ::=
76972. This function is called when Btree p is concluding its ** transaction. If there currently exists a writer, and p is not ** that writer, then the number of locks held by connections other ** than the writer must be about to drop to zero. In this case ** set the BTS_PENDING flag to 0. *** If there is not currently a writer, then BTS_PENDING must ** be zero already. So this next line is harmless in that case.
76973. Btree page of current entry
76974. ACTION => ID
76975. ** For tokenizers with no "unicode" modifier, the set of token characters ** is the same as the set of ASCII range alphanumeric characters.
76976. OUT: WITH clause return value belongs to
76977. Type of join between this table and the previous
76978. Value to destroy
76979. This file is used on unix only
76980. Linked list of all sub-programs used by VM
76981. Authentication not yet checked
76982. **comment:** True after malloc is initialized
label: code-design
76983. sqlite3_test_control(SQLITE_TESTCTRL_LOCALTIME_FAULT, int onoff); *** If parameter onoff is non-zero, configure the wrappers so that all ** subsequent calls to localtime() and variants fail. If onoff is zero, ** undo this setting.
76984. b8..bf,
76985. Write the name of each database file in the transaction into the new ** master journal file. If an error occurs at this point close ** and delete the master journal file. All the individual journal files ** still have 'null' as the master journal pointer, so they will roll ** back independently if a failure occurs.
76986. Statement used to read %_content
76987. szOsFile
76988. **comment:** ** Cleans up the mapped region of the specified file, if any.
label: code-design
76989. ** Ensure that the Fts5Cursor.nInstCount and aInst[] variables are populated ** correctly for the current view. Return SQLITE_OK if successful, or an ** SQLite error code otherwise.

76990. exprlist
76991. ** Force RBU to save its state to disk. ** If a power failure or application crash occurs during an update, following ** system recovery RBU may resume the update from the point at which the state ** was last saved. In other words, from the most recent successful call to ** sqlite3rbu_close() or this function. ** **
SQLITE_OK is returned if successful, or an SQLite error code otherwise.
76992. Make sure the new file size is written into the inode right away. ** Otherwise the journal might resurrect following a power loss and ** cause the last transaction to roll back. See ** https://bugzilla.mozilla.org/show_bug.cgi?id=1072773
76993. SETUP-INVARIANT above
76994. Store result as keys in an index
76995. ** When testing, also keep a count of the number of open files.
76996. Disable the original
76997. ** Array apCell[] contains nCell pointers to b-tree cells. Array szCell ** contains the size in bytes of each such cell. This function adds the ** space associated with each cell in the array that is currently stored ** within the body of pPg to the pPg free-list. The cell-pointers and other ** fields of the page are not updated.
** ** This function returns the total number of cells added to the free-list.
76998. 4-byte unsigned integer
76999. No table references
77000. Size of pOrig region in bytes
77001. xInit
77002. Currently iOff points to the first byte of a varint. This block ** decrements iOff until it points to the first byte of the previous ** varint. Taking care not to read any memory locations that occur ** before the buffer in memory.
77003. #ifndef SQLITE_OMIT_INCRBLOB
77004. Adjust base and n to skip over SQLITE_AFF_BLOB entries at the beginning ** and end of the affinity string.
77005. ** Implementation of the sqlite3_pcache.xCreate method. ** ** Allocate a new cache.
77006. The root page of this tree
77007. * Make sure that the calculated cache size, in pages, cannot cause the * initial size of the Win32-specific heap to exceed the maximum amount * of memory that can be specified in the call to HeapCreate.
77008. "", "\", or '\"'. String literals, quoted ids
77009. MEM_Dyn may only be set if Mem.szMalloc==0. In this way we ** ensure that if Mem.szMalloc>0 then it is safe to do ** Mem.z = Mem.zMalloc without having to check Mem.flags&MEM_Dyn. ** That saves a few cycles in inner loops.
77010. Hash value to return
77011. The tree being checked out
77012. ** Remove a row from the FTS table.
77013. Holding area for temporary registers
77014. This Mem is a shallow copy of pScopyFrom
77015. sortorder ::=
77016. The value of pExpr->op and op are related as follows: ** ** pExpr->op op ** ----- ** TK_ISNULL OP_NotNull ** TK_NOTNULL OP_IsNull **
TK_NE OP_Eq ** TK_EQ OP_Ne ** TK_GT OP_Le ** TK_LE OP_Gt ** TK_GE OP_Lt ** TK_LT OP_Ge *** For other values of pExpr->op, op is undefined and unused. ** The value of TK_ and OP_ constants are arranged such that we ** can compute the mapping above using the following expression. **
Assert(s) verify that the computation is correct.
77017. The page that contains the cell
77018. Do not allow the upper bound of a LIKE optimization range constraint ** to mix with a lower range bound from some other source
77019. Overflows are sequential
77020. If some thread using this PID has a lock via a different unixFile* ** handle that precludes the requested lock, return BUSY.
77021. If the cursor is already positioned at the point we are trying ** to move to, then just return without doing any work
77022. **comment:** ** CAPI3REF: Suspend Execution For A Short Time ** ** The sqlite3_sleep() function causes the current thread to suspend execution ** for at least a number of milliseconds specified in its parameter. ** ** If the operating system does not support sleep requests with ** millisecond time resolution, then the time will be rounded up to ** the nearest second. The number of milliseconds of sleep actually ** requested from the operating system is returned. ** ** ^SQLite implements this interface by calling the xSleep() ** method of the default [sqlite3_vfs] object. If the xSleep() method ** of the default VFS is not implemented correctly, or not implemented at ** all, then the behavior of sqlite3_sleep() may deviate from the description ** in the previous paragraphs.
label: code-design
77023. ** Compare the contents of the two buffers using memcmp(). If one buffer ** is a prefix of the other, it is considered the lesser. ** ** Return -ve if pLeft is smaller than pRight, 0 if they are equal or ** +ve if pRight is smaller than pLeft. In other words: ** ** res = *pLeft - *pRight
77024. ** Allocate space to hold a new trigger step. The allocated space ** holds both the TriggerStep object and the TriggerStep.target.z string. ** ** If an OOM error occurs, NULL is returned and db->mallocFailed is set.
77025. OUT: Token text
77026. Name of symbol to add
77027. Name of index or table
77028. What to do with query results
77029. The parsing context. Errors accumulate here
77030. rbu database handle
77031. PmaReader from which to take the blob
77032. True if token must appear at position 0
77033. The temporary database we vacuum into
77034. The sqlite3_aggregate_count() function is deprecated. But just to make ** sure it still operates correctly, verify that its count agrees with our ** internal count when using count(*) and when the total count can be ** expressed as a 32-bit integer.
77035. Write the bytes into this buffer
77036. Virtual machine under construction
77037. Total term size
77038. in2
77039. Translate JSON formatted string into raw text
77040. OUT: Value of original database size field
77041. ** This is the xColumn method, called by SQLite to request a value from ** the row that the supplied cursor currently points to. ** ** If: ** ** (iCol < p->nColumn) -> The value of the iCol'th user column. ** (iCol == p->nColumn) -> Magic column with the same name as the table. ** (iCol == p->nColumn+1) -> Docid column ** (iCol == p->nColumn+2) -> Langid column
77042. Array of flags, set on target PK columns
77043. If a transaction is open, the disconnectAllVtab() call above ** will not have called the xDisconnect() method on any virtual ** tables in the db->aVTrans[] array. The following sqlite3VtabRollback() ** call will do so. We need to do this before the check for active ** SQL statements below, as the v-table implementation may be storing ** some prepared statements internally.
77044. The WAL has been completely backfilled (or it is empty). ** and can be safely ignored.
77045. ***** End of mem2.c *****
77046. Methods above are valid for version 1
77047. OF => ID
77048. a8..af,
77049. The argument values
77050. ** External API function used to create a new virtual-table module.
77051. Nesting depth
77052. synopsis: root=P2 iDb=P3
77053. This is an in-memory DB
77054. Either chngPk or chngRowid

77055. x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xa xb xc xd xe xf
77056. Different WHERE clauses
77057. Result string
77058. **comment:** ** Advance to the next document that matches the FTS expression in ** Fts3Cursor.pExpr.
 label: documentation
77059. pPager->stmtSize = 0;
77060. If SQLITE_NULLEQ is set (which will only happen if the operator is ** OP_Eq or OP_Ne) then take the jump or not depending on whether ** or not both operands are null.
77061. ** This function is only called from within a pre-update handler for a ** write to table pTab, part of session pSession. If this is the first ** write to this table, initialize the SessionTable.nCol, azCol[] and ** abPK[] arrays accordingly. ** ** If an error occurs, an error code is stored in sqlite3_session.rc and ** non-zero returned. Or, if no error occurs but the table has no primary ** key, sqlite3_session.rc is left set to SQLITE_OK and non-zero returned to ** indicate that updates on this table should be ignored. SessionTable.abPK ** is set to NULL in this case.
77062. True for threadsafe connections
77063. OUT: gid to set on the file
77064. If using the onepass strategy, no-op out the OP_OpenEphemeral coded ** above. Also, if this is a top-level parse (not a trigger), clear the ** multi-write flag so that the VM does not open a statement journal
77065. Step 3: Abort if createFlag is 1 but the cache is nearly full
77066. Number of cells to add to pPg
77067. Collating sequence to use on this term
77068. Recursive part w/o aColCache
77069. Used by: busy_timeout
77070. Aggregate column length array
77071. Database lock protocol error
77072. Register holding assembled index record
77073. 94
77074. SELECT rowid FROM x ... (dup of pSrc)
77075. p1 is smaller
77076. Token associated with this expression
77077. 151
77078. FIX ME: Compute pExpr->affinity based on the expected return ** type of the function
77079. Name of the table to be locked
77080. ** Term pTerm is a vector range comparison operation. The first comparison ** in the vector can be optimized using column nEq of the index. This ** function returns the total number of vector elements that can be used ** as part of the range comparison. ** ** For example, if the query is: ** ** WHERE a = ? AND (b, c, d) > (?, ?, ?) ** ** and the index: ** ** CREATE INDEX ... ON (a, b, c, d, e) ** ** then this function would be invoked with nEq=1. The value returned in ** this case is 3.
77081. ** Bitfield flags for P5 value in various opcodes. ** ** Value constraints (enforced via assert()): ** OPFLAG_LENGTHHARG == SQLITE_FUNC_LENGTH ** OPFLAG_TYPEOFARG == SQLITE_FUNC_TYPEOF ** OPFLAG_BULKCSR == BTREE_BULKLOAD ** OPFLAG_SEEKEQ == BTREE_SEEK_EQ ** OPFLAG_FORDELETE == BTREE_FORDELETE ** OPFLAG_SAVEPOSITION == BTREE_SAVEPOSITION ** OPFLAG_AUXDELETE == BTREE_AUXDELETE
77082. same as TK_GT, synopsis: IF r[P3]>r[P1]
77083. Integer value contained in u.iValue
77084. Offset of pgidx in a[]
77085. **comment:** ** Search for an unused file descriptor that was opened on the database ** file (not a journal or master-journal file) identified by pathname ** zPath with SQLITE_OPEN_XXX flags matching those passed as the second ** argument to this function. ** ** Such a file descriptor may exist if a database connection was closed ** but the associated file descriptor could not be closed because some ** other file descriptor open on the same file is holding a file-lock. ** Refer to comments in the unixClose() function and the lengthy comment ** describing "Posix Advisory Locking" at the start of this file for ** further details. Also, ticket #4018. ** ** If a suitable file descriptor is found, then it is returned. If no ** such file descriptor is located, -1 is returned.
 label: code-design
77086. Column number of LHS
77087. Current 'start' value
77088. ** This function is used to add page iPage to the database file free-list. ** It is assumed that the page is not already a part of the free-list. ** ** The value passed as the second argument to this function is optional. ** If the caller happens to have a pointer to the MemPage object ** corresponding to page iPage handy, it may pass it as the second value. ** Otherwise, it may pass NULL. ** ** If a pointer to a MemPage object is passed as the second argument, ** its reference count is not altered by this function.
77089. ** Construct a WalIterator object that can be used to loop over all ** pages in the WAL in ascending order. The caller must hold the checkpoint ** lock. ** ** On success, make *pp point to the newly allocated WalIterator object ** return SQLITE_OK. Otherwise, return an error code. If this routine ** returns an error, the value of *pp is undefined. ** ** The calling routine should invoke walIteratorFree() to destroy the ** WalIterator object when it has finished with it.
77090. If the P4 argument is not NULL, then it must be an SQL comment string. ** The "-" string is broken up to prevent false-positives with srcck1.c. ** ** This assert() provides evidence for: ** EVIDENCE-OF: R-50676-09860 The callback can compute the same text that ** would have been returned by the legacy sqlite3_trace() interface by ** using the X argument when X begins with "--" and invoking ** sqlite3_expanded_sql(P) otherwise.
77091. ** Return the appropriate collating sequence for the iCol-th column of ** the result set for the compound-select statement "p". Return NULL if ** the column has no default collating sequence. ** ** The collating sequence for the compound select is taken from the ** left-most term of the select that has a collating sequence.
77092. An error has occurred. Delete the hash table and return the error code.
77093. If the "=MODE" part does not match any known journal mode, ** then do a query
77094. The expression is a column. Locate the table the column is being ** extracted from in NameContext.pSrcList. This table may be real ** database table or a subquery.
77095. Assert that P3 is a valid memory cell.
77096. Name of the table to be updated
77097. ** Sometimes we need a small amount of code such as a variable initialization ** to setup for a later assert() statement. We do not want this code to ** appear when assert() is disabled. The following macro is therefore ** used to contain that setup code. The "VVA" acronym stands for ** "Verification, Validation, and Accreditation". In other words, the ** code within VVA_ONLY() will only run during verification processes.
77098. IMPLEMENTATION-OF: R-35210-63508 The sqlite3_libversion_number() function ** returns an integer equal to SQLITE_VERSION_NUMBER.
77099. ** Hash table for global functions - functions common to all ** database connections. After initialization, this table is ** read-only.
77100. Address register for select-B coroutine
77101. **comment:** Increment Parse.nHeight by the height of the largest expression ** tree referred to by this, the parent select. The child select ** may contain expression trees of at most ** (SQLITE_MAX_EXPR_DEPTH-Parse.nHeight) height. This is a bit ** more conservative than necessary, but much easier than enforcing ** an exact limit.
 label: code-design
77102. New table object (if required)
77103. out3: P3 is an output
77104. Child table contain "REFERENCES" keyword
77105. Make this point to any syntax error
77106. Do not defer seeks if unique
77107. Used by: foreign_key_list
77108. ** Release an SQLite statement handle obtained via an earlier call to ** sqlite3Fts5StorageStmt(). The eStmt parameter passed to this function ** must match that passed to the sqlite3Fts5StorageStmt() call.
77109. 500

77110. Reload the constraint value into reg[iReg+j+2]. The same value ** was loaded into the same register prior to the OP_VFilter, but ** the xFilter implementation might have changed the datatype or ** encoding of the value in the register, so it *must* be reloaded.

77111. 4-byte signed integer

77112. End of function

77113. True if REPLACE is used to resolve INT PK conflict

77114. *** pCurrent gets an accurate estimate of the amount of memory used ** to store the schema for all databases (main, temp, and any ATTACHED ** databases. *pHighwater is set to zero.

77115. ** Macros for testing whether or not optimizations are enabled or disabled.

77116. OUT: Return the Pager structure here

77117. Check if any child key columns are being modified.

77118. How to dispose of the results. On of SRT_* above.

77119. (308) exprlist ::= nexprlist

77120. Parent class. MUST BE FIRST

77121. If this is a prefix query, check that the results returned if the ** the index is disabled are the same. In both ASC and DESC order. *** This check may only be performed if the hash table is empty. This ** is because the hash table only supports a single scan query at ** a time, and the multi-iter loop from which this function is called ** is already performing such a scan.

77122. Used by sqlite3WhereExprUsage()

77123. True if an equality comparison has been seen

77124. ** CAPI3REF: Data Change Notification Callbacks ** METHOD: sqlite3 *** ^The sqlite3_update_hook() interface registers a callback function ** with the [database connection] identified by the first argument ** to be invoked whenever a row is updated, inserted or deleted in ** a [rowid table]. ** ^Any callback set by a previous call to this function ** for the same database connection is overridden. *** ^The second argument is a pointer to the function to invoke when a ** row is updated, inserted or deleted in a rowid table. ** ^The first argument to the callback is a copy of the third argument ** to sqlite3_update_hook(). ** ^The second callback argument is one of [SQLITE_INSERT], [SQLITE_DELETE], ** or [SQLITE_UPDATE], depending on the operation that caused the callback ** to be invoked. ** ^The third and fourth arguments to the callback contain pointers to the ** database and table name containing the affected row. ** ^The final callback parameter is the [rowid] of the row. ** ^In the case of an update, this is the [rowid] after the update takes place. *** ^^(The update hook is not invoked when internal system tables are ** modified (i.e. sqlite_master and sqlite_sequence).) ** ^The update hook is not invoked when [WITHOUT ROWID] tables are modified. *** ^In the current implementation, the update hook ** is not invoked when conflicting rows are deleted because of an ** [ON CONFLICT | ON CONFLICT REPLACE] clause. ^Nor is the update hook ** invoked when rows are deleted using the [truncate optimization]. ** The exceptions defined in this paragraph might change in a future ** release of SQLite. *** ^The update hook implementation must not do anything that will modify ** the database connection that invoked the update hook. Any actions ** to modify the database connection must be deferred until after the ** completion of the [sqlite3_step()] call that triggered the update hook. ** Note that [sqlite3_prepare_v2()] and [sqlite3_step()] both modify their ** database connections for the meaning of "modify" in this paragraph. *** ^The sqlite3_update_hook(D,C,P) function ** returns the P argument from the previous call ** on the same [database connection] D, or NULL for ** the first call on D. ** ** See also the [sqlite3_commit_hook()], [sqlite3_rollback_hook()], ** and [sqlite3_prewrite_hook()] interfaces.

77125. ** Return the name of the database from which a result column derives. ** NULL is returned if the result column is an expression or constant or ** anything else which is not an unambiguous reference to a database column.

77126. True if sqlite3_interrupt has been called

77127. Validate index entries for the current row

77128. Round nByte up to the next valid power of two

77129. Writer context

77130. ** Callback for tokenizing terms used by ParseTerm().

77131. Number of terms in the ORDER BY clause

77132. **comment:** sqlite3_test_control(SQLITE_TESTCTRL_VDBE_COVERAGE, xCallback, ptr); *** Set the Vdbe coverage callback function to xCallback with context ** pointer ptr.
label: test

77133. SELECT statement in which to make substitutions

77134. ERROR: insert count exceeds size of delta

77135. An automatic index created by a PRIMARY KEY or UNIQUE constraint

77136. Used by: database_list

77137. ** Decrement the reference count on a vxworksFileId object. Free ** the object when the reference count reaches zero.

77138. ***** Begin file status.c *****

77139. If in full-sync mode, advance to the next disk sector before writing ** the master journal name. This is in case the previous page written to ** the journal has already been synced.

77140. The maximum number of cells on a single page of the database. This ** assumes a minimum cell size of 6 bytes (4 bytes for the cell itself ** plus 2 bytes for the index to the cell in the page header). Such ** small cells will be rare, but they are possible.

77141. Index being probed

77142. Information about each format field

77143. Size of allocation used for *p

77144. List to which to append. Might be NULL

77145. 8

77146. String to quote, escape and append

77147. xDeviceCapabilities

77148. The write-ahead log

77149. ** Check to see if column iCol of the given statement is valid. If ** it is, return a pointer to the Mem for the value of that column. ** If iCol is not valid, return a pointer to a Mem which has a value ** of NULL.

77150. ***** End %stack_overflow code *****

77151. Try to truncate the WAL file to zero bytes if the checkpoint ** completed and fsynced (rc==SQLITE_OK) and we are in persistent ** WAL mode (bPersist) and if the PRAGMA journal_size_limit is a ** non-negative value (pWal->mxWalSize>=0). Note that we truncate ** to zero bytes as truncating to the journal_size_limit might ** leave a corrupt WAL file on disk.

77152. True for statements that read

77153. The specific comparison operator

77154. Column on LHS of MATCH operator

77155. Blob I/O on xxx_node

77156. Before populating the accumulator registers, clear the column cache. ** Otherwise, if any of the required column values are already present ** in registers, sqlite3ExprCode() may use OP_SCopy to copy the value ** to pC->iMem. But by the time the value is used, the original register ** may have been used, invalidating the underlying buffer holding the ** text or blob value. See ticket [883034dc5]. *** Another solution would be to change the OP_SCopy used to copy cached ** values to an OP_Copy.

77157. VIRTUAL => ID

77158. 710

77159. 175

77160. Free any existing lookaside buffer for this handle before ** allocating a new one so we don't have to have space for ** both at the same time.

77161. ** This function is called to rollback or release (commit) a savepoint. ** The savepoint to release or rollback need not be the most recently ** created savepoint. *** Parameter op is always either SAVEPOINT_ROLLBACK or SAVEPOINT_RELEASE. ** If it is SAVEPOINT_RELEASE, then release and destroy the savepoint with ** index iSavepoint. If it is SAVEPOINT_ROLLBACK, then rollback all changes ** that have occurred since the specified savepoint was created. *** The savepoint to rollback or release is identified by parameter ** iSavepoint. A value of 0 means to operate on the outermost savepoint ** (the first created). A value of (Pager.nSavepoint-1) means operate ** on the most recently created savepoint. If iSavepoint is greater than ** (Pager.nSavepoint-1), then this function is a no-op. ** ** If a negative value is passed to this function, then the current ** transaction is rolled back. This is different to calling ** sqlite3PagerRollback() because this function does not terminate ** the transaction or unlock the database, it just restores the ** contents of the database to its original state. *** In any case, all savepoints with an index greater than iSavepoint ** are destroyed. If this is a release operation (op==SAVEPOINT_RELEASE), ** then savepoint

iSavepoint is also destroyed. *** This function may return SQLITE_NOMEM if a memory allocation fails, ** or an IO error code if an IO error occurs while rolling back a ** savepoint. If no errors occur, SQLITE_OK is returned.

77162. The table in which we should change things

77163. **comment:** ** Extract the smallest element from the RowSet. ** Write the element into *pRowid. Return 1 on success. Return ** 0 if the RowSet is already empty. *** After this routine has been called, the sqlite3RowSetInsert() ** routine may not be called again. *** This routine may not be called after sqlite3RowSetTest() has ** been used. Older versions of RowSet allowed that, but as the ** capability was not used by the code generator, it was removed ** for code economy.

label: code-design

77164. Restriction (6)

77165. ** If the error code currently stored in the RBU handle is SQLITE_CONSTRAINT, ** then edit any error message string so as to remove all occurrences of ** the pattern "rbu_imp_[0-9]*".

77166. ** Functions called by the storage module as part of integrity-check.

77167. SHM region size

77168. ** A version of vdbeSorterCompare() that assumes that it has already been ** determined that the first field of key1 is equal to the first field of ** key2.

77169. ** This is common tail processing for btreeParseCellPtr() and ** btreeParseCellPtrIndex() for the case when the cell does not fit entirely ** on a single B-tree page. Make necessary adjustments to the CellInfo ** structure.

77170. ***** End of fts3_porter.c *****

77171. Array of rootpage numbers for tables to be checked

77172. Figure out how much space to allocate

77173. OUT: Pointer to doclist for pTerm

77174. ** Sort the elements in list aList using aContent[] as the sort key. ** Remove elements with duplicate keys, preferring to keep the ** larger aList[] values. *** The aList[] entries are indices into aContent[]. The values in ** aList[] are to be sorted so that for all J<K: ** aContent[aList[J]] < aContent[aList[K]] *** For any X and Y such that ** aContent[aList[X]] == aContent[aList[Y]] ** Keep the larger of the two values aList[X] and aList[Y] and discard ** the smaller.

77175. ** Return a pointer to an sqlite3VdbeRecordCompare() compatible function ** suitable for comparing serialized records to the unpacked record passed ** as the only argument.

77176. ** Close a cache.

77177. No other bits set

77178. Selected leaf node

77179. Number of segments that share a docid

77180. If the file on disk is smaller than the database image, use ** pager_truncate to grow the file here. This can happen if the database ** image was extended as part of the current transaction and then the ** last page in the db image moved to the free-list. In this case the ** last page is never written out to disk, leaving the database file ** undersized. Fix this now if it is the case.

77181. ** 2010 February 23 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. *** ***** This file implements routines used to report what compile-time options ** SQLite was built with.

77182. Not currently locked

77183. No user-visible "rowid" column

77184. ** If pMem is an object with a valid string representation, this routine ** ensures the internal encoding for the string representation is ** 'desiredEnc', one of SQLITE_UTF8, SQLITE_UTF16LE or SQLITE_UTF16BE. *** If pMem is not a string object, or the encoding of the string ** representation is already stored using the requested encoding, then this ** routine is a no-op. ** SQLITE_OK is returned if the conversion is successful (or not required). ** SQLITE_NOMEM may be returned if a malloc() fails during conversion ** between formats.

77185. Name of the shared library containing extension

77186. If we get this far, it means that the xfer optimization is at ** least a possibility, though it might only work if the destination ** table (tab1) is initially empty.

77187. Swap the roles of aFrom and aTo for the next generation

77188. The following code adds nothing to the actual functionality ** of the program. It is only here for testing and debugging. ** On the other hand, it does burn CPU cycles every time through ** the evaluator loop. So we can leave it out when NDEBUG is defined.

77189. Make sure the SQLITE_OPEN_URI flag is set to indicate to the VFS xOpen ** method that there may be extra parameters following the file-name.

77190. **comment:** ** For MinGW, check to see if we can include the header file containing its ** version information, among other things. Normally, this internal MinGW ** header file would [only] be included automatically by other MinGW header ** files; however, the contained version information is now required by this ** header file to work around binary compatibility issues (see below) and ** this is the only known way to reliably obtain it. This entire #if block ** would be completely unnecessary if there was any other way of detecting ** MinGW via their preprocessor (e.g. if they customized their GCC to define ** some MinGW-specific macros). When compiling for MinGW, either the ** _HAVE_MINGW_H or _HAVE__MINGW_H (note the extra underscore) macro must be ** defined; otherwise, detection of conditions specific to MinGW will be ** disabled.

label: code-design

77191. SQLITE_MAX_WORKER_THREADS>0

77192. ** unixepoch *** Treat the current value of p->s as the number of ** seconds since 1970. Convert to a real julian day number.

77193. xRekey

77194. Not an aggregate

77195. If it is not already open and the file exists on disk, open the ** journal for read/write access. Write access is required because ** in exclusive-access mode the file descriptor will be kept open ** and possibly used for a transaction later on. Also, write-access ** is usually required to finalize the journal in journal_mode=persist ** mode (and also for journal_mode=truncate on some systems). *** If the journal does not exist, it usually means that some ** other connection managed to get in and roll it back before ** this connection obtained the exclusive lock above. Or, it ** may mean that the pager was in the error-state when this ** function was called and the journal file does not exist.

77196. Set the value of register r[1] in the SQL statement to integer iRow. ** This is done directly as a performance optimization

77197. Read data from here after Rewind()

77198. If the approximation iKey is smaller than the actual real search ** term, substitute <= for < and > for >=.

77199. defined(__APPLE__) && !defined(SQLITE_WITHOUT_ZONEMALLOC)

77200. The Index.aiColumn[] values are normally positive integer. But ** there are some negative values that have special meaning:

77201. A user column. Or, if this is a full-table scan, possibly the ** language-id column. Seek the cursor.

77202. True if sub-query is correlated

77203. Each column used only once

77204. ***** End of the dot-file lock implementation *****

77205. OUT: VFS to use

77206. ** Generate an instruction that will put the floating point ** value described by z[0..n-1] into register iMem. ** ** The z[] string will probably not be zero-terminated. But the ** z[n] character is guaranteed to be something that does not look ** like the continuation of the number.

77207. Set the JsonString object to an empty string

77208. The OP_RowData opcodes always follow OP_NotExists or ** OP_SeekRowid or OP_Rewind/Op_Next with no intervening instructions ** that might invalidate the cursor. ** If this were not the case, one of the following assert()s ** would fail. Should this ever change (because of changes in the code ** generator) then the fix would be to insert a call to ** sqlite3VdbeCursorMoveto().

77209. ** Set the number of backtrace levels kept for each allocation. ** A value of zero turns off backtracing. The number is always rounded ** up to a multiple of 2.

77210. Last memory cell in new array

77211. ** The argument is an Fts3Expr structure for a binary operator (any type ** except an FTSQUERY_PHRASE). Return an integer value representing the ** precedence of the operator. Lower values have a higher precedence (i.e. ** group more tightly). For example, in the C language, the == operator ** groups more tightly than ||, and would therefore have a higher precedence. *** When using the new fts3 query syntax (when SQLITE_ENABLE_FTS3_PARENTHESIS ** is defined), the order of the operators in precedence from highest to ** lowest is: ** ** NEAR ** NOT ** AND (including implicit ANDs) ** OR ** ** Note that when using the old query syntax, the OR operator has a higher ** precedence than the AND operator.

77212. iTermLeafOffset may be equal to szLeaf if the term is the last ** thing on the page - i.e. the first rowid is on the following page. ** In this case leave pIter->pLeaf==0, this iterator is at EOF.

77213. Estimated number of result rows

77214. Do constraint checks.

77215. Current side of the stack

77216. ** Given the nKey-byte encoding of a record in pKey[], populate the ** UnpackedRecord structure indicated by the fourth argument with the ** contents of the decoded record.

77217. ** The magic Explain opcode are only inserted when explain==2 (which ** is to say when the EXPLAIN QUERY PLAN syntax is used.) ** This opcode records information from the optimizer. It is the ** the same as a no-op. This opcode never appears in a real VM program.

77218. ** CAPI3REF: Constants Returned By The Conflict Handler ** ** A conflict handler callback must return one of the following three values. ** ** <dl> ** <dt>SQLITE_CHANGESET OMIT<dd> ** If a conflict handler returns this value no special action is taken. The ** change that caused the conflict is not applied. The session module ** continues to the next change in the changeset. ** ** <dt>SQLITE_CHANGESET_REPLACE<dd> ** This value may only be returned if the second argument to the conflict ** handler was SQLITE_CHANGESET_DATA or SQLITE_CHANGESET_CONFLICT. If this ** is not the case, any changes applied so far are rolled back and the ** call to sqlite3changeset_apply() returns SQLITE_MISUSE. ** ** If CHANGESET_REPLACE is returned by an SQLITE_CHANGESET_DATA conflict ** handler, then the conflicting row is either updated or deleted, depending ** on the type of change. ** ** If CHANGESET_REPLACE is returned by an SQLITE_CHANGESET_CONFLICT conflict ** handler, then the conflicting row is removed from the database and a ** second attempt to apply the change is made. If this second attempt fails, ** the original row is restored to the database before continuing. ** ** <dt>SQLITE_CHANGESET_ABORT<dd> ** If this value is returned, any changes applied so far are rolled back ** and the call to sqlite3changeset_apply() returns SQLITE_ABORT. ** </dl>

77219. Code AFTER triggers

77220. ** The "unknown" function is automatically substituted in place of ** any unrecognized function name when doing an EXPLAIN or EXPLAIN QUERY PLAN ** when the SQLITE_ENABLE_UNKNOWN_FUNCTION compile-time option is used. ** When the "sqlite3" command-line shell is built using this functionality, ** that allows an EXPLAIN or EXPLAIN QUERY PLAN for complex queries ** involving application-defined functions to be examined in a generic ** sqlite3 shell.

77221. The file descriptor

77222. ** Implementation of the abs() function. ** ** IMP: R-23979-26855 The abs(X) function returns the absolute value of ** the numeric argument X.

77223. True for final fragment in snippet

77224. Flags passed as 4th argument to xOpen()

77225. Allocated size of aOverflow[] array

77226. 870

77227. LIMIT counter

77228. True if a header read failed

77229. ** Walk all expressions associated with SELECT statement p. Do ** not invoke the SELECT callback on p, but do (of course) invoke ** any expr callbacks and SELECT callbacks that come from subqueries. ** Return WRC_Abort or WRC_Continue.

77230. The requested memory region does not exist. If isWrite is set to ** zero, exit early. *pp will be set to NULL and SQLITE_OK returned. ** ** Alternatively, if isWrite is non-zero, use ftruncate() to allocate ** the requested memory region.

77231. ** Initialize a DbFixer structure. This routine must be called prior ** to passing the structure to one of the sqliteFixAAAA() routines below.

77232. Rowid of final block in segment (or 0)

77233. limit_opt ::= LIMIT expr COMMA expr

77234. Name of the view, trigger, or index

77235. Size of zLeTerm in bytes

77236. Only called if really needed

77237. 82

77238. **comment:** ** If *pRc is not SQLITE_OK when this function is called, it is a no-op. ** Otherwise, if the allocation at pBlob->a is not already at least nMin ** bytes in size, extend (realloc) it to be so. ** ** If an OOM error occurs, set *pRc to SQLITE_NOMEM and leave pBlob->a ** unmodified. Otherwise, if the allocation succeeds, update pBlob->nAlloc ** to reflect the new size of the pBlob->a[] buffer.

label: code-design

77239. Mask of phrases covered by this snippet

77240. ** Schema of the terms table.

77241. Table zTbl

77242. Write the new IncrMerger here

77243. ** Handle type for pages.

77244. Column to score

77245. Omit error if index already exists

77246. Attempt to omit tables from the join that do not effect the result

77247. SQLITE_HASH_H

77248. This routine never gets call with a positive pgno except right ** after sqlite3PcacheCleanAll(). So if there are dirty pages, ** it must be that pgno==0.

77249. Rows visited by (pFrom+pWLLoop)

77250. "*" at the end of the pattern matches

77251. Used to iterate through columns

77252. Fts3Table

77253. Skip pages already processed

77254. Current index in output buffer

77255. Length of key in bytes

77256. Register holding return address of addrFillSub

77257. Try to return memory used by the pcache module to the main memory heap

77258. Otherwise, treat the ORDER BY term as an ordinary expression

77259. Arguments for SQLITE_PRINTF_SQLFUNC

77260. Destructor for BtShared.pSchema

77261. ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This header file defines the interface that the sqlite B-Tree file ** subsystem. See comments in the source code for a detailed description ** of what each interface routine does.

77262. Must be compatible with this index

77263. b: p1 (unmasked)

77264. **comment:** This OP_Delete opcode fires the pre-update-hook only. It does ** not modify the b-tree. It is more efficient to let the coming ** OP_Insert replace the existing entry than it is to delete the ** existing entry and then insert a new one.

label: code-design

77265. The user has issued a query of the form "MATCH '*'...". This ** indicates that the MATCH expression is not a full text query, ** but a request for an internal parameter.

77266. Size of the Bitvec structure in bytes.

77267. Propagate these bits up tree

77268. Pointer to doclist of current entry

77269. ** Return a 32-bit hash of the page data for pPage.

77270. Constraint handling. A write operation on an r-tree table may return ** SQLITE_CONSTRAINT for two reasons: ** ** 1. A duplicate rowid value, or ** 2. The supplied data violates the "x2>=x1" constraint. ** ** In the first case, if the conflict-handling mode is REPLACE, then ** the conflicting row can be removed before proceeding. In the second ** case, SQLITE_CONSTRAINT must be returned regardless of the ** conflict-handling mode specified by the user.

77271. OUT: Size of record in bytes

77272. ** Expression pExpr must be of type FTSQUERY_PHRASE. *** If it is not already allocated and populated, this function allocates and ** populates the Fts3Expr.aMI[] array for expression pExpr. If pExpr is part ** of a NEAR expression, then it also allocates and populates the same array ** for all other phrases that are part of the NEAR expression. *** SQLITE_OK is returned if the aMI[] array is successfully allocated and ** populated. Otherwise, if an error occurs, an SQLite error code is returned.

77273. **comment:** TODO(shess) This needs expansion to handle UTF-8 ** case-insensitivity.
label: code-design

77274. Template WhereLoop object

77275. Number of entries in pPath->aLoop[]

77276. Flags passed to sqlite3WhereBegin()

77277. ** Change the P5 operand on the last opcode (which should be an OP_MakeRecord) ** to be the number of columns in table pTab that must not be NULL-trimmed. *** Or if no columns of pTab may be NULL-trimmed, leave P5 at zero.

77278. Keying information for the group by clause

77279. #include <time.h>

77280. ** Find a unique file ID for the given absolute pathname. Return ** a pointer to the vxworksFileId object. This pointer is the unique ** file ID. *** The nRef field of the vxworksFileId object is incremented before ** the object is returned. A new vxworksFileId object is created ** and added to the global list if necessary. *** If a memory allocation error occurs, return NULL.

77281. ** This routine generates VDBE code that causes the deletion of all ** index entries associated with a single row of a single table, pTab *** Preconditions: ***
1. A read/write cursor "iDataCur" must be open on the canonical storage ** btree for the table pTab. (This will be either the table itself ** for rowid tables or to the primary key index for WITHOUT ROWID ** tables.) *** 2. Read/write cursors for all indices of pTab must be open as ** cursor number iIdxCur+i for the i-th index. (The pTab->pIndex ** index is the 0-th index.) *** 3. The "iDataCur" cursor must be already be positioned on the row ** that is to be deleted.

77282. Register all VFSes defined in the aVFs[] array

77283. Database number

77284. star_opt ::=

77285. Finally, delete what is left of the subquery and return ** success.

77286. Generate the subroutine return

77287. Mask of cursor flags (see below)

77288. we couldn't create the proxy lock file with the old lock file path ** so try again via auto-naming

77289. Appending to an existing hash-entry. Check that there is enough ** space to append the largest possible new entry. Worst case scenario ** is: *** + 9 bytes for a new rowid, ** + 4 byte reserved for the "poslist size" varint. ** + 1 byte for a "new column" byte, ** + 3 bytes for a new column number (16-bit max) as a varint, ** + 5 bytes for the new position offset (32-bit max).

77290. The ORDER BY and GROUP BY clauses may not refer to terms in ** outer queries

77291. leaves_end_block

77292. Do the rest of the initialization under the recursive mutex so ** that we will be able to handle recursive calls into ** sqlite3_initialize(). The recursive calls normally come through ** sqlite3_os_init() when it invokes sqlite3_vfs_register(), but other ** recursive calls might also be possible. *** IMPLEMENTATION-OF: R-00140-37445 SQLite automatically serializes calls ** to the xInit method, so the xInit method need not be threadsafe. *** The following mutex is what serializes access to the appdef pcache xInit ** methods. The sqlite3_pcachemethods.xInit() all is embedded in the ** call to sqlite3PcacheInitialize().

77293. Load an unaligned and unsigned 32-bit integer

77294. ** Compare the contents of the pLeft buffer with the pRight/nRight blob. ** ** Return -ve if pLeft is smaller than pRight, 0 if they are equal or ** +ve if pRight is smaller than pLeft. In other words: *** res = *pLeft - *pRight

77295. ***** Everything after this point is just test code.

77296. OUT: uid of zFile.

77297. Not the root node

77298. Bitmap of additional columns

77299. Obtain the average docsize (in pages).

77300. ** The various operations on open token or token prefix iterators opened ** using sqlite3Fts5IndexQuery().

77301. ** pZ is a UTF-8 encoded unicode string. If nByte is less than zero, ** return the number of unicode characters in pZ up to (but not including) ** the first 0x00 byte. If nByte is not less than zero, return the ** number of unicode characters in the first nByte of pZ (or up to ** the first 0x00, whichever comes first).

77302. The first nOBSat columns of the previous row

77303. Opcode: AggFinal P1 P2 * P4 *** Synopsis: accum=r[P1] N=P2 *** Execute the finalizer function for an aggregate. P1 is ** the memory location that is the accumulator for the aggregate. *** P2 is the number of arguments that the step function takes and ** P4 is a pointer to the FuncDef for this function. The P2 ** argument is not used by this opcode. It is only there to disambiguate ** functions that can take varying numbers of arguments. The ** P4 argument is only needed for the degenerate case where ** the step function was not previously called.

77304. 1170

77305. ** Subterms pOne and pTwo are contained within WHERE clause pWC. The ** two subterms are in disjunction - they are OR-ed together. *** If these two terms are both of the form: "A op B" with the same ** A and B values but different operators and if the operators are ** compatible (if one is = and the other is <, for example) then ** add a new virtual AND term to pWC that is the combination of the ** two. *** Some examples: *** x<y OR x=y --> x<=y ** x=y OR x=y --> x<=y ** x<=y OR x<y --> x<=y *** The following is NOT generated: *** x<y OR x>y --> x!=y

77306. Containing WITH clause, or NULL

77307. End if not an rowid index

77308. ** Playback savepoint pSavepoint. Or, if pSavepoint==NULL, then playback ** the entire master journal file. The case pSavepoint==NULL occurs when ** a ROLLBACK TO command is invoked on a SAVEPOINT that is a transaction ** savepoint. *** When pSavepoint is not NULL (meaning a non-transaction savepoint is ** being rolled back), then the rollback consists of up to three stages, ** performed in the order specified: *** * Pages are played back from the main journal starting at byte ** offset PagerSavepoint.iOffset and continuing to ** PagerSavepoint.iHdrOffset, or to the end of the main journal ** file if PagerSavepoint.iHdrOffset is zero. *** * If PagerSavepoint.iHdrOffset is not zero, then pages are played ** back starting from the journal header immediately following ** PagerSavepoint.iHdrOffset to the end of the main journal file. *** * Pages are then played back from the sub-journal file, starting ** with the PagerSavepoint.iSubRec and continuing to the end of ** the journal file. *** Throughout the rollback process, each time a page is rolled back, the ** corresponding bit is set in a bitvec structure (variable pDone in the ** implementation below). This is used to ensure that a page is only ** rolled back the first time it is encountered in either journal. *** If pSavepoint is NULL, then pages are only played back from the main ** journal file. There is no need for a bitvec in this case. *** In either case, before playback commences the Pager.dbSize variable ** is reset to the value that it held at the start of the savepoint ** (or transaction). No page with a page-number greater than this value ** is played back. If one is encountered it is simply skipped.

77309. ** Find the current time (in Universal Coordinated Time). Write into *piNow ** the current time and date as a Julian Day number times 86_400_000. In ** other words, write into *piNow the number of milliseconds since the Julian ** epoch of noon in Greenwich on November 24, 4714 B.C according to the ** proleptic Gregorian calendar. *** On success, return SQLITE_OK. Return SQLITE_ERROR if the time and date ** cannot be found.

77310. ** The winFile structure is a subclass of sqlite3_file* specific to the win32 ** portability layer.

77311. 289

77312. True if this is a VIEW

77313. destructor for json_each virtual table

77314. Add a WO_MATCH auxiliary term to the constraint set if the ** current expression is of the form: column MATCH expr. ** This information is used by the xBestIndex methods of ** virtual tables. The native query optimizer does not attempt ** to do anything with MATCH functions.

77315. IN/OUT: Pointer to position list

77316. ** The pVal argument is known to be a value other than NULL. ** Convert it into a string with encoding enc and return a pointer ** to a zero-terminated version of that string.

77317. **comment:** Malloc utility
label: code-design

77318. **comment:** unused
label: code-design

77319. ** This routine deallocates a previously allocated mutex.

77320. Do no locking
77321. ** Generate a CREATE TABLE statement appropriate for the given ** table. Memory to hold the text of the statement is obtained ** from sqliteMalloc() and must be freed by the calling function.
77322. 1180
77323. ** A sort order can be either ASC or DESC.
77324. New cell to insert if nData>1
77325. If non-NULL, store record pointer here
77326. ** Attempt to extract a value from pExpr and use it to construct *ppVal. ** ** If pAlloc is not NULL, then an UnpackedRecord object is created for ** pAlloc if one does not exist and the new value is added to the ** UnpackedRecord object. ** ** A value is extracted in the following cases: ** ** *(pExpr==0). In this case the value is assumed to be an SQL NULL, ** ** * The expression is a bound variable, and this is a reprepare, or ** ** * The expression is a literal value. ** ** On success, *ppVal is made to point to the extracted value. The caller ** is responsible for ensuring that the value is eventually freed.
77327. * Calculate the maximum legal cache size, in pages, based on the maximum * possible initial heap size and the default page size, setting aside the * needed extra space.
77328. Binary operator
77329. Trace sqlite3VdbeAddOp() calls
77330. Return the current rowid value
77331. Combines with pBt->pPager->iDataVersion
77332. If we have not already resolved the name, then maybe ** it is a new.* or old.* trigger argument reference
77333. **comment:** ** The following two macros are used internally. They are similar to the ** sqlite3changeset_new() and sqlite3changeset_old() functions, except that ** they omit all error checking and return a pointer to the requested value.
label: code-design
77334. !defined(SQLITE_OMIT_WAL) || SQLITE_MAX_MMAP_SIZE>0
77335. aIndex
77336. Database page number for frame
77337. The underlying unixShmNode object
77338. ** 2011 Jan 27 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

77339. The sqlite3GetBoolean() function is used by other modules but the ** remainder of this file is specific to PRAGMA processing. So omit ** the rest of the file if PRAGMAs are omitted from the build.
77340. Varini that may be written to pOut
77341. Number of outputs for this subquery
77342. Hint blob to append to
77343. For scanning through pCell
77344. A "NOT INDEXED" clause was supplied. See parse.y ** construct "indexed_opt" for details.
77345. ** Extract the user data from a sqlite3_context structure and return a ** pointer to it. ** ** IMPLEMENTATION-OF: R-46798-50301 The
sqlite3_context_db_handle() interface ** returns a copy of the pointer to the database connection (the 1st ** parameter) of the sqlite3_create_function() and **
sqlite3_create_function16() routines that originally registered the ** application defined function.
77346. ** Close a file.
77347. 190
77348. The virtual machine being built up
77349. The letter 'x', or start of BLOB literal
77350. OPTIMIZATION-IF-TRUE
77351. Append JSON content here
77352. **comment:** ** This routine redistributes cells on the iParentIdx'th child of pParent ** (hereafter "the page") and up to 2 siblings so that all pages have about the ** same amount of free space. Usually a single sibling on either side of the ** page are used in the balancing, though both siblings might come from one ** side if the page is the first or last child of its parent. If the page ** has fewer than 2 siblings (something which can only happen if the page ** is a root page or a child of a root page) then all available siblings ** participate in the balancing. ** ** The number of siblings of the page might be increased or decreased by ** one or two in an effort to keep pages nearly full but not over full. ** ** Note that when this routine is called, some of the cells on the page ** might not actually be stored in MemPage.aData[]. This can happen ** if the page is overfull. This routine ensures that all cells allocated ** to the page and its siblings fit into MemPage.aData[] before returning. ** ** In the course of balancing the page and its siblings, cells may be ** inserted into or removed from the parent page (pParent). Doing so ** may cause the parent page to become overfull or underfull. If this ** happens, it is the responsibility of the caller to invoke the correct ** balancing routine to fix this problem (see the balance() routine). ** ** If this routine fails for any reason, it might leave the database ** in a corrupted state. So if this routine fails, the database should ** be rolled back. ** ** The third argument to this function, aOvflSpace, is a pointer to a ** buffer big enough to hold one page. If while inserting cells into the parent ** page (pParent) the parent page becomes overfull, this buffer is ** used to store the parent's overflow cells. Because this function inserts ** a maximum of four divider cells into the parent page, and the maximum ** size of a cell stored within an internal node is always less than 1/4 ** of the page-size, the aOvflSpace[] buffer is guaranteed to be large ** enough for all overflow cells. ** ** If aOvflSpace is set to a null pointer, this function returns **
SQLITE_NOMEM.
label: code-design
77353. If non-NULL, write serialized value here
77354. p2 is smaller
77355. Create transient tab and store like SRT_Table
77356. **comment:** ** Return the number of bytes of space at the end of every page that ** are intentionally left unused. This is the "reserved" space that is ** sometimes used by extensions. ** ** If SQLITE_HAS_MUTEX is defined then the number returned is the ** greater of the current reserved space and the maximum requested ** reserve space.
label: code-design
77357. xGeom and xQueryFunc argument
77358. we can skip these cause they were (effectively) done above in calc'ing s
77359. Skip backwards passed any trailing 0x00 bytes added by NearTrim()
77360. Page number of the new child page
77361. Count rows changed by INSERT,
77362. The token itself
77363. Best snippet score found so far
77364. append the "-conch" suffix to the file
77365. The count-of-view optimization
77366. Result of comparison operation
77367. All duplicates are adjacent
77368. If using separate RBU and state databases, attach the state database to ** the RBU db handle now.
77369. Offset in zDoc of start of token
77370. ** 2006 July 10 ** ** The author disclaims copyright to this source code. **

Defines the interface to tokenizers used by fulltext-search. There ** are three basic components: ** ** sqlite3_tokenizer_module is a singleton defining the tokenizer ** interface functions. This is essentially the class structure for ** tokenizers. ** ** sqlite3_tokenizer is used to define a particular tokenizer, perhaps ** including customization information defined at creation time. ** **
sqlite3_tokenizer_cursor is generated by a tokenizer to generate ** tokens from a particular input.
77371. Cursors in the equivalence class
77372. 2 NORMAL:
77373. **comment:** ** This is part of the fts5_decode() debugging aid. ** ** Arguments pBlob/nBlob contain a serialized Fts5Structure object. This ** function appends a human-readable representation of the same object ** to the buffer passed as the second argument.

label: code-design

77374. ** Write a 64-bit variable-length integer to memory starting at p[0]. ** The length of data write will be between 1 and 9 bytes. The number ** of bytes written is returned. *** ** A variable-length integer consists of the lower 7 bits of each byte ** for all bytes that have the 8th bit set and one byte with the 8th ** bit clear. Except, if we get to the 9th byte, it stores the full ** 8 bits and is the last byte.

77375. FTS3 Cursor object

77376. OUT: Dlidx flag

77377. True to fail localtime() calls

77378. ASC or DESC on the i-th expression

77379. EVIDENCE-OFF: R-62920-47450 The busy-handler callback is never invoked ** in the SQLITE_CHECKPOINT_PASSIVE mode.

77380. I/O error: lastErrno set by seekAndRead

77381. CTIME_KW => ID

77382. 328

77383. ** PRAGMA [schema.]wal_checkpoint = passive|full|restart|truncate ** ** Checkpoint the database.

77384. Size of buffer at a[] in bytes

77385. Check if there were any options specified that should be interpreted ** here. Options that are interpreted here include "vfs" and those that ** correspond to flags that may be passed to the sqlite3_open_v2() ** method.

77386. Verify the affinity and collating sequence match

77387. True to sort in descending order

77388. Open an ephemeral index to use for the distinct set.

77389. **comment:** ** EXTENSION API FUNCTIONS ** ** xUserData(pFts): ** Return a copy of the context pointer the extension function was ** registered with. ** ** xColumnTotalSize(pFts, iCol, pnToken): ** If parameter iCol is less than zero, set output variable *pnToken ** to the total number of tokens in the FTS5 table. Or, if iCol is ** non-negative but less than the number of columns in the table, return ** the total number of tokens in column iCol, considering all rows in ** the FTS5 table. *** ** If parameter iCol is greater than or equal to the number of columns ** in the table, SQLITE_RANGE is returned. Or, if an error occurs (e.g. ** an OOM condition or IO error), an appropriate SQLite error code is ** returned. *** ** xColumnCount(pFts): ** Return the number of columns in the table. *** ** xColumnSize(pFts, iCol, pnToken): ** If parameter iCol is less than zero, set output variable *pnToken ** to the total number of tokens in the current row. Or, if iCol is ** non-negative but less than the number of columns in the table, set ** *pnToken to the number of tokens in column iCol of the current row. *** ** If parameter iCol is greater than or equal to the number of columns ** in the table, SQLITE_RANGE is returned. Or, if an error occurs (e.g. ** an OOM condition or IO error), an appropriate SQLite error code is ** returned. *** ** This function may be quite inefficient if used with an FTS5 table ** created with the "columnsize=0" option. *** ** xColumnText: ** This function attempts to retrieve the text of column iCol of the ** current document. If successful, (*pz) is set to point to a buffer ** containing the text in utf-8 encoding, (*pn) is set to the size in bytes ** (not characters) of the buffer and SQLITE_OK is returned. Otherwise, ** if an error occurs, an SQLite error code is returned and the final values ** of (*pz) and (*pn) are undefined. *** ** xPhraseCount: ** Returns the number of phrases in the current query expression. *** ** xPhraseSize: ** Returns the number of tokens in phrase iPhrase of the query. Phrases ** are numbered starting from zero. *** ** xInstCount: ** Set *pnInst to the total number of occurrences of all phrases within ** the query within the current row. Return SQLITE_OK if successful, or ** an error code (i.e. SQLITE_NOMEM) if an error occurs. *** ** This API can be quite slow if used with an FTS5 table created with the ** "detail=none" or "detail=column" option. If the FTS5 table is created ** with either "detail=none" or "detail=column" and "content=" option ** (i.e. if it is a contentless table), then this API always returns 0. *** ** xInst: ** Query for the details of phrase match iIdx within the current row. ** Phrase matches are numbered starting from zero, so the iIdx argument ** should be greater than or equal to zero and smaller than the value ** output by xInstCount(). *** ** Usually, output parameter *piPhrase is set to the phrase number, *piCol ** to the column in which it occurs and *piOff the token offset of the ** first token of the phrase. The exception is if the table was created ** with the offsets=0 option specified. In this case *piOff is always ** set to -1. *** ** Returns SQLITE_OK if successful, or an error code (i.e. SQLITE_NOMEM) ** if an error occurs. *** ** This API can be quite slow if used with an FTS5 table created with the ** "detail=none" or "detail=column" option. *** ** xRowid: ** Returns the rowid of the current row. *** ** xTokenize: ** Tokenize text using the tokenizer belonging to the FTS5 table. *** ** xQueryPhrase(pFts5, iPhrase, pUserData, xCallback): ** This API function is used to query the FTS table for phrase iPhrase ** of the current query. Specifically, a query equivalent to: *** ... FROM ftstable WHERE ftstable MATCH \$p ORDER BY rowid *** with \$p set to a phrase equivalent to the phrase iPhrase of the ** current query is executed. Any column filter that applies to ** phrase iPhrase of the current query is included in \$p. For each ** row visited, the callback function passed as the fourth argument ** is invoked. The context and API objects passed to the callback ** function may be used to access the properties of each matched row. ** Invoking Api.xUserData() returns a copy of the pointer passed as ** the third argument to pUserData. *** ** If the callback function returns any value other than SQLITE_OK, the ** query is abandoned and the xQueryPhrase function returns immediately. ** If the returned value is SQLITE_DONE, xQueryPhrase returns SQLITE_OK. ** Otherwise, the error code is propagated upwards. ** ** If the query runs to completion without incident, SQLITE_OK is returned. ** Or, if some error occurs before the query completes or is aborted by ** the callback, an SQLite error code is returned. *** ** xSetAuxdata(pFts5, pAux, xDelete) ** ** Save the pointer passed as the second argument as the extension functions ** "auxiliary data". The pointer may then be retrieved by the current or any ** future invocation of the same fts5 extension function made as part of ** of the same MATCH query using the xGetAuxdata() API. *** ** Each extension function is allocated a single auxiliary data slot for ** each FTS query (MATCH expression). If the extension function is invoked ** more than once for a single FTS query, then all invocations share a ** single auxiliary data context. *** ** If there is already an auxiliary data pointer when this function is ** invoked, then it is replaced by the new pointer. If an xDelete callback ** was specified along with the original pointer, it is invoked at this ** point. *** ** The xDelete callback, if one is specified, is also invoked on the ** auxiliary data pointer after the FTS5 query has finished. *** ** If an error (e.g. an OOM condition) occurs within this function, an ** the auxiliary data is set to NULL and an error code returned. If the ** xDelete parameter was not NULL, it is invoked on the auxiliary data ** pointer before returning. *** ** xGetAuxdata(pFts5, bClear) ** ** Returns the current auxiliary data pointer for the fts5 extension ** function. See the xSetAuxdata() method for details. *** ** If the bClear argument is non-zero, then the auxiliary data is cleared ** (set to NULL) before this function returns. In this case the xDelete, ** if any, is not invoked. *** ** xRowCount(pFts5, pnRow) ** ** This function is used to retrieve the total number of rows in the table. ** In other words, the same value that would be returned by: ** SELECT count(*) FROM ftstable; *** ** xPhraseFirst(): ** This function is used, along with type Fts5PhraseIter and the xPhraseNext() method, to iterate through all instances of a single query phrase within ** the current row. This is the same information as is accessible via the ** xInstCount/xInst APIs. While the xInstCount/xInst APIs are more convenient ** to use, this API may be faster under some circumstances. To iterate ** through instances of phrase iPhrase, use the following code: *** ** Fts5PhraseIter iter; ** int iCol, iOff; ** for(pApi->xPhraseFirst(pFts, iPhrase, &iter, &iCol, &iOff); ** iCol>=0; ** pApi->xPhraseNext(pFts, &iter, &iCol, &iOff) **){ ** // An instance of phrase iPhrase at offset iOff of column iCol ** } *** ** The Fts5PhraseIter structure is defined above. Applications should not ** modify this structure directly - it should only be used as shown above ** with the xPhraseFirst() and xPhraseNext() API methods (and by ** xPhraseFirstColumn() and xPhraseNextColumn() as illustrated below). *** ** This API can be quite slow if used with an FTS5 table created with the ** "detail=none" or "detail=column" option. If the FTS5 table is created ** with either "detail=none" or "detail=column" and "content=" option ** (i.e. if it is a contentless table), then this API always iterates ** through an empty set (all calls to xPhraseFirst() set iCol to -1). *** ** xPhraseNext(): ** See xPhraseFirst above. *** ** xPhraseFirstColumn(): ** This function and xPhraseNextColumn() are similar to the xPhraseFirst() ** and xPhraseNext() APIs described above. The difference is that instead ** of iterating through all instances of a phrase in the current row, these ** APIs are used to iterate through the set of columns in the current row ** that contain one or more instances of a specified phrase. For example: *** ** Fts5PhraseIter iter; ** int iCol; ** for(pApi->xPhraseFirstColumn(pFts, iPhrase, &iter, &iCol); ** iCol>=0; ** pApi->xPhraseNextColumn(pFts, &iter, &iCol) **){ ** // Column iCol contains at least one instance of phrase iPhrase ** } *** ** This API can be quite slow if used with an FTS5 table created with the ** "detail=none" option. If the FTS5 table is created with either ** "detail=none" or "content=" option (i.e. if it is a contentless table), ** then this API always iterates through an empty set (all calls to ** xPhraseFirstColumn() set iCol to -1). *** ** The information accessed using this API and its companion ** xPhraseFirstColumn() may also be obtained using xPhraseFirst/xPhraseNext ** (or xInst/xInstCount). The chief advantage of this API is that it is ** significantly more efficient than those alternatives when used with ** "detail=column" tables. *** ** xPhraseNextColumn(): ** See xPhraseFirstColumn above.

label: code-design

77390. A WO_xx value describing <op>

77391. defer_subclause_opt ::=

77392. SELECT may not have a WHERE clause

77393. ** Return the pathname of the journal file for this database. The return ** value of this routine is the same regardless of whether the journal file ** has been created or not. *** ** The pager journal filename is invariant as long as the pager is ** open so it is safe to access without the BtShared mutex.

77394. ** Prototypes for the VDBE interface. See comments on the implementation ** for a description of what each of these routines does.

77395. 650

77396. Name context to resolve expressions in

77397. 203

77398. **comment:** ** All of the static variables used by this module are collected ** into a single structure named "mem5". This is to keep the ** static variables organized and to reduce namespace pollution ** when this module is combined with other in the amalgamation.
label: code-design

77399. 12
77400. Collation sequence name
77401. Lock offset to lock for checkpoint
77402. Allowed values for second argument to pcacheManageDirtyList()
77403. Original I/O methods for close
77404. ** Deinitialize this module.
77405. Opcode: ShiftRight P1 P2 P3 * * * Synopsis: r[P3]=r[P2]>>r[P1] ** ** Shift the integer value in register P2 to the right by the ** number of bits specified by the integer in register P1. ** Store the result in register P3. ** If either input is NULL, the result is NULL.
77406. Do no file locking
77407. First search for a match amongst the application-defined functions.
77408. wqlist ::= nm eidlist_opt AS LP select RP
77409. ** Undo the effects of sqlite3_initialize(). Must not be called while ** there are outstanding database connections or memory allocations or ** while any part of SQLite is otherwise in use in any thread. This ** routine is not threadsafe. But it is safe to invoke this routine ** on when SQLite is already shut down. If SQLite is already shut down ** when this routine is invoked, then this routine is a harmless no-op.
77410. 8-byte signed integer
77411. The table to be opened
77412. Name of the journal file
77413. ** Compute the iLimit and iOffset fields of the SELECT based on the ** pLimit and pOffset expressions. pLimit and pOffset hold the expressions ** that appear in the original SQL statement after the LIMIT and OFFSET ** keywords. Or NULL if those keywords are omitted. iLimit and iOffset ** are the integer memory register numbers for counters used to compute ** the limit and offset. If there is no limit and/or offset, then ** iLimit and iOffset are negative. ** ** This routine changes the values of iLimit and iOffset only if ** a limit or offset is defined by pLimit and pOffset. iLimit and ** iOffset should have been preset to appropriate default values (zero) ** prior to calling this routine. ** ** The iOffset register (if it exists) is initialized to the value ** of the OFFSET. The iLimit register is initialized to LIMIT. Register ** iOffset+1 is initialized to LIMIT+OFFSET. ** ** Only if pLimit!=0 or pOffset!=0 do the limit registers get ** redefined. The UNION ALL operator uses this property to force ** the reuse of the same limit and offset registers across multiple ** SELECT statements.
77414. ** If the Vdbe passed as the first argument opened a statement-transaction, ** close it now. Argument eOp must be either SAVEPOINT_ROLLBACK or ** SAVEPOINT_RELEASE. If it is SAVEPOINT_ROLLBACK, then the statement ** transaction is rolled back. If eOp is SAVEPOINT_RELEASE, then the ** statement transaction is committed. ** ** If an IO error occurs, an SQLITE_IOERR_XXX error code is returned. ** Otherwise SQLITE_OK.
77415. True if cache is purgeable
77416. Variables below this point are populated by fts3_expr.c when parsing ** a MATCH expression. Everything above is part of the evaluation phase.
77417. ** Write nData bytes of data to the PMA. Return SQLITE_OK ** if successful, or an SQLite error code if an error occurs.
77418. Check if the current doclist ends on this page. If it does, return ** early without loading the doclist-index (as it belongs to a different ** term).
77419. Counter incremented each transaction
77420. The number of pcache slots
77421. ***** What follows is a massive switch statement where each case implements a ** separate instruction in the virtual machine. If we follow the usual ** indentation conventions, each case should be indented by 6 spaces. But ** that is a lot of wasted space on the left margin. So the code within ** the switch statement will break with convention and be flush-left. Another ** big comment (similar to this one) will mark the point in the code where ** we transition back to normal indentation. ** ** The formatting of each case is important. The makefile for SQLite ** generates two C files "opcodes.h" and "opcodes.c" by scanning this ** file looking for lines that begin with "case OP_". The opcodes.h files ** will be filled with #defines that give unique integer values to each ** opcode and the opcodes.c file is filled with an array of strings where ** each string is the symbolic name for the corresponding opcode. If the ** case statement is followed by a comment of the form "# same as ... #" ** that comment is used to determine the particular value of the opcode. ** ** Other keywords in the comment that follows each case are used to ** construct the OPFLG_INITIALIZER value that initializes opcodeProperty[J]. ** Keywords include: in1, in2, in3, out2, out3. See ** the mkopcodeh.awk script for additional information. ** ** Documentation about VDBE opcodes is generated by scanning this file ** for lines of that contain "Opcode:". That line and all subsequent ** comment lines are used in the generation of the opcode.html documentation ** file. ** ** SUMMARY: ** ** Formatting is important to scripts that scan this file. ** Do not deviate from the formatting style currently in use. *****
77422. Number used to identify the index
77423. Initialize the name-context
77424. Begin writing at this offset into the file
77425. cmd ::= PRAGMA nm dbnm
77426. **comment:** If the payload will not fit completely on the local page, we have ** to decide how much to store locally and how much to spill onto ** overflow pages. The strategy is to minimize the amount of unused ** space on overflow pages while keeping the amount of local storage ** in between minLocal and maxLocal. ** ** Warning: changing the way overflow payload is distributed in any ** way will result in an incompatible file format.
label: code-design

77427. Phrase number (numbered from zero)
77428. First part of [schema.Jid field
77429. Initialize the locking parameters
77430. Mutex used to control access to shared lock
77431. Right key
77432. mem(P3) holds an integer
77433. If an existing TriggerPrg could not be located, create a new one.
77434. ** Compare two blobs. Return negative, zero, or positive if the first ** is less than, equal to, or greater than the second, respectively. ** If one blob is a prefix of the other, then the shorter is the lessor.
77435. ** Memory map or remap the file opened by file-descriptor pFd (if the file ** is already mapped, the existing mapping is replaced by the new). Or, if ** there already exists a mapping for this file, and there are still ** outstanding xFetch() references to it, this function is a no-op. ** ** If parameter nByte is non-negative, then it is the requested size of ** the mapping to create. Otherwise, if nByte is less than zero, then the ** requested size is the size of the file on disk. The actual size of the ** created mapping is either the requested size or the value configured ** using SQLITE_FCNTL_MMAP_SIZE, whichever is smaller. ** ** SQLITE_OK is returned if no error occurs (even if the mapping is not ** recreated as a result of outstanding references) or an SQLite error ** code otherwise.
77436. **comment:** ** Execute a rollback if a transaction is active and unlock the ** database file. ** ** If the pager has already entered the ERROR state, do not attempt ** the rollback at this time. Instead, pager_unlock() is called. The ** call to pager_unlock() will discard all in-memory pages, unlock ** the database file and move the pager back to OPEN state. If this ** means that there is a hot-journal left in the file-system, the next ** connection to obtain a shared lock on the pager (which may be this one) ** will roll it back. ** ** If the pager has not already entered the ERROR state, but an IO or ** malloc error occurs during a rollback, then this will itself cause ** the pager to enter the ERROR state. Which will be cleared by the ** call to pager_unlock(), as described above.
label: code-design

77437. **comment:** Bytes to copy from previous term
label: code-design

77438. Level of %_segdir entries to delete
77439. Flags passed to sqlite3_vfs.xOpen()
77440. OP_Delete/Insert: save cursor pos
77441. The next group of variables are used to track whether or not the ** transaction counter in bytes 24-27 of database files are updated ** whenever any part of the database changes. An assertion fault will ** occur if a file is updated without also updating the transaction ** counter. This test is made to avoid new problems similar to the ** one described by ticket #3584.
77442. True if rerunning after an auto-reprepare
77443. 220
77444. The database index in sqlite3.aDb[]
77445. How to compare records

77446. Read-only page is acceptable
77447. Built-in length() function
77448. Flags. See WINFILE_* below
77449. Values loaded from the %_config table
77450. This Vdbe program seeks a btree cursor to the identified ** db/table/row entry. The reason for using a vdbe program instead ** of writing code to use the b-tree layer directly is that the ** vdbe program will take advantage of the various transaction, ** locking and error handling infrastructure built into the vdbe. *** After seeking the cursor, the vdbe executes an OP_ResultRow. ** Code external to the Vdbe then "borrows" the b-tree cursor and ** uses it to implement the blob_read(), blob_write() and ** blob_bytes() functions. *** The sqlite3_blob_close() function finalizes the vdbe program, ** which closes the b-tree cursor and (possibly) commits the ** transaction.
77451. The mutex type
77452. Lastest rowid any iterator points to
77453. True if token ends with a "*" character
77454. If pTab is a virtual table, call ViewGetColumnNames() to ensure ** it is initialized.
77455. ** The second argument passed to this function may be NULL, or it may be ** an existing Fts5Colset object. This function returns a pointer to ** a new colset object containing the contents of (p) with new value column ** number iCol appended. *** If an OOM error occurs, store an error code in pParse and return NULL. ** The old colset object (if any) is not freed in this case.
77456. Create the rootpage for the index using CreateIndex. But before ** doing so, code a Noop instruction and store its address in ** Index.nnum. This is required in case this index is actually a ** PRIMARY KEY and the table is actually a WITHOUT ROWID table. In ** that case the convertToWithoutRowidTable() routine will replace ** the Noop with a Goto to jump over the VDBE code generated below.
77457. Maximum amount of payload held locally
77458. ** Allocate and return an Fts5Colset object specifying the inverse of ** the colset passed as the second argument. Free the colset passed ** as the second argument before returning.
77459. ** The backtrace functionality is only available with GLIBC
77460. ** Indicate that sqlite3ParserFree() will never be called with a null ** pointer.
77461. REAL
77462. Used by OP_Delete
77463. ** Put the DateTime object into its error state.
77464. Bitmask of phrases covered by snippet
77465. Location in parent of right-sibling pointer
77466. The parser context
77467. ** Return the current journal mode.
77468. **comment:** Byte of unused space on each page
 label: code-design
77469. 262
77470. ** This function attempts to transform the expression tree at (*pp) to ** an equivalent but more balanced form. The tree is modified in place. ** If successful, SQLITE_OK is returned and (*pp) set to point to the ** new root expression node. *** nMaxDepth is the maximum allowable depth of the balanced sub-tree. ** ** Otherwise, if an error occurs, an SQLite error code is returned and ** expression (*pp) freed.
77471. first page on free list trunk
77472. This mutex is needed because pFile->pInode is shared across threads
77473. Address of the A==B subroutine
77474. Wanting an exclusive lock?
77475. **comment:** Call to try to make pages clean
 label: code-design
77476. ** Free a dynamic mutex.
77477. ** If there are no outstanding cursors and we are not in the middle ** of a transaction but there is a read lock on the database, then ** this routine unrefs the first page of the database file which ** has the effect of releasing the read lock. *** If there is a transaction in progress, this routine is a no-op.
77478. NULL
77479. Next object in linked list
77480. Add the current term to the interior node tree. The term added to ** the interior tree must: *** a) be greater than the largest term on the leaf node just written ** to the database (still available in pWriter->zTerm), and *** b) be less than or equal to the term about to be added to the new ** leaf node (zTerm/nTerm). *** In other words, it must be the prefix of zTerm 1 byte longer than ** the common prefix (if any) of zTerm and pWriter->zTerm.
77481. ** This function is used to iterate backwards (from the end to start) ** through doclists. It is used by this module to iterate through phrase ** doclists in reverse and by the fts3_write.c module to iterate through ** pending-terms lists when writing to databases with "order=desc". *** The doclist may be sorted in ascending (parameter bDescIdx==0) or ** descending (parameter bDescIdx==1) order of docid. Regardless, this ** function iterates from the end of the doclist to the beginning.
77482. OOM detected prior to this routine
77483. OUT: Number of columns in table
77484. Write the page out to disk
77485. There is a quirk here. The users INSERT statement may have specified ** a value for the "rowid" field, for the "docid" field, or for both. ** Which is a problem, since "rowid" and "docid" are aliases for the ** same value. For example: *** INSERT INTO fts3tbl(rowid, docid) VALUES(1, 2); *** In FTS3, this is an error. It is an error to specify non-NULL values ** for both docid and some other rowid alias.
77486. True if constraints are supported
77487. Invalid name?
77488. ** Create a new JsonNode instance based on the arguments and append that ** instance to the JsonParse. Return the index in pParse->aNode[] of the ** new node, or -1 if a memory allocation fails.
77489. 32-bit is the default
77490. don't break the lock on short read or a version mismatch
77491. Flags to pass to mmap()
77492. Scan past delimiter characters
77493. ** Convert a Microsoft Unicode string to a multi-byte character string, ** using the ANSI or OEM code page. *** Space to hold the returned string is obtained from sqlite3_malloc().
77494. If the handle had any kind of transaction open, decrement the ** transaction count of the shared btree. If the transaction count ** reaches 0, set the shared state to TRANS_NONE. The unlockBtreeIfUnused() ** call below will unlock the pager.
77495. Append the pgidx to the page buffer. Set the szLeaf header field.
77496. TODO1: Fix this
77497. Destroy an sqlite3_value object previously obtained from ** sqlite3_value_dup().
77498. Reset the checkpoint-header. This is safe because this thread is ** currently holding locks that exclude all other readers, writers and ** checkpointer.
77499. Size of buffer pOut in bytes
77500. 56
77501. Prevent ON clause terms of a LEFT JOIN from being used to drive ** an index for tables to the left of the join.
77502. One side is a column, the other is not. Use the columns affinity.
77503. Assert that the caller has opened the required transaction.
77504. Page number corresponding to bterm
77505. Pointer to the "extra" space
77506. A long-standing parser bug is that this syntax was allowed: *** CREATE TRIGGER attached.demo AFTER INSERT ON attached.tab ** ^~~~~~ ** ** To maintain backwards compatibility, ignore the database ** name on pTableName if we are reparsing out of SQLITE_MASTER.
77507. Execute the statement to insert the record. Set *piDocid to the ** new docid value.
77508. Cache of doclist for this phrase.

77509. Task context
77510. True if a lookup of a single entry.
77511. A transaction must be open when this is called.
77512. ** Argument pVector points to a vector expression - either a TK_VECTOR ** or TK_SELECT that returns more than one column. This function returns ** the register number of a register that contains the value of ** element iField of the vector. *** If pVector is a TK_SELECT expression, then code for it must have ** already been generated using the exprCodeSubselect() routine. In this ** case parameter regSelect should be the first in an array of registers ** containing the results of the sub-select. *** If pVector is of type TK_VECTOR, then code for the requested field ** is generated. In this case (*pRegFree) may be set to the number of ** a temporary register to be freed by the caller before returning. *** Before returning, output parameter (*ppExpr) is set to point to the ** Expr object corresponding to element iElem of the vector.
77513. bLocaltimeFault
77514. FTS3 table
77515. ** End of interface to code in fts5_tokenizer.c. *****
77516. **comment:** If control flows to this point, this must be a regular token, or ** the end of the input. Read a regular token using the sqlite3_tokenizer ** interface. Before doing so, figure out if there is an explicit ** column specifier for the token. *** TODO: Strangely, it is not possible to associate a column specifier ** with a quoted phrase, only with a single token. Not sure if this was ** an implementation artifact or an intentional decision when fts3 was ** first implemented. Whichever it was, this module duplicates the ** limitation.
 label: code-design
77517. **comment:** Query flattening in sqlite3Select() might refill p->pOrderBy. ** Be sure to delete p->pOrderBy, therefore, to avoid a memory leak.
 label: code-design
77518. Total overflow pages used by doclists
77519. ** A function that loads a shared-library extension then returns NULL.
77520. Options
77521. If the current leaf page is full, flush it to disk.
77522. ** Allocate an RbuState object and load the contents of the rbu_state ** table into it. Return a pointer to the new object. It is the ** responsibility of the caller to eventually free the object using ** sqlite3_free(). *** If an error occurs, leave an error code and message in the rbu handle ** and return NULL.
77523. Sync the database file to disk.
77524. If mapping failed, close the shared memory handle and erase it
77525. Probe for stat4 (if required)
77526. Collect 'global' stats as well as local
77527. aPgn0[] index of page number to use
77528. 184
77529. (3)
77530. Check that the list really is sorted now.
77531. ***** End of memjournal.c *****
77532. Preallocated space for the PCache
77533. Conflict resolution strategy
77534. Only one recursive reference is permitted.
77535. Rendering buffer
77536. 261
77537. The largest iLevel value in the tree
77538. WHEN clause
77539. Already in locking_mode=NORMAL
77540. The only reason to read this page is to obtain the page ** number for the next page in the overflow chain. The page ** data is not required. So first try to lookup the overflow ** page-list cache, if any, then fall back to the getOverflowPage() ** function.
77541. 10
77542. Non-zero if in an error state
77543. 290
77544. Linear scan of %_content table
77545. synopsis: r[P2]=P4 (len=P1)
77546. ** Bind the PRIMARY KEY values from the change passed in argument pChange ** to the SELECT statement passed as the first argument. The SELECT statement ** is as prepared by function sessionSelectStmt(). *** Return SQLITE_OK if all PK values are successfully bound, or an SQLite ** error code (e.g. SQLITE_NOMEM) otherwise.
77547. Number of entries in aInLoop[]
77548. The cksum argument passed to this function is a checksum calculated ** based on all expected entries in the FTS index (including prefix index ** entries). This block checks that a checksum calculated based on the ** actual contents of FTS index is identical. *** Two versions of the same checksum are calculated. The first (stack ** variable cksum2) based on entries extracted from the full-text index ** while doing a linear scan of each individual index in turn. *** As each term visited by the linear scans, a separate query for the ** same term is performed. cksum3 is calculated based on the entries ** extracted by these queries.
77549. ** CAPI3REF: Determine If A Prepared Statement Has Been Reset ** METHOD: sqlite3_stmt ** ** ^The sqlite3_stmt_busy(S) interface returns true (non-zero) if the ** [prepared statement] S has been stepped at least once using ** [sqlite3_step(S)] but has neither run to completion (returned ** [SQLITE_DONE] from [sqlite3_step(S)]) nor ** been reset using [sqlite3_reset(S)]. ^The sqlite3_stmt_busy(S) ** interface returns false if S is a NULL pointer. If S is not a ** NULL pointer and is not a pointer to a valid [prepared statement] ** object, then the behavior is undefined and probably undesirable. *** This interface can be used in combination [sqlite3_next_stmt()] ** to locate all prepared statements associated with a database ** connection that are in need of being reset. This can be used, ** for example, in diagnostic routines to search for prepared ** statements that are holding a transaction open.
77550. ** Return true if the argument interpreted as a unicode codepoint ** is a diacritical modifier character.
77551. e8..ef

77552. The underlying mmapped file
77553. State used by the fts5DataXXX() functions.
77554. If creating a temp table, the name may not be qualified. Unless ** the database name is "temp" anyway.
77555. Get a list of all dirty pages in the cache, sorted by page number
77556. ** Remove the i-th cell from pPage. This routine effects pPage only. ** The cell content is not freed or deallocated. It is assumed that ** the cell content has been copied someplace else. This routine just ** removes the reference to the cell from pPage. *** "sz" must be the number of bytes in the cell.
77557. The "subtype" set for JSON values
77558. ** Rtree virtual table module xClose method.
77559. Allow infinite precision
77560. Number of characters in string value, excluding '\0'
77561. Size of zName and its '\0' terminator
77562. One of OE_None, OE_Cascade etc.
77563. Wildcard of the form "?nnn". Convert "nnn" to an integer and ** use it as the variable number
77564. ** An instance of the following structure is used by the tree walker ** to count references to table columns in the arguments of an ** aggregate function, in order to implement the ** sqlite3FunctionThisSrc() routine.
77565. EVIDENCE-OF: R-49794-35026 Value is a big-endian 16-bit ** twos-complement integer.
77566. **comment:** ** Attempt to start a read transaction. This might fail due to a race or ** other transient condition. When that happens, it returns WAL_RETRY to ** indicate to the caller that it is safe to retry immediately. *** On success return SQLITE_OK. On a permanent failure (such an ** I/O error or an SQLITE_BUSY because another process is running ** recovery) return a positive error code. *** The useWal parameter is true to force the use of the WAL and disable ** the case where the WAL is bypassed because it has been completely ** checkpointed. If useWal==0 then this routine calls walIndexReadHdr() ** to make a copy of the wal-index header into pWal->hdr. If the ** wal-index header has changed, *pChanged is set to 1 (as an indication ** to the caller that the local paget cache is obsolete and needs to be ** flushed.) When useWal==1, the wal-index header is assumed to already ** be loaded and the pChanged parameter is unused. ***
 The caller must set the cnt parameter to the number of prior calls to ** this routine during the current read attempt that returned WAL_RETRY. ** This routine will

start taking more aggressive measures to clear the ** race conditions after multiple WAL_RETRY returns, and after an excessive ** number of errors will ultimately return SQLITE_PROTOCOL. The ** SQLITE_PROTOCOL return indicates that some other process has gone rogue ** and is not honoring the locking protocol. There is a vanishingly small ** chance that SQLITE_PROTOCOL could be returned because of a run of really ** bad luck when there is lots of contention for the wal-index, but that ** possibility is so small that it can be safely neglected, we believe. *** On success, this routine obtains a read lock on ** WAL_READ_LOCK(pWal->readLock). The pWal->readLock integer is ** in the range $0 \leq pWal->readLock < WAL_NREADER$. If pWal->readLock==(-1) ** that means the Wal does not hold any read lock. The reader must not ** access any database page that is modified by a WAL frame up to and ** including frame number aReadMark[pWal->readLock]. The reader will ** use WAL frames up to and including pWal->hdr.mxFrame if pWal->readLock>0 ** Or if pWal->readLock==0, then the reader will ignore the WAL ** completely and get all content directly from the database file. ** If the useWal parameter is 1 then the WAL will never be ignored and ** this routine will always set pWal->readLock>0 on success. ** When the read transaction is completed, the caller must release the ** lock on WAL_READ_LOCK(pWal->readLock) and set pWal->readLock to -1. *** This routine uses the nBackfill and aReadMark[] fields of the header ** to select a particular WAL_READ_LOCK() that strives to let the ** checkpoint process do as much work as possible. This routine might ** update values of the aReadMark[] array in the header, but if it does ** so it takes care to hold an exclusive lock on the corresponding ** WAL_READ_LOCK() while changing values.

label: code-design

77567. ** Add an OP_ParseSchema opcode. This routine is broken out from ** sqlite3VdbeAddOp4() since it needs to also needs to mark all btrees ** as having been used. *** The zWhere string must have been obtained from sqlite3_malloc(). ** This routine will take ownership of the allocated memory.
77568. If this is a deferred FK constraint, or a CASCADE or SET NULL ** action applies, then any foreign key violations caused by ** removing the parent key will be rectified by the action trigger. ** So do not set the "may-abort" flag in this case. *** Note 1: If the FK is declared "ON UPDATE CASCADE", then the ** may-abort flag will eventually be set on this statement anyway ** (when this function is called as part of processing the UPDATE ** within the action trigger). *** Note 2: At first glance it may seem like SQLite could simply omit ** all OP_FkCounter related scans when either CASCADE or SET NULL ** applies. The trouble starts if the CASCADE or SET NULL action ** trigger causes other triggers or action rules attached to the ** child table to fire. In these cases the fk constraint counters ** might be set incorrectly if any OP_FkCounter related scans are ** omitted.

77569. Finish the work of sqlite3DbMallocRawNN for the unusual and ** slower case when the allocation cannot be fulfilled using lookaside.
77570. ** PRAGMA encoding ** PRAGMA encoding = "utf-8"|"utf-16"|"utf-16le"|"utf-16be" *** In its first form, this pragma returns the encoding of the main ** database. If the database is not initialized, it is initialized now. *** The second form of this pragma is a no-op if the main database file ** has not already been initialized. In this case it sets the default ** encoding that will be used for the main database file if a new file ** is created. If an existing main database file is opened, then the ** default text encoding for the existing database is used. *** In all cases new databases created using the ATTACH command are ** created to use the same default text encoding as the main database. If ** the main database has not been initialized and/or created when ATTACH ** is executed, this is done before the ATTACH operation. *** In the second form this pragma sets the text encoding to be used in ** new database files created using this database handle. It is only ** useful if invoked immediately after the main database i

77571. Overwrite deleted content with zeros

77572. ** This is the value that walTryBeginRead returns when it needs to ** be retried.

77573. ** Attempt to obtain the exclusive WAL lock defined by parameters lockIdx and ** n. If the attempt fails and parameter xBusy is not NULL, then it is a ** busy-handler function. Invoke it and retry the lock until either the ** lock is successfully obtained or the busy-handler returns 0.

77574. ** Populate the buffer pointed to by zBufOut with nByte bytes of ** random data.

77575. 14

77576. Opcode: Null P1 P2 P3 * * * Synopsis: r[P2..P3]=NULL * * * Write a NULL into registers P2. If P3 greater than P2, then also write ** NULL into register P3 and every register in between P2 and P3. If P3 ** is less than P2 (typically P3 is zero) then only register P2 is ** set to NULL. *** If the P1 value is non-zero, then also set the MEM_Cleared flag so that ** NULL values will not compare equal even if SQLITE_NULLEQ is set on ** OP_Ne or OP_Eq.

77577. Index of column in pFrom

77578. **comment:** may be negative.

label: code-design

77579. 1400

77580. ** The number of bits in a Bitmask. "BMS" means "BitMask Size".

77581. Object iterator

77582. Initialize the rowset register to contain NULL. An SQL NULL is ** equivalent to an empty rowset. Or, create an ephemeral index ** capable of holding primary keys in the case of a WITHOUT ROWID. *** Also initialize regReturn to contain the address of the instruction ** immediately following the OP_Return at the bottom of the loop. This ** is required in a few obscure LEFT JOIN cases where control jumps ** over the top of the loop into the body of it. In this case the ** correct response for the end-of-loop code (the OP_Return) is to ** fall through to the next instruction, just as an OP_Next does if ** called on an uninitialized cursor.

77583. The default safety_level for the main database is FULL; for the temp ** database it is OFF. This matches the pager layer defaults.

77584. If necessary, grow the pIter->aRowidOffset[] array.

77585. This step function is used for both the min() and max() aggregates, ** the only difference between the two being that the sense of the ** comparison is inverted. For the max() aggregate, the ** sqlite3_user_data() function returns (void *)-1. For min() it ** returns (void *)db, where db is the sqlite3* database pointer. ** Therefore the next statement sets variable 'max' to 1 for the max() ** aggregate, or 0 for min().

77586. All data is stored on the current page. Populate the output ** variables to point into the body of the page object.

77587. Unique file ID for vxworks.

77588. ** Iterator pIter currently points to a valid entry (not EOF). This ** function appends the position list data for the current entry to ** buffer pBuf. It does not make a copy of the position-list size ** field.

77589. ** An instance of the following structure describes the event of a ** TRIGGER. "a" is the event type, one of TK_UPDATE, TK_INSERT, ** TK_DELETE, or TK_INSTEAD. If the event is of the form ** ** UPDATE ON (a,b,c) ** ** Then the "b" IdList records the list "a,b,c".

77590. refact ::= SET NULL

77591. Jump here if the UNIQUE constraint is satisfied

77592. SQLITE_ENABLE_UNKNOWN_SQL_FUNCTION

77593. Defined at the top of this function

77594. The next rowid or record number (different terms for the same ** thing) is obtained in a two-step algorithm. *** First we attempt to find the largest existing rowid and add one ** to that. But if the largest existing rowid is already the maximum ** positive integer, we have to fall through to the second ** probabilistic algorithm *** The second algorithm is to select a rowid at random and see if ** it already exists in the table. If it does not exist, we have ** succeeded. If the random rowid does exist, we select a new one ** and try again, up to 100 times.

77595. ** CAPI3REF: Reset A Prepared Statement Object ** METHOD: sqlite3_stmt ** ** The sqlite3_reset() function is called to reset a [prepared statement] ** object back to its initial state, ready to be re-executed. ** ^Any SQL statement variables that had values bound to them using ** the [sqlite3_bind_blob | sqlite3_bind_*()] API] retain their values. ** Use [sqlite3_clear_bindings()] to reset the bindings. *** ^The [sqlite3_reset(S)] interface resets the [prepared statement] S ** back to the beginning of its program. *** ^If the most recent call to [sqlite3_step(S)] for the ** [prepared statement] S returned [SQLITE_ROW] or [SQLITE_DONE], ** or if [sqlite3_step(S)] has never before been called on S, ** then [sqlite3_reset(S)] returns [SQLITE_OK]. *** ^If the most recent call to [sqlite3_step(S)] for the ** [prepared statement] S indicated an error, then ** [sqlite3_reset(S)] returns an appropriate [error code]. *** ^The [sqlite3_reset(S)] interface does not change the values ** of any [sqlite3_bind_blob|bindings] on the [prepared statement] S.

77596. The merge engine to advance to the next row

77597. Invoked at every commit.

77598. documents

77599. ** 2010 August 30 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

77600. ** Delete all information from a single table in the database. iTTable is ** the page number of the root of the table. After this routine returns, ** the root page is empty, but still exists. *** This routine will fail with SQLITE_LOCKED if there are any open ** read cursors on the table. Open write cursors are moved to the ** root of the table. *** If pnChange is not NULL, then table iTTable must be an intkey table. The ** integer value pointed to by pnChange is incremented by the number of ** entries in the table.

77601. The function name.

77602. Fetch the integer off the end of the index record

77603. Restriction (15)
77604. Automatic indexes
77605. BTree containing table to open
77606. Generated temporary filenames are always double-zero terminated ** for use by sqlite3_uri_parameter().
77607. Block id of first leaf node
77608. The Table object to which the virtual table belongs
77609. (303) oneselect ::= values
77610. VDBE cursor that is the main data repository
77611. True after sqlite3_declare_vtab() is called
77612. The X==Ei expression
77613. Search for a page near this one
77614. ** Open the file zPath. ** ** Previously, the SQLite OS layer used three functions in place of this ** one: ** ** sqlite3OsOpenReadWrite(); ** sqlite3OsOpenReadOnly(); ** sqlite3OsOpenExclusive(); ** ** These calls correspond to the following combinations of flags: ** ** ReadWrite() -> (READWRITE | CREATE) ** ReadOnly() -> (READONLY) ** OpenExclusive() -> (READWRITE | CREATE | EXCLUSIVE) ** ** The old OpenExclusive() accepted a boolean argument - "delFlag". If ** true, the file was configured to be automatically deleted when the ** file handle closed. To achieve the same effect using this new ** interface, add the DELETEONCLOSE flag to those specified above for ** OpenExclusive().
77615. Set the EP_Reduced, EP_TokenOnly, and EP_Static flags appropriately.
77616. Replacement expressions
77617. Indexed column is the rowid
77618. The main parser program. ** The first argument is a pointer to a structure obtained from ** "sqlite3Fts5ParserAlloc" which describes the current state of the parser. ** The second argument is the major token number. The third is ** the minor token. The fourth optional argument is whatever the ** user wants (and specified in the grammar) and is available for ** use by the action routines. ** ** Inputs: ** ** A pointer to the parser (an opaque structure.) ** The major token number. ** The minor token number. ** An option argument of a grammar-specified type. ** ** ** Outputs: ** None.
77619. Generate code to query the parent index for a matching parent ** key. If a match is found, jump to addrOk.
77620. Simple "PRAGMA locking_mode;" statement. This is a query for ** the current default locking mode (which may be different to ** the locking-mode of the main database).
77621. True if EXPLAIN present on SQL command
77622. 38
77623. ** sqlite3_test_control(SQLITE_TESTCTRL_BYTORDER); ** ** The integer returned reveals the byte-order of the computer on which ** SQLite is running: ** ** 1 big-endian, determined at run-time ** 10 little-endian, determined at run-time ** 432101 big-endian, determined at compile-time ** 123410 little-endian, determined at compile-time
77624. Path of the file to examine
77625. If nIncr is less than zero, then check at runtime if there are any ** outstanding constraints to resolve. If there are not, there is no need ** to check if deleting this row resolves any outstanding violations. ** ** Check if any of the key columns in the child table row are NULL. If ** any are, then the constraint is considered satisfied. No need to ** search for a matching row in the parent table.
77626. 'occurrences' values for current csr row
77627. Not a recursive common table expression.
77628. Deleted OK.
77629. The root page of the table to be locked
77630. New aRegion[] array
77631. This could happen with a network mount
77632. tcons ::= FOREIGN KEY LP eidlist RP REFERENCES nm eidlist_opt refargs defer_subclause_opt
77633. The "langid = ?" constraint, if any
77634. 990
77635. Space to hold the virtual machine's program
77636. If the statement journal is open and the page is not in it, ** then write the page into the statement journal.
77637. Store results here
77638. ** Called by the parser to compile a DETACH statement. ** ** DETACH pDname
77639. Address of the A<B subroutine
77640. ** Write out a single frame of the WAL
77641. Declare and initialize constant integer 'isDirect'. If the ** atomic-write optimization is enabled in this build, then isDirect ** is initialized to the value passed as the isDirectMode parameter ** to this function. Otherwise, it is always set to zero. ** ** The idea is that if the atomic-write optimization is not ** enabled at compile time, the compiler can omit the tests of ** 'isDirect' below, as well as the block enclosed in the ** "if(isDirect)" condition.
77642. Mutex type
77643. Size of vector pLeft
77644. ** CAPI3REF: Set the Last Insert Rowid value. ** METHOD: sqlite3 ** ** The sqlite3_set_last_insert_rowid(D, R) method allows the application to ** set the value returned by calling sqlite3_last_insert_rowid(D) to R ** without inserting a row into the database.
77645. ** Exit the recursive mutex on a Btree.
77646. At this point variable c contains the first character of the ** pattern string past the "**". Search in the input string for the ** first matching character and recursively continue the match from ** that point. ** ** For a case-insensitive search, set variable cx to be the same as ** c but in the other case and search the input string for either ** c or cx.
77647. testcase(IS_BIG_INT(iOffset)); // requires a 4GiB WAL file
77648. Maximum bytes of data to store
77649. Type passed as 4th argument to SegmentReaderIterate()
77650. ** The following macros are used to cast pointers to integers and ** integers to pointers. The way you do this varies from one compiler ** to the next, so we have developed the following set of #if statements ** to generate appropriate macros for a wide range of compilers. ** ** The correct "ANSI" way to do this is to use the intptr_t type. ** Unfortunately, that typedef is not available on all compilers, or ** if it is available, it requires an #include of specific headers ** that vary from one machine to the next. ** ** Ticket #3860: The llvm-gcc-4.2 compiler from Apple chokes on ** the ((void*)&((char*)0)[X]) construct. But MSVC chokes on ((void*)(X)). ** So we have to define the macros in different ways depending on the ** compiler.
77651. If the file format and encoding in the database have not been set, ** set them now.
77652. SQLITE_SO_ASC or SQLITE_SO_DESC
77653. Merge engine to return
77654. Deal with as much of this read request as possible by transferring ** data from the memory mapping using memcpy().
77655. Address of next instruction to return
77656. Root page of table to open
77657. We will need to create our own temporary table to hold the ** intermediate results.
77658. 241
77659. Mem.eSubtype is valid
77660. An index we are evaluating
77661. expr ::= ID|INDEXED LP distinct exprlist RP
77662. ** Information about the current state of the WAL file and where ** the next fsync should occur - passed from sqlite3WalFrames() into ** walWriteToLog().
77663. So that sizeof(Mem) is a multiple of 8
77664. Rows processed for all objects
77665. OUT: Error code
77666. Determine the number of integers in the buffer returned by this call.
77667. ** CAPI3REF: Compile-Time Authorization Callbacks ** METHOD: sqlite3 ** KEYWORDS: {authorizer callback} ** ** ^This routine registers an authorizer callback with a particular ** [database connection], supplied in the first argument. ** ^The authorizer callback is invoked as SQL statements are being compiled ** by [sqlite3_prepare()] or its variants [sqlite3_prepare_v2()], ** [sqlite3_prepare_v3()], [sqlite3_prepare16()], [sqlite3_prepare16_v2()], ** and

[sqlite3_prepare16_v3()]. ^At various ** points during the compilation process, as logic is being created ** to perform various actions, the authorizer callback is invoked to ** see if those actions are allowed. ^The authorizer callback should ** return [SQLITE_OK] to allow the action, [SQLITE_IGNORE] to disallow the ** specific action but allow the SQL statement to continue to be ** compiled, or [SQLITE_DENY] to cause the entire SQL statement to be ** rejected with an error. ^If the authorizer callback returns ** any value other than [SQLITE_IGNORE], [SQLITE_OK], or [SQLITE_DENY] ** then the [sqlite3_prepare_v2()] or equivalent call that triggered ** the authorizer will fail with an error message. *** When the callback returns [SQLITE_OK], that means the operation ** requested is ok. ^When the callback returns [SQLITE_DENY], the ** [sqlite3_prepare_v2()] or equivalent call that triggered the ** authorizer will fail with an error message explaining that ** access is denied. *** ^The first parameter to the authorizer callback is a copy of the third ** parameter to the sqlite3_set_authorizer() interface. ^The second parameter ** to the callback is an integer [SQLITE_COPY | action code] that specifies ** the particular action to be authorized. ^The third through sixth parameters ** to the callback are either NULL pointers or zero-terminated strings ** that contain additional details about the action to be authorized. ** Applications must always be prepared to encounter a NULL pointer in any ** of the third through the sixth parameters of the authorization callback. *** ^If the action code is [SQLITE_READ] ** and the callback returns [SQLITE_IGNORE] then the ** [prepared statement] statement is constructed to substitute ** a NULL value in place of the table column that would have ** been read if [SQLITE_OK] had been returned. The [SQLITE_IGNORE] ** return can be used to deny an untrusted user access to individual ** columns of a table. *** ^When a table is referenced by a [SELECT] but no column values are ** extracted from that table (for example in a query like ** "SELECT count(*) FROM tab") then the [SQLITE_READ] authorizer callback ** is invoked once for that table with a column name that is an empty string. ** ^If the action code is [SQLITE_DELETE] and the callback returns ** [SQLITE_IGNORE] then the [DELETE] operation proceeds but the ** [truncate optimization] is disabled and all rows are deleted individually. *** An authorizer is used when [sqlite3_prepare | preparing] ** SQL statements from an untrusted source, to ensure that the SQL statements ** do not try to access data they are not allowed to see, or that they do not ** try to execute malicious statements that damage the database. For ** example, an application may allow a user to enter arbitrary ** SQL queries for evaluation by a database. But the application does ** not want the user to be able to make arbitrary changes to the ** database. An authorizer could then be put in place while the ** user-entered SQL is being [sqlite3_prepare | prepared] that ** disallows everything except [SELECT] statements. *** Applications that need to process SQL from untrusted sources ** might also consider lowering resource limits using [sqlite3_limit()] ** and limiting database size using the [max_page_count] [PRAGMA] ** in addition to using an authorizer. *** ^Only a single authorizer can be in place on a database connection ** at a time. Each call to sqlite3_set_authorizer overrides the ** previous call.^A ^Disable the authorizer by installing a NULL callback. ** The authorizer is disabled by default. *** ^The authorizer callback must not do anything that will modify ** the database connection that invoked the authorizer callback. ** Note that [sqlite3_prepare_v2()] and [sqlite3_step()] both modify their ** database connections for the meaning of "modify" in this paragraph. *** ^When [sqlite3_prepare_v2()] is used to prepare a statement, the ** statement might be re-prepared during [sqlite3_step()] due to a ** schema change. Hence, the application should ensure that the ** correct authorizer callback remains in place during the [sqlite3_step()]. *** ^Note that the authorizer callback is invoked only during ** [sqlite3_prepare()] or its variants. Authorization is not ** performed during statement evaluation in [sqlite3_step()], unless ** as stated in the previous paragraph, sqlite3_step() invokes ** sqlite3_prepare_v2() to reprepare a statement after a schema change.

77668. All collating sequences

77669. **comment:** ** This function is called when the user executes the following statement: ** ** INSERT INTO <tbl><tbl> VALUES('rebuild'); ** ** The entire FTS index is discarded and rebuilt. If the table is one ** created using the content=xxx option, then the new index is based on ** the current contents of the xxx table.

label: code-design

77670. Cursor number of the first index

77671. **comment:** ** Macros for performance tracing. Normally turned off. Only works ** on i486 hardware.

label: code-design

77672. Mutex to access this object

77673. Used to catch return codes

77674. ** Process statements of the form: ** ** INSERT INTO table(table) VALUES('automerge=X'); ** ** where X is an integer. X==0 means to turn automerge off. X!=0 means ** turn it on. The setting is persistent.

77675. **comment:** Size of the temporary register block

label: code-design

77676. Language being queried

77677. ** If defined as non-zero, auto-vacuum is enabled by default. Otherwise ** it must be turned on for each database using "PRAGMA auto_vacuum = 1".

77678. True if this expression is entirely deferred

77679. Integer value of right operand

77680. Attach the hook to this database

77681. ** The functions querySharedCacheTableLock(), setSharedCacheTableLock(), ** and clearAllSharedCacheTableLocks() ** manipulate entries in the BtShared.pLock linked list used to store ** shared-cache table level locks. If the library is compiled with the ** shared-cache feature disabled, then there is only ever one user ** of each BtShared structure and so this locking is not necessary. ** So define the lock related functions as no-ops.

77682. ** Rtree virtual table module xDestroy method.

77683. Array of nCol best samples

77684. Index of column matching zCol

77685. Make sure this analysis did not leave any unref() pages.

77686. All unixShm objects pointing to this

77687. ** Implementation of SQL scalar function rbu_fossil_delta(). ** ** This function applies a fossil delta patch to a blob. Exactly two ** arguments must be passed to this function. The first is the blob to ** patch and the second the patch to apply. If no error occurs, this ** function returns the patched blob.

77688. Only interested in == results

77689. Hash slot index

77690. No more tokens here

77691. ** Deserialize cell iCell of node pNode. Populate the structure pointed ** to by pCell with the results.

77692. synopsis: if r[P3] in rowset(P1) goto P2

77693. 206

77694. The following loop runs once for each term in a compound-subquery ** flattening (as described above). If we are doing a different kind ** of flattening - a flattening other than a compound-subquery flattening - ** then this loop only runs once. *** This loop moves all of the FROM elements of the subquery into the ** the FROM clause of the outer query. Before doing this, remember ** the cursor number for the original outer query FROM element in ** iParent. The iParent cursor will never be used. Subsequent code ** will scan expressions looking for iParent references and replace ** those references with expressions that resolve to the subquery FROM ** elements we are now copying in.

77695. **comment:** ** This division contains definitions of sqlite3_io_methods objects that ** implement various file locking strategies. It also contains definitions ** of "finder" functions. A finder-function is used to locate the appropriate ** sqlite3_io_methods object for a particular database file. The pAppData ** field of the sqlite3_vfs objects are initialized to be pointers to ** the correct finder-function for that VFS. *** Most finder functions return a pointer to a fixed sqlite3_io_methods ** object. The only interesting finder-function is autolockIoFinder, which ** looks at the filesystem type and tries to guess the best locking ** strategy from that. *** For finder-function F, two objects are created: *** (1) The real finder-function named "FImpt()". *** (2) A constant pointer to this function named just "F". *** A pointer to the F pointer is used as the pAppData value for VFS ** objects. We have to do this instead of letting pAppData point ** directly at the finder-function since C90 rules prevent a void* ** from be cast into a function pointer. *** Each instance of this macro generates two objects: *** * A constant sqlite3_io_methods object call METHOD that has locking ** methods CLOSE, LOCK, UNLOCK, CKRESLOCK. *** * An I/O method finder function called FINDER that returns a pointer ** to the METHOD object in the previous bullet.

label: code-design

77696. ** Try to convert the type of a function argument or a result column ** into a numeric representation. Use either INTEGER or REAL whichever ** is appropriate. But only do the conversion if it is possible without ** loss of information and return the revised type of the argument.

77697. Pointer to <id> token

77698. **comment:** If we are just shaving the last few pages off the end of the ** cache, then there is no point in scanning the entire hash table. ** Only scan those hash slots that might contain pages that need to ** be removed.

label: code-design

77699. BLOB, TEXT, CLOB -> r=5 (approx 20 bytes)

77700. Name of fts3 table

77701. If shifting by a negative amount, shift in the other direction

77702. ** Added after 3.3.13
77703. Name of file to check
77704. MSVC is picky about pulling func ptrs from va lists. ** http://support.microsoft.com/kb/47961 ** sqlite3GlobalConfig.xLog = va_arg(ap, void*)(void*,int,const char*);
77705. WHERE_GROUPBY or _DISTINCTBY or _ORDERBY_LIMIT
77706. ** This is defined in tokenize.c. We just have to import the definition.
77707. Value to append
77708. xCreate/xConnect argument array
77709. ** The following value as a destructor means to use sqlite3DbFree(). ** The sqlite3DbFree() routine requires two parameters instead of the ** one parameter that destructors normally want. So we have to introduce ** this magic value that the code knows to handle differently. Any ** pointer will work here as long as it is distinct from SQLITE_STATIC ** and SQLITE_TRANSIENT.
77710. Size of zBlob in bytes
77711. The VDBE cursor
77712. in2: P2 is an input
77713. figure out how many UTF-8 characters are in zName
77714. ** The following are the meanings of bits in the Expr.flags field.
77715. True if this is a patchset
77716. Return code of various sub-routines
77717. ** Implementation of the scalar function fts3_tokenizer_internal_test(). ** This function is used for testing only, it is not included in the ** build unless SQLITE_TEST is defined. ** ** The purpose of this is to test that the fts3_tokenizer() function ** can be used as designed by the C-code in the queryTokenizer and ** registerTokenizer() functions above. These two functions are repeated ** in the README.tokenizer file as an example, so it is important to ** test them. ** ** To run the tests, evaluate the fts3_tokenizer_internal_test() scalar ** function with no arguments. An assert() will fail if a problem is ** detected. i.e.: ** ** SELECT fts3_tokenizer_internal_test(); **
77718. Lock the source database handle. The destination database ** handle is not locked in this routine, but it is locked in ** sqlite3_backup_step(). The user is required to ensure that no ** other thread accesses the destination handle for the duration ** of the backup operation. Any attempt to use the destination ** database connection while a backup is in progress may cause ** a malfunction or a deadlock.
77719. joinop ::= JOIN_KW nm nm JOIN
77720. transitive constraints
77721. The fields of the result
77722. ** CAPI3REF: Virtual Table Interface Configuration ** ** This function may be called by either the [xConnect] or [xCreate] method ** of a [virtual table] implementation to configure ** various facets of the virtual table interface. ** ** If this interface is invoked outside the context of an xConnect or ** xCreate virtual table method then the behavior is undefined. ** ** At present, there is only one option that may be configured using ** this function. (See [SQLITE_VTAB_CONSTRAINT_SUPPORT].) Further options ** may be added in the future.
77723. Path to target db
77724. Database file path
77725. Buffer containing page-index
77726. # include <time.h>
77727. The comparison operation
77728. First rowid on leaf iLeafPgno
77729. Flags to use to sync header writes
77730. ** Write VDBE code to erase table pTab and all associated indices on disk. ** Code to update the sqlite_master tables and internal schema definitions ** in case a root-page belonging to another table is moved by the btree layer ** is also added (this can happen with an auto-vacuum database).
77731. ** CAPI3REF: Virtual File System Objects ** ** A virtual filesystem (VFS) is an [sqlite3_vfs] object ** that SQLite uses to interact ** with the underlying operating system. Most SQLite builds come with a ** single default VFS that is appropriate for the host computer. ** New VFSes can be registered and existing VFSes can be unregistered. ** The following interfaces are provided. ** ** ^The sqlite3_vfs_find() interface returns a pointer to a VFS given its name. ** ^Names are case sensitive. ** ^Names are zero-terminated UTF-8 strings. ** ^If there is no match, a NULL pointer is returned. ** ^If zVfsName is NULL then the default VFS is returned. ** ** ^New VFSes are registered with sqlite3_vfs_register(). ** ^Each new VFS becomes the default VFS if the makeDflt flag is set. ** ^The same VFS can be registered multiple times without injury. ** ^To make an existing VFS into the default VFS, register it again ** with the makeDflt flag set. If two different VFSes with the ** same name are registered, the behavior is undefined. If a ** VFS is registered with a name that is NULL or an empty string, ** then the behavior is undefined. ** ** ^Unregister a VFS with the sqlite3_vfs_unregister() interface. ** ^^(If the default VFS is unregistered, another VFS is chosen as ** the default. The choice for the new VFS is arbitrary.)^
77732. Used when p4type is P4_VTAB
77733. Advance iterator at least this far
77734. Configuration object
77735. Flush the last leaf page to disk. Set the output segment b-tree height ** and last leaf page number at the same time.
77736. If a rowid value was supplied, check if it is already present in ** the table. If so, the constraint has failed.
77737. ** Mark every prepared statement associated with a database connection ** as expired. ** ** An expired statement means that recompilation of the statement is ** recommend. Statements expire when things happen that make their ** programs obsolete. Removing user-defined functions or collating ** sequences, or changing an authorization function are the types of ** things that make prepared statements obsolete.
77738. 134
77739. Database page-size
77740. The name context cannot be NULL.
77741. Make sure the encoding is UTF-8
77742. Columns in this table that point to other table
77743. Increment our pretend rowid value.
77744. ** The contents of the "structure" record for each index are represented ** using an Fts5Structure record in memory. Which uses instances of the ** other Fts5StructureXXX types as components.
77745. pHeap
77746. **comment:** ** Macros to determine whether the machine is big or little endian, ** and whether or not that determination is run-time or compile-time. ** ** For best performance, an attempt is made to guess at the byte-order ** using C-preprocessor macros. If that is unsuccessful, or if ** -DSQLITE_BYTEORDER=0 is set, then byte-order is determined ** at run-time.
label: code-design
77747. Session changeset-apply context
77748. Take advantage of short-circuit false optimization for AND
77749. The rowid that might be changing
77750. 210
77751. Do not use prefix index
77752. **comment:** VFS to use if no "vfs=xxx" query option
label: code-design
77753. ** The snippet() and offsets() functions both return text values. An instance ** of the following structure is used to accumulate those values while the ** functions are running. See fts3StringAppend() for details.
77754. Number of ORDER BY clause terms
77755. One table in the FROM clause
77756. The virtual table being constructed
77757. Linked list of auxdata allocations
77758. Obtain an exclusive lock on all byte in the locking range not already ** locked by the caller. The caller is guaranteed to have locked the ** WAL_WRITE_LOCK byte, and may have also locked the WAL_CKPT_LOCK byte. ** If successful, the same bytes that are locked here are unlocked before ** this function returns.
77759. Number of entries with this hash

77760. True if holding the unixShmNode mutex
77761. expr ::= nm DOT nm
77762. Used to iterate through VMs
77763. INSERT or UPDATE operations
77764. ** Open a handle to begin or resume an RBU VACUUM operation.
77765. Check all the tables.
77766. Boolean parameters. See BTS_* macros below
77767. Fts5 context
77768. Read input from all segments in the input level
77769. ** This object represents a single thread of control in a sort operation. ** Exactly VdbeSorter.nTask instances of this object are allocated ** as part of each VdbeSorter object. Instances are never allocated any ** other way. VdbeSorter.nTask is set to the number of worker threads allowed ** (see SQLITE_CONFIG_WORKER_THREADS) plus one (the main thread). Thus for ** single-threaded operation, there is exactly one instance of this object ** and for multi-threaded operation there are two or more instances. ** ** Essentially, this structure contains all those fields of the VdbeSorter ** structure for which each thread requires a separate instance. For example, ** each thread requires its own UnpackedRecord object to unpack records in ** as part of comparison operations. ** ** Before a background thread is launched, variable bDone is set to 0. Then, ** right before it exits, the thread itself sets bDone to 1. This is used for ** two purposes: ** ** 1. When flushing the contents of memory to a level-0 PMA on disk, to ** attempt to select a SortSubtask for which there is not already an ** active background thread (since doing so causes the main thread ** to block until it finishes). ** ** 2. If SQLITE_DEBUG_SORTER_THREADS is defined, to determine if a call ** to sqlite3ThreadJoin() is likely to block. Cases that are likely to ** block provoke debugging output. ** ** In both cases, the effects of the main thread seeing (bDone==0) even ** after the thread has finished are not dire. So we don't worry about ** memory barriers and such here.
77770. If pTab is really a view, make sure it has been initialized.
77771. WinRT has no way to convert a relative path to an absolute one.
77772. First frame that may be overwritten
77773. OUT: New iterator object
77774. No match was found. We will make a new file ID
77775. when_clause ::= WHEN expr
77776. Opcode: Rowid P1 P2 * * * ** Synopsis: r[P2]=rowid ** ** Store in register P2 an integer which is the key of the table entry that ** P1 is currently point to. ** ** P1 can be either an ordinary table or a virtual table. There used to ** be a separate OP_VRowid opcode for use with virtual tables, but this ** one opcode now works for both table types.
77777. Allocate a MultiSegReader for each token in the expression.
77778. Allocate additional space if needed
77779. Array containing doclist (or NULL)
77780. 244
77781. Pointer to cursor object to return
77782. ** This routine is the only routine in this file with external ** linkage. ** ** Populate the low-level memory allocation function pointers in ** sqlite3GlobalConfig.m with pointers to the routines in this file. The ** arguments specify the block of memory to manage. ** ** This routine is only called by sqlite3_config(), and therefore ** is not required to be threadsafe (it is not).
77783. Name context of this SELECT
77784. Variables related to SQLITE_CONFIG_PAGECACHE settings. The ** szSlot, nSlot, pStart, pEnd, nReserve, and isInit values are all ** fixed at sqlite3_initialize() time and do not require mutex protection. ** The nFreeSlot and pFree values do require mutex protection.
77785. EVIDENCE-OF: R-29851-52272 Value is a big-endian 64-bit ** two-complement integer.
77786. If needSyncPnno is non-zero, then the journal file needs to be ** sync(jed before any data is written to database file page needSyncPnno. ** Currently, no such page exists in the page-cache and the ** "is journaled" bitvec flag has been set. This needs to be remedied by ** loading the page into the pager-cache and setting the PGHDR_NEED_SYNC ** flag. ** ** If the attempt to load the page into the page-cache fails, (due ** to a malloc() or IO failure), clear the bit in the pInJournal[] ** array. Otherwise, if the page is loaded and written again in ** this transaction, it may be written to the database file before ** it is synced into the journal file. This way, it may end up in ** the journal file twice, but that is not a problem.
77787. Resolve all names in the ORDER BY term expression
77788. ** Seek to the offset in id->offset then read cnt bytes into pBuf. ** Return the number of bytes actually read. Update the offset. ** ** To avoid stomping the errno value on a failed write the lastErrno value ** is set before returning.
77789. ** Implementation of xBegin() method.
77790. Rowid for this entry
77791. **comment:** ** 2004 April 6 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file implements an external (disk-based) database using BTrees. ** For a detailed discussion of BTrees, refer to ** ** Donald E. Knuth, THE ART OF COMPUTER PROGRAMMING, Volume 3: ** "Sorting And Searching", pages 473-480. Addison-Wesley ** Publishing Company, Reading, Massachusetts. ** ** The basic idea is that each page of the file contains N database ** entries and N+1 pointers to subpages. ** ** ----- ** | Ptr(0) | Key(0) | Ptr(1) | Key(1) | ... | Key(N-1) | Ptr(N) | ** ----- ** All of the keys on the page that Ptr(0) points to have values less ** than Key(0). All of the keys on page Ptr(1) and its subpages have ** values greater than Key(0) and less than Key(1). All of the keys ** on Ptr(N) and its subpages have values greater than Key(N-1). And ** so forth. ** ** Finding a particular key requires reading O(log(M)) pages from the ** disk where M is the number of entries in the tree. ** ** In this implementation, a single file can hold one or more separate ** BTrees. Each BTree is identified by the index of its root page. The ** key and data for any entry are combined to form the "payload". A ** fixed amount of payload can be carried directly on the database ** page. If the payload is larger than the preset amount then surplus ** bytes are stored on overflow pages. The payload for an entry ** and the preceding pointer are combined to form a "Cell". Each ** page has a small header which contains the Ptr(N) pointer and other ** information such as the size of key and data. ** ** FORMAT DETAILS ** ** The file is divided into pages. The first page is called page 1, ** the second is page 2, and so forth. A page number of zero indicates ** "no such page". The page size can be any power of 2 between 512 and 65536. ** Each page can be either a btree page, a freelist page, an overflow ** page, or a pointer-map page. ** ** The first page is always a btree page. The first 100 bytes of the first ** page contain a special header (the "file header") that describes the file. ** The format of the file header is as follows: ** ** OFFSET SIZE DESCRIPTION ** 0 16 Header string: "SQLite format 3|000" ** 16 2 Page size in bytes. (1 means 65536) ** 18 1 File format write version ** 19 1 File format read version ** 20 1 Bytes of unused space at the end of each page ** 21 1 Max embedded payload fraction (must be 64) ** 22 1 Min embedded payload fraction (must be 32) ** 23 1 Min leaf payload fraction (must be 32) ** 24 4 File change counter ** 28 4 Reserved for future use ** 32 4 First freelist page ** 36 4 Number of freelist pages in the file ** 40 60 15 4-byte meta values passed to higher layers ** ** 40 4 Schema cookie ** 44 4 File format of schema layer ** 48 4 Size of page cache ** 52 4 Largest root-page (auto/incr_vacuum) ** 56 4 1=UTF-8 2=UTF16le 3=UTF16be ** 60 4 User version ** 64 4 Incremental vacuum mode ** 68 4 Application-ID ** 72 20 unused ** 92 4 The version-valid-for number ** 96 4 SQLITE_VERSION_NUMBER ** ** All of the integer values are big-endian (most significant byte first). ** ** The file change counter is incremented when the database is changed ** This counter allows other processes to know when the file has changed ** and thus when they need to flush their cache. ** ** The max embedded payload fraction is the amount of the total usable ** space in a page that can be consumed by a single cell for standard ** B-tree (non-LEAFDATA) tables. A value of 255 means 100%. The default ** is to limit the maximum cell size so that at least 4 cells will fit ** on one page. Thus the default max embedded payload fraction is 64. ** ** If the payload for a cell is larger than the max payload, then extra ** payload is spilled to overflow pages. Once an overflow page is allocated, ** as many bytes as possible are moved into the overflow pages without letting ** the cell size drop below the min embedded payload fraction. ** ** The min leaf payload fraction is like the min embedded payload fraction ** except that it applies to leaf nodes in a LEAFDATA tree. The maximum ** payload fraction for a LEAFDATA tree is always 100% (or 255) and it ** not specified in the header. ** ** Each btree pages is divided into three sections: The header, the ** cell pointer array, and the cell content area. Page 1 also has a 100-byte ** file header that occurs before the page header. ** ** ----- ** | file header | 100 bytes. Page 1 only. ** | ----- | ** | page header | 8 bytes for leaves. 12 bytes for interior nodes ** |-----| ** | cell pointer | 2 bytes per cell. Sorted order. ** | array | | Grows downward ** | | v ** |-----| ** | unallocated | ** | space | ** |-----| ^ Grows upwards ** | cell content | | Arbitrary order interspersed with freeblocks. ** | area | | and free space fragments. ** |-----| ** ** The page headers looks like this. ** ** OFFSET SIZE DESCRIPTION ** 0 1 Flags. 1: intkey, 2: zerodata, 4: leafdata, 8: leaf ** 1 2 byte offset to the first freeblock ** 3 2 number of cells on this page ** 5 2 first byte of the cell content area ** 7 1 number of fragmented free bytes ** 8 4 Right child (the Ptr(N) value). Omitted on leaves. ** ** The flags define the format of this btree page. The leaf flag means that ** this page has no children. The zerodata flag means that this page carries ** only keys and no data. The intkey flag means that the key is an integer ** which is stored in the key size entry of the cell header rather than in ** the payload area. ** ** The cell pointer array begins on the first byte after the page header. ** The cell pointer array

contains zero or more 2-byte numbers which are ** offsets from the beginning of the page to the cell content in the cell ** content area. The cell pointers occur in sorted order. The system strives ** to keep free space after the last cell pointer so that new cells can ** be easily added without having to defragment the page. ** ** Cell content is stored at the very end of the page and grows toward the ** beginning of the page. ** ** Unused space within the cell content area is collected into a linked list of ** freeblocks. Each freeblock is at least 4 bytes in size. The byte offset ** to the first freeblock is given in the header. Freeblocks occur in ** increasing order. Because a freeblock must be at least 4 bytes in size, ** any group of 3 or fewer unused bytes in the cell content area cannot ** exist on the freeblock chain. A group of 3 or fewer free bytes is called ** a fragment. The total number of bytes in all fragments is recorded. ** in the page header at offset 7. ** ** SIZE DESCRIPTION ** 2 Byte offset of the next freeblock ** 2 Bytes in this freeblock ** ** Cells are of variable length. Cells are stored in the cell content area at ** the end of the page. Pointers to the cells are in the cell pointer array ** that immediately follows the page header. Cells is not necessarily ** contiguous or in order, but cell pointers are contiguous and in order. ** ** Cell content makes use of variable length integers. A variable ** length integer is 1 to 9 bytes where the lower 7 bits of each ** byte are used. The integer consists of all bytes that have bit 8 set and ** the first byte with bit 8 clear. The most significant byte of the integer ** appears first. A variable-length integer may not be more than 9 bytes long. ** As a special case, all 8 bytes of the 9th byte are used as data. This ** allows a 64-bit integer to be encoded in 9 bytes. ** ** 0x00 becomes 0x00000000 ** 0x7f becomes 0x00000007f ** 0x81 0x00 becomes 0x00000080 ** 0x82 0x00 becomes 0x00000100 ** 0x80 0x7f becomes 0x0000007f ** 0x8a 0x91 0xd1 0xac 0x78 becomes 0x12345678 ** 0x81 0x81 0x81 0x81 0x01 becomes 0x10204081 ** ** Variable length integers are used for rowids and to hold the number of ** bytes of key and data in a btree cell. ** ** The content of a cell looks like this: ** ** SIZE DESCRIPTION ** 4 Page number of the left child. Omitted if leaf flag is set. ** var Number of bytes of data. Omitted if the zerodata flag is set. ** var Number of bytes of key. Or the key itself if intkey flag is set. ** * Payload ** 4 First page of the overflow chain. Omitted if no overflow ** ** Overflow pages form a linked list. Each page except the last is completely ** filled with data (pagesize - 4 bytes). The last page can have as little ** as 1 byte of data. ** ** SIZE DESCRIPTION ** 4 Page number of next overflow page ** * Data ** ** Freelist pages come in two subtypes: trunk pages and leaf pages. The ** file header points to the first in a linked list of trunk page. Each trunk ** page points to multiple leaf pages. The content of a leaf page is ** unspecified. A trunk page looks like this: ** ** SIZE DESCRIPTION ** 4 Page number of next trunk page ** 4 Number of leaf pointers on this page ** * zero or more pages numbers of leaves

label: code-design

77792. Re-enable the lookaside buffer, if it was disabled earlier.

77793. Error message text, if an error occurs

77794. ** Free a PendingList object allocated by fts3PendingListAppend().

77795. defined SCANSTAT or STAT4 or ESTIMATED_ROWS

77796. If this is being called to read the first page of the target ** database as part of an rbu vacuum operation, synthesize the ** contents of the first page if it does not yet exist. Otherwise, ** SQLite will not check for a *-wal file.

77797. 130

77798. pPager->stmtOpen = 0;

77799. ** This (sqlite3EndBenignMalloc()) is called by SQLite code to indicate that ** subsequent malloc failures are benign. A call to sqlite3EndBenignMalloc() ** indicates that subsequent malloc failures are non-benign.

77800. True to enable tracing

77801. ** This function is called from within a pre-update callback to retrieve ** a field of the row currently being updated or deleted.

77802. ** Structures used by the tokenizer interface. When a new tokenizer ** implementation is registered, the caller provides a pointer to ** an

sqlite3_tokenizer_module containing pointers to the callback ** functions that make up an implementation. ** ** When an fts3 table is created, it passes any arguments passed to ** the tokenizer clause of the CREATE VIRTUAL TABLE statement to the ** sqlite3_tokenizer_module.xCreate() function of the requested tokenizer ** implementation. The xCreate() function in turn returns an ** sqlite3_tokenizer structure representing the specific tokenizer to ** be used for the fts3 table (customized by the tokenizer clause arguments). ** ** To tokenize an input buffer, the sqlite3_tokenizer_module.xOpen() ** method is called. It returns an sqlite3_tokenizer_cursor object ** that may be used to tokenize a specific input buffer based on ** the tokenization rules supplied by a specific sqlite3_tokenizer ** object.

77803. 40

77804. 0x06

77805. ** Invalidate temp storage, either when the temp storage is changed ** from default, or when 'file' and the temp_store_directory has changed

77806. There is no constraint to check

77807. clearCell never fails when nLocal==nPayload

77808. Corresponding column number in the eq-class

77809. Number of phrases in expression

77810. **comment:** EVIDENCE-OF: R-55594-21030 The SQLITE_CONFIG_MALLOC option takes a ** single argument which is a pointer to an instance of the ** sqlite3_mem_methods structure. The argument specifies alternative ** low-level memory allocation routines to be used in place of the memory ** allocation routines built into SQLite.

label: code-design

77811. ** The default lookaside-configuration, the format "SZ,N". SZ is the ** number of bytes in each lookaside slot (should be a multiple of 8) ** and N is the number of slots. The lookaside-configuration can be ** changed as start-time using sqlite3_config(SQLITE_CONFIG_LOOKASIDE) ** or at run-time for an individual database connection using ** sqlite3_db_config(db, SQLITE_DBCONFIG_LOOKASIDE);

77812. The ORDER BY term is an integer constant. Again, set the column ** number so that sqlite3ResolveOrderGroupBy() will convert the ** order-by term to a copy of the result-set expression

77813. ** Release the BtShared mutex associated with B-Tree handle p and ** clear the p->locked boolean.

77814. ** Check every term in the ORDER BY or GROUP BY clause pOrderBy of ** the SELECT statement pSelect. If any term is reference to a ** result set expression (as determined by the ExprList.a.u.x.iOrderByCol ** field) then convert that term into a copy of the corresponding result set ** column. ** ** If any errors are detected, add an error message to pParse and ** return non-zero. Return zero if no errors are seen.

77815. Processing a sub-WHERE as part of ** the OR optimization

77816. **comment:** ** This routine generates code to finish the INSERT or UPDATE operation ** that was started by a prior call to sqlite3GenerateConstraintChecks. ** A consecutive range of registers starting at regNewData contains the ** rowid and the content to be inserted. ** ** The arguments to this routine should be the same as the first six ** arguments to sqlite3GenerateConstraintChecks.

label: code-design

77817. Opcode: Int64 * P2 * P4 * ** Synopsis: r[P2]=P4 ** ** P4 is a pointer to a 64-bit integer value. ** Write that value into register P2.

77818. Mask of all possible WO_* values

77819. ** Iterator pIter must point to an SQLITE_INSERT entry. This function ** transfers new.* values from the current iterator entry to statement ** pStmt. The table being inserted into has nCol columns. ** ** New.* value \$i from the iterator is bound to variable (\$i+1) of ** statement pStmt. If parameter abPK is NULL, all values from 0 to (nCol-1) ** are transferred to the statement. Otherwise, if abPK is not NULL, it points ** to an array nCol elements in size. In this case only those values for ** which abPK[\$i] is true are read from the iterator and bound to the ** statement. ** ** An SQLite error code is returned if an error occurs. Otherwise, SQLITE_OK.

77820. Expression list to search

77821. ** A hash table for built-in function definitions. (Application-defined ** functions use a regular table table from hash.h.) ** ** Hash each FuncDef structure into one of the FuncDefHash.a[] slots. ** Collisions are on the FuncDef.u.pHash chain.

77822. Release the checkpointer and writer locks

77823. Already gone?

77824. Next chunk in the journal

77825. ** Open a new statvfs cursor.

77826. IMPLEMENTATION-OF: R-46199-30249 SQLite guarantees that the second ** argument to xRealloc is always a value returned by a prior call to ** xRoundup.

77827. Allocate the space required for the Stat4Accum object

77828. Allocate required registers.

77829. Uses the skip-scan algorithm

77830. All pages have been processed exactly once

77831. Named POSIX semaphore

77832. Did we have an exclusive lock?

77833. IN/OUT: error code

77834. Trigger definition to return
77835. SQLITE_DEBUG
77836. Prepared statements under construction
77837. Read all frames from the log file.
77838. Call xBestIndex once for each distinct value of (prereqRight & ~mPrereq) ** in the set of terms that apply to the current virtual table.
77839. Integer 1
77840. Index object
77841. ** Implementation of the like() SQL function. This function implements ** the build-in LIKE operator. The first argument to the function is the ** pattern and the second argument is the string. So, the SQL statements: ** ** A LIKE B ** ** is implemented as like(B,A). ** ** This same function (with a different compareInfo structure) computes ** the GLOB operator.
77842. SQLITE OMIT_XFER_OPT
77843. Evaluate the expression and insert it into the temp table
77844. The next sections is a series of control #defines. ** YYCODETYPE is the data type used to store the integer codes ** that represent terminal and non-terminal symbols. ** "unsigned char" is used if there are fewer than ** 256 symbols. Larger types otherwise. ** YYNOCODE is a number of type YYCODETYPE that is not used for ** any terminal or nonterminal symbol. ** YYFALLBACK If defined, this indicates that one or more tokens ** (also known as: "terminal symbols") have fall-back ** values which should be used if the original symbol ** would not parse. This permits keywords to sometimes ** be used as identifiers, for example. ** YYACTIONTYPE is the data type used for "action codes" - numbers ** that indicate what to do in response to the next ** token. ** sqlite3ParserTOKENTYPE is the data type used for minor type for terminal ** symbols. Background: A "minor type" is a semantic ** value associated with a terminal or non-terminal ** symbols. For example, for an "ID" terminal symbol, ** the minor type might be the name of the identifier. ** Each non-terminal can have a different minor type. ** Terminal symbols all have the same minor type, though. ** This macros defines the minor type for terminal ** symbols. ** YYMINORTYPE is the data type used for all minor types. ** This is typically a union of many types, one of ** which is sqlite3ParserTOKENTYPE. The entry in the union ** for terminal symbols is called "yy0". ** YYSTACKDEPTH is the maximum depth of the parser's stack. If ** zero the stack is dynamically sized using realloc() ** sqlite3ParserARG_SDECL A static variable declaration for the %extra_argument ** sqlite3ParserARG_PDECL A parameter declaration for the %extra_argument ** sqlite3ParserARG_STORE Code to store %extra_argument into yyParser ** sqlite3ParserARG_FETCH Code to extract %extra_argument from yyParser ** YYERRORSYMBOL is the code number of the error symbol. If not ** defined, then do no error processing. ** YYNSTATE the combined number of states. ** YYNRULE the number of rules in the grammar ** YY_MAX_SHIFT Maximum value for shift actions ** YY_MIN_SHIFTREDUCE Minimum value for shift-reduce actions ** YY_MAX_SHIFTREDUCE Maximum value for shift-reduce actions ** YY_MIN_REDUCE Maximum value for reduce actions ** YY_ERROR_ACTION The yy_action[] code for syntax error ** YY_ACCEPT_ACTION The yy_action[] code for accept ** YY_NO_ACTION The yy_action[] code for no-op
77845. Custom rank function
77846. Routine to fetch a patch
77847. 60
77848. Temporary VM
77849. If we reach this point, that means the index pIdx is usable
77850. Offset to the start of the cell content area
77851. ** Return the memory allocator mutex. sqlite3_status() needs it.
77852. "DELETE FROM %_idx WHERE segid=?
77853. ** The main routine for background threads that write level-0 PMAs.
77854. Number of left-most columns to skip
77855. A MATCH constraint. Use a full-text search. ** ** If there is more than one MATCH constraint available, use the first ** one encountered. If there is both a MATCH constraint and a direct ** rowid/docid lookup, prefer the MATCH strategy. This is done even ** though the rowid/docid lookup is faster than a MATCH query, selecting ** it would lead to an "unable to use function MATCH in the requested ** context" error.
77856. 0x0d
77857. ** CAPI3REF: Flags For File Open Operations ** ** These bit values are intended for use in the ** 3rd parameter to the [sqlite3_open_v2()] interface and ** in the 4th parameter to the [sqlite3_vfs.xOpen] method.
77858. same as TK_MINUS, in1, in2, out3
77859. The VFS used to create pDbFd
77860. indexed_opt ::= NOT INDEXED
77861. 295
77862. Size of the binary record
77863. extern "C"
77864. Number of bytes in term suffix
77865. ** Log a I/O error retry episode.
77866. The last token parsed
77867. **comment:** Human readable error text
label: code-design
77868. ** During code generation of statements that do inserts into AUTOINCREMENT ** tables, the following information is attached to the Table.u.autoInc.p ** pointer of each autoincrement table to record some side information that ** the code generator needs. We have to keep per-table autoincrement ** information in case inserts are done within triggers. Triggers do not ** normally coordinate their activities, but we do need to coordinate the ** loading and saving of autoincrement information.
77869. This case when the file is being statically linked into the ** application
77870. ** Reset the SQL statement passed as the first argument. Return a copy ** of the value returned by sqlite3_reset(). ** ** If an error has occurred, then set *pzErrmsg to point to a buffer ** containing an error message. It is the responsibility of the caller ** to eventually free this buffer using sqlite3_free().
77871. colsetlist ::= colsetlist STRING
77872. in_op ::= IN
77873. maxLocal is the maximum amount of payload to store locally for ** a cell. Make sure it is small enough so that at least minFanout ** cells can will fit on one page. We assume a 10-byte page header. ** Besides the payload, the cell must store: ** 2-byte pointer to the cell ** 4-byte child pointer ** 9-byte nKey value ** 4-byte nData value ** 4-byte overflow page pointer ** So a cell consists of a 2-byte pointer, a header which is as much as ** 17 bytes long, 0 to N bytes of payload, and an optional 4 byte overflow ** page pointer.
77874. The page that contains the Cell
77875. Error message prefix
77876. Set the right-child pointer of pParent to point to the new page.
77877. RIGHT
77878. OFFSET => ID
77879. A virtual meta-value
77880. Database Name NULL
77881. The ORDER BY clause
77882. first lock byte
77883. The cached entry matches, so return it
77884. expr
77885. **comment:** Temporary use register
label: code-design
77886. all negative values become -1.
77887. ** date(TIMESTRING, MOD, MOD, ...) ** ** Return YYYY-MM-DD
77888. If an error occurred while creating or writing to the file, restore ** the original before returning. This way, SQLite uses the in-memory ** journal data to roll back changes made to the internal page-cache ** before this function was called.
77889. 1140
77890. Next sub-program already visited

77891. Authorization context
77892. Load new statistics out of the sqlite_stat1 table
77893. 141
77894. **comment:** ** This function does not contribute anything to the operation of SQLite. ** it is sometimes activated temporarily while debugging code responsible ** for setting pointer-map entries.
label: code-design
77895. If there is an ORDER BY clause, then we need to sort the results ** and send them to the callback one by one.
77896. This case when the file really is being compiled as a loadable ** extension
77897. Insert the new index entries and the new record.
77898. Set default precision
77899. Add OP_CursorHint opcodes
77900. ***** End of mutex.h *****
77901. Array indicating UPDATEd columns (or 0)
77902. A source cursor for SORTED_MATCH
77903. Form 3: Analyze the table or index named as an argument
77904. Number of non-wildcard prefix characters
77905. The table or view or trigger name is passed to this routine via tokens ** pName1 and pName2. If the table name was fully qualified, for example: ** ** CREATE TABLE xxx.yyy (...); ** ** Then pName1 is set to "xxx" and pName2 "yyy". On the other hand if ** the table name is not fully qualified, i.e.: ** ** CREATE TABLE yyy(...); ** ** Then pName1 is set to "yyy" and pName2 is "". ** ** This routine sets the *ppUnqual pointer to point at the token (pName1 or ** pName2) that stores the unqualified table name. The index of the ** database "xxx" is returned.
77906. 2*nField extra array elements allocated for aType[], beyond the one ** static element declared in the structure. nField total array slots for ** aType[] and nField+1 array slots for aOffset[]
77907. Number of bytes in buffer pA
77908. If we are doing a normal write to a database file (as opposed to ** doing a hot-journal rollback or a write to some file other than a ** normal database file) then record the fact that the database ** has changed. If the transaction counter is modified, record that ** fact too.
77909. True for a leaf node
77910. Stat4Accum.anDLT
77911. **comment:** ** This function allocates a new parser. ** The only argument is a pointer to a function which works like ** malloc. ** ** Inputs: ** A pointer to the function used to allocate memory. ** ** Outputs: ** A pointer to a parser. This pointer is used in subsequent calls ** to sqlite3Fts5Parser and sqlite3Fts5ParserFree.
label: code-design
77912. Index in some Table.aCol[] of a column named zName
77913. WRITEPOSLISTSIZE
77914. ** Check schema cookies in all databases. If any cookie is out ** of date set pParse->rc to SQLITE_SCHEMA. If all schema cookies ** make no changes to pParse->rc.
77915. Jump here if a comparison is not true in step 6
77916. Check that: ** ** a) The incremental merge object is configured to use the ** right task, and ** b) If it is using task (nTask-1), it is configured to run ** in single-threaded mode. This is important, as the ** root merge (INCRINIT_ROOT) will be using the same task ** object.
77917. ** Arguments aLeft and aRight both point to buffers containing change ** records with nCol columns. This function "merges" the two records into ** a single records which is written to the buffer at *paOut. *paOut is ** then set to point to one byte after the last byte written before ** returning. ** ** The merging of records is done as follows: For each column, if the ** aRight record contains a value for the column, copy the value from ** their. Otherwise, if aLeft contains a value, copy it. If neither ** record contains a value for a given column, then neither does the ** output record.
77918. ** Report the current page size and number of reserved bytes back ** to the codec.
77919. Write minimum integer primary key here
77920. Table leaf node
77921. If eType==EXTERNAL, root of PK index
77922. Jump to here if not distinct
77923. If the db handle must hold the connection handle mutex here. ** Otherwise the read (and possible write) of db->mallocFailed ** is unsafe, as is the call to sqlite3Error().
77924. same as TK_FLOAT, synopsis: r[P2]=P4
77925. Build up text to return here
77926. High-water mark of the stack
77927. ** Return the total number of pages in the source database as of the most ** recent call to sqlite3_backup_step().
77928. ** Function assertTruncateConstraint(pPager) checks that one of the ** following is true for all dirty pages currently in the page-cache: ** ** a) The page number is less than or equal to the size of the ** current database image, in pages, OR ** ** b) if the page content were written at this time, it would not ** be necessary to write the current content out to the sub-journal ** (as determined by function subjRequiresPage()). ** ** If the condition asserted by this function were not true, and the ** dirty page were to be discarded from the cache via the pagerStress() ** routine, pagerStress() would not write the current page content to ** the database file. If a savepoint transaction were rolled back after ** this happened, the correct behavior would be to restore the current ** content of the page. However, since this content is not present in either ** the database file or the portion of the rollback journal and ** sub-journal rolled back the content could not be restored and the ** database image would become corrupt. It is therefore fortunate that ** this circumstance cannot arise.
77929. The pointer map data
77930. 110
77931. ROLLBACK => ID
77932. ** Register a geometry callback named zGeom that can be used as part of an ** R-Tree geometry query as follows: ** ** SELECT ... FROM <rtree> WHERE <rtree col> MATCH \$zGeom(... params ...)
77933. **comment:** ** Array apCell[] contains nCell pointers to b-tree cells. Array szCell ** contains the size in bytes of each such cell. This function attempts to ** add the cells stored in the array to page pPg. If it cannot (because ** the page needs to be defragmented before the cells will fit), non-zero ** is returned. Otherwise, if the cells are added successfully, zero is ** returned. ** ** Argument pCellptr points to the first entry in the cell-pointer array ** (part of page pPg) to populate. After cell apCell[0] is written to the ** page body, a 16-bit offset is written to pCellptr. And so on, for each ** cell in the array. It is the responsibility of the caller to ensure ** that it is safe to overwrite this part of the cell-pointer array. ** ** When this function is called, *ppData points to the start of the ** content area on page pPg. If the size of the content area is extended, ** *ppData is updated to point to the new start of the content area ** before returning. ** ** Finally, argument pBegin points to the byte immediately following the ** end of the space required by this page for the cell-pointer area (for ** all cells - not just those inserted by the current call). If the content ** area must be extended to before this point in order to accomodate all ** cells in apCell[], then the cells do not fit and non-zero is returned.
label: code-design
77934. '\$'
77935. First part of the name of the table or view
77936. collate ::=
77937. ** Allocate a new SegReader object.
77938. Input text
77939. Used to iterate through levels
77940. **comment:** ** "Special" FTS4 arguments are column specifications of the following form: ** ** <key> = <value> ** ** There may not be whitespace surrounding the "=" character. The <value> ** term may be quoted, but the <key> may not.
label: code-design
77941. !SQLITE_OMIT_BLOB_LITERAL || SQLITE_HAS_CODEC
77942. prereq always non-zero
77943. 4
77944. Register holding operands

77945. Set of characters to trim
77946. Last key written to the current block
77947. Information used by aggregate queries
77948. Compile a SELECT statement for this cursor. For a full-table-scan, the ** statement loops through all rows of the %_content table. For a ** full-text query or docid lookup, the statement retrieves a single ** row by docid.
77949. If this is a leaf page or the tree is not an int-key tree, then ** this page contains countable entries. Increment the entry counter ** accordingly.
77950. ** Check to see if this machine uses EBCDIC. (Yes, believe it or ** not, there are still machines out there that use EBCDIC.)
77951. A SELECT statement that supplies values
77952. Constraint value.
77953. Zero or more NC_* flags defined below
77954. ** Destroy an existing tokenizer. The fts3 module calls this method ** exactly once for each successful call to xCreate().
77955. All the rest should always be initialized to zero
77956. Verify that integers and floating point values use the same ** byte order. Or, that if SQLITE_MIXED_ENDIAN_64BIT_FLOAT is ** defined that 64-bit floating point values really are mixed ** endian.
77957. Source data
77958. xUnlock method
77959. Test to see if the iRowid value appears anywhere in the forest. ** Return 1 if it does and 0 if not.
77960. Address of "rows visited" counter
77961. ***** End of status.c *****
77962. ** Return TRUE (non-zero) of the statement supplied as an argument needs ** to be recompiled. A statement needs to be recompiled whenever the ** execution environment changes in a way that would alter the program ** that sqlite3_prepare() generates. For example, if new functions or ** collating sequences are registered or if an authorizer function is ** added or changed.
77963. Attempt to extend file to this size
77964. The MergeEngine that the IncrMerger will control
77965. Array of expression tokens and costs
77966. Current offset within position list
77967. P5 value for OP_Open* opcodes (except on WITHOUT ROWID)
77968. OS_VXWORKS
77969. Accumulate the output here
77970. IN/OUT: SQLITE_OPEN_XXX flags
77971. ** sqlite3_changegroup handle.
77972. **comment:** The where level to be coded
 label: code-design
77973. At this point, the (iCol+1) field prefix of aSample[i] is the first ** sample that is greater than pRec. Or, if i==pIdx->nSample then pRec ** is larger than all samples in the array.
77974. More ARRAY/OBJECT entries at u.iAppend
77975. rbu_control value ('x.x.')
77976. ** Call exprAnalyze on all terms in a WHERE clause. *** Note that exprAnalyze() might add new virtual terms onto the ** end of the WHERE clause. We do not want to analyze these new ** virtual terms, so start analyzing at the end and work forward ** so that the added virtual terms are never processed.
77977. **comment:** ** Stem the input word zIn[0..nIn-1]. Store the output in zOut. ** zOut is at least big enough to hold nIn bytes. Write the actual ** size of the output word (exclusive of the '\0' terminator) into *pnOut. *** Any upper-case characters in the US-ASCII character set ([A-Z]) ** are converted to lower case. Upper-case UTF characters are ** unchanged. *** Words that are longer than about 20 bytes are stemmed by retaining ** a few bytes from the beginning and the end of the word. If the ** word contains digits, 3 bytes are taken from the beginning and ** 3 bytes from the end. For long words without digits, 10 bytes ** are taken from each end. US-ASCII case folding still applies. *** If the input word contains not digits but does characters not ** in [a-zA-Z] then no stemming is attempted and this routine just ** copies the input into the output with US-ASCII ** case folding. *** Stemming never increases the length of the word. So there is ** no chance of overflowing the zOut buffer.
 label: code-design
77978. The following variables are all protected by the STATIC_MASTER ** mutex, not by sqlite3.mutex. They are used by code in notify.c. *** When X.pUnlockConnection==Y, that means that X is waiting for Y to ** unlock so that it can proceed. *** When X.pBlockingConnection==Y, that means that something that X tried ** tried to do recently failed with an SQLITE_LOCKED error due to locks ** held by Y.
77979. ** If the given Mem* has a zero-filled tail, turn it into an ordinary ** blob stored in dynamically allocated space.
77980. SQLITE_OMIT_LOCALTIME
77981. same as TK_ISNULL, jump, in1
77982. If we are trying to update a view, realize that view into ** an ephemeral table.
77983. Find the best snippet of NFToken tokens in column iRead.
77984. Cursor to create snippet for
77985. Find the named savepoint. If there is no such savepoint, then an ** an error is returned to the user.
77986. ** json_valid(JSON) *** Return 1 if JSON is a well-formed JSON string according to RFC-7159. ** Return 0 otherwise.
77987. IMPLEMENTATION-OF: R-26000-56589 The xShutdown() method may be NULL.
77988. Number of arguments to VUpdate
77989. 282
77990. User data for aux. function
77991. Datatype of the argument to the memory allocated passed as the ** second argument to sqlite3Fts5ParserAlloc() below. This can be changed by ** putting an appropriate #define in the %include section of the input ** grammar.
77992. ** Magic number used for the JSON parse cache in sqlite3_get_auxdata()
77993. tab2 may not be a view
77994. 80..87,
77995. (<tbl> MATCH ? ORDER BY rank)
77996. A regular INSERT, UPDATE or DELETE statement. The trick here is that ** any conflict on the rowid value must be detected before any ** modifications are made to the database file. There are 4 cases: *** 1) DELETE ** 2) UPDATE (rowid not modified) ** 3) UPDATE (rowid modified) ** 4) INSERT *** Cases 3 and 4 may violate the rowid constraint.
77997. Allocate and initialize the hash-table used to store tokenizers.
77998. True if current operation is a delete
77999. Parse the filename/URI argument *** Only allow sensible combinations of bits in the flags argument. ** Throw an error if any non-sense combination is used. If we ** do not block illegal combinations here, it could trigger ** assert() statements in deeper layers. Sensible combinations ** are: *** 1:
 SQLITE_OPEN_READONLY ** 2: SQLITE_OPEN_READWRITE ** 6: SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE
78000. Resolve the references in the WHERE clause.
78001. ** Decode the "end_block" field, selected by column iCol of the SELECT ** statement passed as the first argument. *** The "end_block" field may contain either an integer, or a text field ** containing the text representation of two non-negative integers separated ** by one or more space (0x20) characters. In the first case, set *piEndBlock ** to the integer value and *pnByte to zero before returning. In the second, ** set *piEndBlock to the first value and *pnByte to the second.
78002. Initialize a tokenizer iterator to iterate through column iCol.
78003. ** Discard all pages from cache pCache with a page number (key value) ** greater than or equal to iLimit. Any pinned pages that meet this ** criteria are unpinned before they are discarded. *** The PCache mutex must be held when this function is called.
78004. 13
78005. Open the table. Loop through all rows of the table, inserting index ** records into the sorter.
78006. Opcode: IdxLE P1 P2 P3 P4 P5 ** Synopsis: key=r[P3@P4] *** The P4 register values beginning with P3 form an unpacked index ** key that omits the PRIMARY KEY or ROWID. Compare this key value against ** the index that P1 is currently pointing to, ignoring the PRIMARY KEY or ** ROWID on the P1 index. *** If the P1 index entry is less than or equal to the key value then jump ** to P2. Otherwise fall through to the next instruction.

78007. Fd for main db of dbRbu
78008. This code runs once to initialize everything.
78009. ** Set the soft heap-size limit for the library. Passing a zero or ** negative value indicates no limit.
78010. Size of utf-16 input string in bytes
78011. ***** End of %include directives *****
78012. ** Add a new column to the table currently being constructed. ** ** The parser calls this routine once for each column declaration ** in a CREATE TABLE statement. sqlite3StartTable() gets called ** first to get things going. Then this routine is called for each ** column.
78013. ** Each cache entry is represented by an instance of the following ** structure. Unless SQLITE_PCACHE_SEPARATE_HEADER is defined, a buffer of ** PgHdr1.pCache->szPage bytes is allocated directly before this structure ** in memory.
78014. If this is an auto-vacuum database, update the pointer map ** with entries for the new page, and any pointer from the ** cell on the page to an overflow page. If either of these ** operations fails, the return code is set, but the contents ** of the parent page are still manipulated by thh code below. ** That is Ok, at this point the parent page is guaranteed to ** be marked as dirty. Returning an error code will cause a ** rollback, undoing any changes made to the parent page.
78015. ** This routine is called when a commit occurs.
78016. Decode the page just read from disk
78017. case PragTyp_BUSY_TIMEOUT
78018. If this is a DELETE or UPDATE operation, remove the old record.
78019. Seek opcode
78020. **comment:** TUNING: Currently sSum.a[i].rRun is set to the sum of the costs ** of all sub-scans required by the OR-scan. However, due to rounding ** errors, it may be that the cost of the OR-scan is equal to its ** most expensive sub-scan. Add the smallest possible penalty ** (equivalent to multiplying the cost by 1.07) to ensure that ** this does not happen. Otherwise, for WHERE clauses such as the ** following where there is an index on "y", ** ** WHERE likelihood(x=?, 0.99) OR y=? ** ** the planner may elect to "OR" together a full-table scan and an ** index lookup. And other similarly odd results.
label: code-design
78021. sqlite3_int64, sqlite3_int64
78022. If no zContent option was specified, fill in the default values.
78023. SQLITE_BLDF_* flags
78024. Two or more OR-connected terms
78025. **comment:** ** Return FALSE if there is no chance that the expression can be NULL. ** ** If the expression might be NULL or if the expression is too complex ** to tell return TRUE. ** ** This routine is used as an optimization, to skip OP_IsNull opcodes ** when we know that a value cannot be NULL. Hence, a false positive ** (returning TRUE when in fact the expression can never be NULL) might ** be a small performance hit but is otherwise harmless. On the other ** hand, a false negative (returning FALSE when the result could be NULL) ** will likely result in an incorrect answer. So when in doubt, return ** TRUE.
label: code-design
78026. SQLITE_OS_SETUP_H
78027. ** Default synchronous levels. ** ** Note that (for historical reasons) the PAGER_SYNCHRONOUS_* macros differ ** from the SQLITE_DEFAULT_SYNCHRONOUS value by 1. ** ** PAGER_SYNCHRONOUS DEFAULT_SYNCHRONOUS ** OFF 1 0 ** NORMAL 2 1 ** FULL 3 2 ** EXTRA 4 3 ** ** The "PRAGMA synchronous" statement also uses the zero-based numbers. ** In other words, the zero-based numbers are used for all external interfaces ** and the one-based values are used internally.
78028. NEAR parameter
78029. Store the SELECT statement in pRight so it will be deleted when ** sqlite3ExprListDelete() is called
78030. Table being queried
78031. Only change the value of sqlite.enc if the database handle is not ** initialized. If the main database exists, the new sqlite.enc value ** will be overwritten when the schema is next loaded. If it does not ** already exists, it will be created to use the new encoding value.
78032. The JSON that is being patched
78033. SQLITE_OMIT_DEPRECATED
78034. Database filename (UTF-16)
78035. (322) kwcolumn_opt ::= COLUMNKW
78036. The following condition causes URIs with five leading / characters ** like file:///host/path to be converted into UNC's like //host/path. ** The correct URI for that UNC has only two or four leading / characters ** file://host/path or file:///host/path. But 5 leading slashes is a ** common error, we are told, so we handle it as a special case.
78037. Expected pointer map parent page number
78038. Do not allow any cells smaller than 4 bytes. If a smaller cell ** does exist, pad it with 0x00 bytes.
78039. (rowid = ?)
78040. ** Check to see if there are references to columns in table ** pWalker->u.pIdxCover->iCur can be satisfied using the index ** pWalker->u.pIdxCover->pIdx.
78041. New virtual table object
78042. ** An instance of the following structure represents a single search term ** or term prefix.
78043. create_table_args ::= AS select
78044. Chunk size configured by FCNTL_CHUNK_SIZE
78045. **comment:** ** This function is called to drop a trigger from the database schema. ** ** This may be called directly from the parser and therefore identifies ** the trigger by name. The sqlite3DropTriggerPtr() routine does the ** same job as this routine except it takes a pointer to the trigger ** instead of the trigger name. *
label: code-design
78046. ** Allowed values for Table.tabFlags. ** ** TF_OOHidden applies to tables or view that have hidden columns that are ** followed by non-hidden columns. Example: "CREATE VIRTUAL TABLE x USING ** vtab1(a HIDDEN, b)". Since "b" is a non-hidden column but "a" is hidden, ** the TF_OOHidden attribute would apply in this case. Such tables require ** special handling during INSERT processing.
78047. The AND operator in x>=y AND x<=z
78048. Index on key columns in pTo
78049. ** End of interface to code in fts5_config.c. *****
78050. Number of prefix indexes
78051. Changes list if ON UPDATE CASCADE
78052. **comment:** The following blocks should probably assert in debug mode, but they are to cleanup in case any locks remained open
label: code-design
78053. Total number of pages in database file
78054. case_else ::= ELSE expr
78055. Mark a single page as clean
78056. Zero or more SQLITE_PREPARE_flags
78057. Error message from sqlite3_mprintf()
78058. Sum of ref counts over all pages
78059. Open a read-only cursor on the index being analyzed.
78060. first rowid on page
78061. Number of matchable phrases in query
78062. mutex
78063. ** This function is called when the user invokes "PRAGMA wal_checkpoint", ** "PRAGMA wal_blocking_checkpoint" or calls the sqlite3_wal_checkpoint() ** or wal_blocking_checkpoint() API functions. ** ** Parameter eMode is one of SQLITE_CHECKPOINT_PASSIVE, FULL or RESTART.
78064. Search for equality and range constraints on the "term" column. ** And equality constraints on the hidden "languageid" column.
78065. Number of bytes to return.
78066. Page size must be a power of 2
78067. Fallback loop must terminate
78068. Memory management strategy for zName
78069. ** This routine is the only routine in this file with external linkage. ** ** Populate the low-level memory allocation function pointers in ** sqlite3GlobalConfig.m with pointers to the routines in this file.

78070. Index into Parse.aAlias[] for zName
78071. Fall through into OP_Function
78072. ** The first argument, pCur, is a cursor opened on some b-tree. Count the ** number of entries in the b-tree and write the result to *pnEntry. ** ** SQLITE_OK is returned if the operation is successfully executed. ** Otherwise, if an error is encountered (i.e. an IO error or database ** corruption) an SQLite error code is returned.
78073. Rtree table
78074. If this is an UPDATE OF <column-list> trigger, the <column-list> is stored here
78075. ** This routine generates the code needed to write autoincrement ** maximum rowid values back into the sqlite_sequence register. ** Every statement that might do an INSERT into an autoincrement ** table (either directly or through triggers) needs to call this ** routine just before the "exit" code.
78076. *
78077. Non-zero if locking is disabled.
78078. Write the page data
78079. func or agg-step
78080. Offset of start of buffer in file
78081. **comment:** TODO: Do we need this if the leaf-index is appended? Probably...
 label: code-design
78082. synopsis: Start at P2
78083. ** Generate code to drop and reload the internal representation of table ** pTab from the database, including triggers and temporary triggers. ** Argument zName is the name of the table in the database schema at ** the time the generated code is executed. This can be different from ** pTab->zName if this function is being called to code part of an ** "ALTER TABLE RENAME TO" statement.
78084. The vector inequality constraint
78085. 182
78086. Do not allow backup if the destination database is in WAL mode ** and the page sizes are different between source and destination
78087. ** Free all memory associated with foreign key definitions attached to ** table pTab. Remove the deleted foreign keys from the Schema.fkeyHash ** hash table.
78088. If pState is NULL, then the wal file may not have been opened and ** recovered. Running a read-statement here to ensure that doing so ** does not interfere with the "capture" process below.
78089. Cursor snippet is being generated from
78090. OOM
78091. True to enable memory status
78092. Value extracted from record
78093. Try to delete the WAL file if the checkpoint completed and ** fsyned (rc==SQLITE_OK) and if we are not in persistent-wal ** mode (!bPersist)
78094. If the user specified a negative value for the languageid, use zero ** instead. This works, as the "languageid=?" constraint will also ** be tested by the VDBE layer. The test will always be false (since ** this module will not return a row with a negative languageid), and ** so the overall query will return zero rows.
78095. Parent table that child points to
78096. Subroutine return code
78097. Pointer to a single WhereLoop object
78098. ** Append a single varint to a PendingList buffer. SQLITE_OK is returned ** if successful, or an SQLite error code otherwise. ** ** This function also serves to allocate the PendingList structure itself. ** For example, to create a new PendingList structure containing two ** varints: ** ** PendingList *p = 0; ** fts3PendingListAppendVarint(&p, 1); ** fts3PendingListAppendVarint(&p, 2);
78099. Skip any MATCH_ALL or MATCH_ONE characters that follow a ** MATCH_ALL. For each MATCH_ONE, skip one character in the ** test string.
78100. ** Initialize the mutex system.
78101. ** If X and Y are NULL (in other words if only the column name Z is ** supplied) and the value of Z is enclosed in double-quotes, then ** Z is a string literal if it doesn't match any column names. In that ** case, we need to return right away and not make any changes to ** pExpr. ** ** Because no reference was made to outer contexts, the pNC->nRef ** fields are not changed in any context.
78102. Cursor object to populate
78103. ** The following routines implement the various date and time functions ** of SQLite.
78104. 0=Monday, 1=Tuesday, ... 6=Sunday
78105. Type of the container - used for error messages
78106. getenv("SQLITE_TMPDIR")
78107. ** Perform a shift action.
78108. Works for compilers other than LLVM
78109. It is not possible to do a full assert_pager_state() here, as this ** function may be called from within PagerOpen(), before the state ** of the Pager object is internally consistent. ** ** At one point this function returned an error if the pager was in ** PAGER_ERROR state. But since PAGER_ERROR state guarantees that ** there is at least one outstanding page reference, this function ** is a no-op for that case anyhow.
78110. Find the next free blockid in the %_segments table
78111. True for an update, false otherwise
78112. ** Find and return the schema associated with a BTree. Create ** a new one if necessary.
78113. Second arg to matchinfo() function
78114. IMPLEMENTATION-OF: R-49045-42493 SQLite will use the xCurrentTimeInt64() ** method to get the current date and time if that method is available ** (if iVersion is 2 or greater and the function pointer is not NULL) and ** will fall back to xCurrentTime() if xCurrentTimeInt64() is ** unavailable.
78115. Expr list from which to derive column names
78116. Cursor to receive the page, or NULL
78117. Table cursor common to all terms
78118. List of binary trees of entries
78119. Index of PmaReader to advance
78120. OUT: Length of *ppOffsetList in bytes
78121. **comment:** Table type - an RBU_PK_XXX value
 label: code-design
78122. ** Default maximum size of memory used by memory-mapped I/O in the VFS
78123. ** Return the best representation of pMem that we can get into a ** double. If pMem is already a double or an integer, return its ** value. If it is a string or blob, try to convert it to a double. ** If it is a NULL, return 0.0.
78124. **comment:** ** This is the obsolete pcache_methods object that has now been replaced ** by sqlite3_pcache_methods2. This object is not used by SQLite. It is ** retained in the header file for backwards compatibility only.
 label: code-design
78125. 1390
78126. **comment:** ** The first argument must be a nul-terminated string. This function ** returns a copy of the string in memory obtained from sqlite3_malloc(). ** It is the responsibility of the caller to eventually free this memory ** using sqlite3_free(). ** ** If an OOM condition is encountered when attempting to allocate memory, ** output variable (*pRc) is set to SQLITE_NOMEM before returning. Otherwise, ** if the allocation succeeds, (*pRc) is left unchanged.
 label: code-design
78127. SQLITE OMIT AUTHORIZATION
78128. generate the SELECT expression tree.
78129. Index of the database holding pTab
78130. Previous element in list of dirty pages
78131. OUT: Pointer to buffer containing data
78132. distinct ::= DISTINCT
78133. Address of OP_OpenEphemeral instruction
78134. root

78135. Decode the arguments passed through to this function. *** Note: The following set of if(...) statements must be in the same ** order as the corresponding entries in the struct at the top of ** fts5BestIndexMethod().

78136. Virtual table configuration

78137. ** Given a page number of a regular database page, return the page ** number for the pointer-map page that contains the entry for the ** input page number. *** Return 0 (not a valid page) for pgno==1 since there is ** no pointer map associated with page 1. The integrity_check logic ** requires that ptrmapPageNo(*,1)!=1.

78138. EVIDENCE-OF: R-10421-19736 If any other process is running a ** checkpoint operation at the same time, the lock cannot be obtained and ** SQLITE_BUSY is returned. ** EVIDENCE-OF: R-53820-33897 Even if there is a busy-handler configured, ** it will not be invoked in this case.

78139. ** The maximum (and only) versions of the wal and wal-index formats ** that may be interpreted by this version of SQLite. *** If a client begins recovering a WAL file and finds that (a) the checksum ** values in the wal-header are correct and (b) the version field is not ** WAL_MAX_VERSION, recovery fails and SQLite returns SQLITE_CANTOPEN. *** Similarly, if a client successfully reads a wal-index header (i.e. the ** checksum test is successful) and finds that the version field is not ** WALINDEX_MAX_VERSION, then no read-transaction is opened and SQLite ** returns SQLITE_CANTOPEN.

78140. Parent savepoint (if any)

78141. assert(seenReplace==0);

78142. Used by: index_xinfo

78143. ** Based on the primary key values stored in change aRecord, calculate a ** hash key. Assume the has table has nBucket buckets. The hash keys ** calculated by this function are compatible with those calculated by ** sessionPreupdateHash(). *** The bPkOnly argument is non-zero if the record at aRecord[] is from ** a patchset DELETE. In this case the non-PK fields are omitted entirely.

78144. Zero pad

78145. ** Change the error string stored in Vdbe.zErrMsg

78146. synopsis: r[P3]=PX

78147. EVIDENCE-OF: R-44885-25196 Value is an 8-bit twos-complement ** integer.

78148. ** Convert the JsonNode pNode into a pure JSON string and ** append to pOut. Subsubstructure is also included. Return ** the number of JsonNode objects that are encoded.

78149. New Pager.aSavepoint array

78150. Verify that all AggInfo registers are within the range specified by ** AggInfo.mnReg..AggInfo.mxReg

78151. ***** This division contains the implementation of methods on the ** sqlite3_vfs object.

78152. Name of column in parent table

78153. True to ignore xSavepoint invocations

78154. ** The default initial allocation for the pagecache when using separate ** pagecaches for each database connection. A positive number is the ** number of pages. A negative number N translations means that a buffer ** of -1024*N bytes is allocated and used for as many pages as it will hold. *** The default value of "20" was chosen to minimize the run-time of the ** speedtest1 test program with options: --shrink-memory --reprepare

78155. Seek the table cursor, if required

78156. ** Return SQLITE_CORRUPT_VTAB.

78157. ***** End of backup.c *****

78158. If p holds a string or blob, the Mem.z must point to exactly ** one of the following: *** (1) Memory in Mem.zMalloc and managed by the Mem object ** (2) Memory to be freed using Mem.xDel ** (3) An ephemeral string or blob ** (4) A static string or blob

78159. SYNC_NORMAL or SYNC_FULL for wal writes

78160. Datatype of the argument to the memory allocated passed as the ** second argument to sqlite3ParserAlloc() below. This can be changed by ** putting an appropriate #define in the %include section of the input ** grammar.

78161. SQL for the statement

78162. #ifndef SQLITE_OMIT_FOREIGN_KEY

78163. If not NULL, info on how to process ORDER BY

78164. Output the final row of result

78165. Sizes of deleted document written here

78166. ** Close the blob handle at p->pSegments, if it is open. See comments above ** the sqlite3Fts3ReadBlock() function for details.

78167. ** CAPI3REF: Virtual Table Object ** KEYWORDS: sqlite3_module {virtual table module} *** This structure, sometimes called a "virtual table module", ** defines the implementation of a [virtual tables]. ** This structure consists mostly of methods for the module. *** ^A virtual table module is created by filling in a persistent ** instance of this structure and passing a pointer to that instance ** to [sqlite3_create_module()] or [sqlite3_create_module_v2()]. ** ^The registration remains valid until it is replaced by a different ** module or until the [database connection] closes. The content ** of this structure must not change while it is registered with ** any database connection.

78168. 6-byte signed integer

78169. Access permission denied

78170. Check this handle's interrupt flag

78171. if the conch isn't writable and doesn't match, we can't take it

78172. READWRITE | CREATE

78173. **comment:** Unused
label: code-design

78174. Index (0..Fts3Table.nIndex-1)

78175. ** Obtain the STATIC_MASTER mutex.

78176. OUT: Set to true if constraint may cause a replace

78177. **comment:** ** Return a pointer to a buffer containing a text representation of the ** expression passed as the first argument. The buffer is obtained from ** sqlite3_malloc(). It is the responsibility of the caller to use ** sqlite3_free() to release the memory. If an OOM condition is encountered, ** NULL is returned. ** If the second argument is not NULL, then its contents are prepended to ** the returned expression text and then freed using sqlite3_free().
label: code-design

78178. Average number of tokens in each row

78179. Global context for this function

78180. Activation phrase

78181. Evaluate the current GROUP BY terms and store in b0, b1, b2... ** (b0 is memory location iBMem+0, b1 is iBMem+1, and so forth) ** Then compare the current GROUP BY terms against the GROUP BY terms ** from the previous row currently stored in a0, a1, a2...

78182. ** Do up to nPg pages of automerge work on the index. *** Return true if any changes were actually made, or false otherwise.

78183. ** The remainder of this file contains the declarations of the functions ** that make up the Pager sub-system API. See source code comments for ** a detailed description of each routine.

78184. B-Tree Database

78185. TUNING: Cost of full table scan is (N*3.0).

78186. The ',' token after the last column defn.

78187. All conditions meet

78188. Acceptable operators

78189. ** Structure used to store the context required by the ** sqlite3_prewrite_*() API functions.

78190. The major token value. This is the code ** number for the token at this stack level

78191. Return the results

78192. Argument to xRollbackCallback()

78193. Sorted index for this segment

78194. ** This function is called to obtain a pointer to region iRegion of the ** shared-memory associated with the database file fd. Shared-memory regions ** are numbered starting from zero. Each shared-memory region is szRegion ** bytes in size. *** If an error occurs, an error code is returned and *pp is set to NULL. ** Otherwise, if the bExtend parameter is 0 and the requested shared-memory ** region has not been allocated (by any client, including one running in a ** separate process), then *pp is set to NULL and SQLITE_OK returned. If ** bExtend is non-zero and the requested shared-memory region has not yet ** been

allocated, it is allocated by this function. *** If the shared-memory region has already been allocated or is allocated by ** this call as described above, then it is mapped into this processes ** address space (if it is not already), *pp is set to point to the mapped ** memory and SQLITE_OK returned.

78195. WAL handle

78196. ** The next global variable records the size of the largest MEM_Blob ** or MEM_Str that has been used by a VDBE opcode. The test procedures ** use this information to make sure that the zero-blob functionality ** is working correctly. This variable has no function other than to ** help verify the correct operation of the library.

78197. Unless this is a view, open cursors for the table we are ** deleting from and all its indices. If this is a view, then the ** only effect this statement has is to fire the INSTEAD OF ** triggers.

78198. Figure out the effective temporary directory. First, check if one ** has been explicitly set by the application; otherwise, use the one ** configured by the operating system.

78199. One of TK_UNION, TK_EXCEPT etc.

78200. PmaReader to populate

78201. DELETE

78202. Must be compatible with this index, if not NULL

78203. Number of slots in aOut[]

78204. P4 is a pointer to a Table structure

78205. Docs in descending rowid order

78206. ifndef SQLITE OMIT WAL

78207. fts3_snippet.c

78208. **comment:** * If this is non-zero, an isolated heap will be created by the native Win32 * allocator subsystem; otherwise, the default process heap will be used. This * setting has no effect when compiling for WinRT. By default, this is enabled * and an isolated heap will be created to store all allocated data. *

***** * WARNING: It is important to note that when this setting is non-zero and the * winMemShutdown function is called (e.g. by the sqlite3_shutdown * function), all data that was allocated using the isolated heap will * be freed immediately and any attempt to access any of that freed * data will almost certainly result in an immediate access violation.

label: code-design

78209. ** Make yyytestCase() the same as testcase()

78210. True if attempting to insert into a view

78211. 130

78212. Open file descriptor on path

78213. Mask of function arguments that are constant

78214. Number of columns in result set

78215. SimulateIOError(rc=0; cnt=MX_CLOSE_ATTEMPT;);

78216. Unless the pager is in noSync mode, the journal file was just ** successfully synced. Either way, clear the PGHDR_NEED_SYNC flag on ** all pages.

78217. Set *pCurrent to non-zero if there are unresolved deferred foreign ** key constraints. Set *pCurrent to zero if all foreign key constraints ** have been satisfied. The *pHighwater is always set to zero.

78218. Verify that the database file has not been deleted or renamed out from ** under the pager. Return SQLITE_OK if the database is still where it ought ** to be on disk. Return non-zero (SQLITE_READONLY_DBMOVED or some other error ** code from sqlite3OsAccess()) if the database has gone missing.

78219. ** Free the list of FileChunk structures headed at MemJournal.pFirst.

78220. ** xSetOutputs callback used by detail=full when there is a column filter.

78221. 0x70..0x7F

78222. This is the common case where the desired content fits on the original ** page - where the content is not on an overflow page

78223. **comment:** Read and write the xxx_parent table

label: code-design

78224. ** This function counts the total number of docids in the doclist stored ** in buffer aList[], size nList bytes. *** If the isPoslist argument is true, then it is assumed that the doclist ** contains a position-list following each docid. Otherwise, it is assumed ** that the doclist is simply a list of docids stored as delta encoded ** varints.

78225. Insert the end of a co-routine

78226. #include "fts3_hash.h"

78227. ** Session handle structure.

78228. ** Return the total number of references to all pages held by the cache. *** This is not the total number of pages referenced, but the sum of the ** reference count for all pages.

78229. synopsis: r[P1]=max(r[P1],r[P2])

78230. ** A container for a temp file handle and the current amount of data ** stored in the file.

78231. Callback pointers

78232. ** Valid values for the second argument to sqlite3PagerLockingMode().

78233. Next in a list of them all

78234. Convert to ascii

78235. The new leaf holds no terms or rowids

78236. 0,1

78237. 2: Seek the cursor to rowid=r[1]

78238. 91

78239. Estimated number of output rows

78240. ** Return N random bytes.

78241. ** This function changes all write-locks held by Btree p into read-locks.

78242. List of attached tables

78243. Current docid (if pList!=0)

78244. Size of array apWiData

78245. If pSynced is NULL and this page has a clear NEED_SYNC flag, set ** pSynced to point to it. Checking the NEED_SYNC flag is an ** optimization, as if pSynced points to a page with the NEED_SYNC ** flag set sqlite3PcacheFetchStress() searches through all newer ** entries of the dirty-list for a page with NEED_SYNC clear anyway.

78246. NOTE 1

78247. Top of the loop body

78248. Address of the select-A-exhausted subroutine

78249. The FROM clause of the subquery

78250. **comment:** ** 2008 February 16 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** * This file implements an object that represents a fixed-length ** bitmap. Bits are numbered starting with 1. *** A bitmap is used to record which pages of a database file have been ** journaled during a transaction, or which pages have the "dont-write" ** property. Usually only a few pages are meet either condition. ** So the bitmap is usually sparse and has low cardinality. ** But sometimes (for example when during a DROP of a large table) most ** or all of the pages in a database can get journaled. In those cases, ** the bitmap becomes dense with high cardinality. The algorithm needs ** to handle both cases well. *** The size of the bitmap is fixed when the object is created. *** All bits are clear when the bitmap is created. Individual bits ** may be set or cleared one at a time. *** Test operations are about 100 times more common than set operations. ** Clear operations are exceedingly rare. There are usually between ** 5 and 500 set operations per Bitvec object, though the number of sets can ** sometimes grow into tens of thousands or larger. The size of the ** Bitvec object is the number of pages in the database file at the ** start of a transaction, and is thus usually less than a few thousand, ** but can be as large as 2 billion for a really big database.

label: code-design

78251. ** Similar to sqlite3Fts3GetVarint(), except that the output is truncated to ** a non-negative 32-bit integer before it is returned.

78252. **comment:** ** 2004 May 22 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the VFS implementation for unix-like operating systems ** include Linux, MacOSX, *BSD, QNX, VxWorks, AIX, HPUX, and others. ** ** There are actually several different VFS implementations in this file. ** The differences are in the way that file locking is done. The default ** implementation uses Posix Advisory Locks. Alternative implementations ** use flock(), dot-files, various proprietary locking schemas, or simply ** skip locking all together. ** ** This source file is organized into divisions where the logic for various ** subfunctions is contained within the appropriate division. PLEASE ** KEEP THE STRUCTURE OF THIS FILE INTACT. New code should be placed ** in the correct division and should be clearly labeled. ** ** The layout of divisions is as follows: ** ** * General-purpose declarations and utility functions. ** * Unique file ID logic used by VxWorks. ** * Various locking primitive implementations (all except proxy locking): ** + for Posix Advisory Locks ** + for no-op locks ** + for dot-file locks ** + for flock() locking ** + for named semaphore locks (VxWorks only) ** + for AFP filesystem locks (MacOSX only) ** * sqlite3_file methods not associated with locking. ** * Definitions of sqlite3_io_methods objects for all locking ** methods plus "finder" functions for each locking method. ** * sqlite3_vfs method implementations. ** * Locking primitives for the proxy uber-locking-method. (MacOSX only) ** * Definitions of sqlite3_vfs objects for all locking methods ** plus implementations of sqlite3_os_init() and sqlite3_os_end().

label: code-design

78253. Number of bytes in buffer z

78254. Number of bytes in zTerm

78255. ** This constant should already be defined (in the "WinDef.h" SDK file).

78256. **comment:** List of unused pcache-local pages

label: code-design

78257. nCol values

78258. ***** Continuing where we left off in os_win.c *****

78259. EVIDENCE-OF: R-39100-27317 The SQLITE_CONFIG_PCACHE_HDRSZ option takes ** a single parameter which is a pointer to an integer and writes into ** that integer the number of extra bytes per page required for each page ** in SQLITE_CONFIG_PAGECACHE.

78260. Snippet to extract

78261. Name of SQL function

78262. If non-NULL, store size of record here

78263. Write this value to the list

78264. ** Return an estimate for the number of rows in the table that pCur is ** pointing to. Return a negative number if no estimate is currently ** available.

78265. EQP id of left-hand query

78266. ** Prepare a virtual machine for execution for the first time after ** creating the virtual machine. This involves things such ** as allocating registers and initializing the program counter. ** After the Vdbe has been prepped, it can be executed by one or more ** calls to sqlite3VdbeExec(). ** ** This function may be called exactly once on each virtual machine. ** After this routine is called the VM has been "packaged" and is ready ** to run. After this routine is called, further calls to ** sqlite3VdbeAddOp() functions are prohibited. This routine disconnects ** the Vdbe from the Parse object that helped generate it so that the ** Vdbe becomes an independent entity and the Parse object can be ** destroyed. ** ** Use the sqlite3VdbeRewind() procedure to restore a virtual machine back ** to its initial state after it has been run.

78267. Low-level file information

78268. Total segments in this structure

78269. **comment:** ** This macro is used when a local variable is set to a value that is ** [sometimes] not used by the code (e.g. via conditional compilation).

label: code-design

78270. Store SQLITE_NOMEM here if required

78271. Turns out that wasn't a keyword after all. This happens if the ** user has supplied a token such as "ORacle". Continue.

78272. Index into aSub array

78273. Node or entry ID

78274. **comment:** Avoid leaking memory if malloc has failed.

label: code-design

78275. Either "tbl" or "idx"

78276. Add a field to the new.* record. Or the only record if currently ** generating a patchset.

78277. Copy of pFKey->pNextFrom

78278. 180

78279. xSync

78280. The WhereLoop builder

78281. We are told that some versions of Android contain a bug that ** sizes ino_t at only 32-bits instead of 64-bits. (See ** <https://android-review.googlesource.com/#/c/115351/3/dist/sqlite3.c>) ** To work around this, always allocate 64-bits for the inode number. ** On small machines that only have 32-bit inodes, this wastes 4 bytes, ** but that should not be a big deal.

78282. **comment:** SQLITE_WIN32_MALLOC

label: code-design

78283. ** This function does the work for an sqlite3rbu_step() call. ** ** The object-iterator (p->objiter) currently points to a valid object, ** and the input cursor (p->objiter.pSelect) currently points to a valid ** input row. Perform whatever processing is required and return. ** ** If no error occurs, SQLITE_OK is returned. Otherwise, an error code ** and message is left in the RBU handle and a copy of the error code ** returned.

78284. Extract the PRIMARY KEY from the end of the index entry and ** store it in registers regR..regR+nPk-1

78285. getenv("TMPDIR")

78286. Link the foreign key to the table as the last step.

78287. True if output is already ordered

78288. **comment:** If the LHS and RHS of the IN operator do not match, that ** error will have been caught long before we reach this point.

label: code-design

78289. ** Copy the values stored in the VdbeFrame structure to its Vdbe. This ** is used, for example, when a trigger sub-program is halted to restore ** control to the main program.

78290. Number of entries in aColCache[]

78291. In case the cursor has been used before, clear it now.

78292. ** Variables in which to record status information.

78293. True if the DISTINCT keyword is present

78294. ** Buffers p1 and p2 contain doclists. This function merges the content ** of the two doclists together and sets buffer p1 to the result before ** returning. ** ** If an error occurs, an error code is left in p->rc. If an error has ** already occurred, this function is a no-op.

78295. Register holding assembled record for the table

78296. On commit, recalculate checksums from here

78297. Parent of this frame, or NULL if parent is main

78298. Cursor to iterate through zInput

78299. Token to add or remove to or from index

78300. TEMP

78301. **comment:** ** This function allocates a new parser. ** The only argument is a pointer to a function which works like ** malloc. ** ** Inputs: ** A pointer to the function used to allocate memory. ** ** Outputs: ** A pointer to a parser. This pointer is used in subsequent calls ** to sqlite3Parser and sqlite3ParserFree.

label: code-design

78302. ** Malloc function used by SQLite to obtain space from the buffer configured ** using sqlite3_config(SQLITE_CONFIG_PAGECACHE) option. If no such buffer ** exists, this function falls back to sqlite3Malloc().

78303. ** Destroy an existing tokenizer cursor. The fts3 module calls this ** method exactly once for each successful call to xOpen().

78304. ** This function is called after the cursor passed as the only argument ** is moved to point at a different row. It clears all cached data ** specific to the previous row stored by the cursor object.

78305. ** This routine does a complete check of the given BTree file. aRoot[] is ** an array of pages numbers were each page number is the root page of ** a table. nRoot is the number of entries in aRoot. *** A read-only or read-write transaction must be opened before calling ** this function. *** Write the number of error seen in *pnErr. Except for some memory ** allocation errors, an error message held in memory obtained from ** malloc is returned if *pnErr is non-zero. If *pnErr==0 then NULL is ** returned. If a memory allocation error occurs, NULL is returned.

78306. Index of column to create snippet from

78307. fullname := nm dbnm

78308. ** Each foreign key constraint is an instance of the following structure. *** A foreign key is associated with two tables. The "from" table is ** the table that contains the REFERENCES clause that creates the foreign ** key. The "to" table is the table that is named in the REFERENCES clause. ** Consider this example: *** CREATE TABLE ex1(** a INTEGER PRIMARY KEY, ** b INTEGER CONSTRAINT fk1 REFERENCES ex2(x)); *** For foreign key "fk1", the from-table is "ex1" and the to-table is "ex2". ** Equivalent names: *** from-table == child-table ** to-table == parent-table *** Each REFERENCES clause generates an instance of the following structure ** which is attached to the from-table. The to-table need not exist when ** the from-table is created. The existence of the to-table is not checked. *** The list of all parents for child Table X is held at X.pfKey. *** A list of all children for a table named Z (which might not even exist) ** is held in Schema.fkeyHash with a hash key of Z.

78309. **comment:** ** An open page cache is an instance of struct Pager. A description of ** some of the more important member variables follows: *** eState *** The current 'state' of the pager object. See the comment and state ** diagram above for a description of the pager state. *** eLock *** For a real on-disk database, the current lock held on the database file - ** NO_LOCK, SHARED_LOCK, RESERVED_LOCK or EXCLUSIVE_LOCK. *** For a temporary or in-memory database (neither of which require any ** locks), this variable is always set to EXCLUSIVE_LOCK. Since such ** databases always have Pager.exclusiveMode==1, this tricks the pager ** logic into thinking that it already has all the locks it will ever ** need (and no reason to release them). *** In some (obscure) circumstances, this variable may also be set to ** UNKNOWN_LOCK. See the comment above the #define of UNKNOWN_LOCK for ** details. *** changeCountDone *** This boolean variable is used to make sure that the change-counter ** (the 4-byte header field at byte offset 24 of the database file) is ** not updated more often than necessary. *** It is set to true when the change-counter field is updated, which ** can only happen if an exclusive lock is held on the database file. *** It is cleared (set to false) whenever an exclusive lock is ** relinquished on the database file. Each time a transaction is committed, ** The changeCountDone flag is inspected. If it is true, the work of ** updating the change-counter is omitted for the current transaction. *** This mechanism means that when running in exclusive mode, a connection ** need only update the change-counter once, for the first transaction ** committed. *** setMaster *** When PagerCommitPhaseOne() is called to commit a transaction, it may ** (or may not) specify a master-journal name to be written into the ** journal file before it is synced to disk. *** Whether or not a journal file contains a master-journal pointer affects ** the way in which the journal file is finalized after the transaction is ** committed or rolled back when running in "journal_mode=PERSIST" mode. ** If a journal file does not contain a master-journal pointer, it is ** finalized by overwriting the first journal header with zeroes. If ** it does contain a master-journal pointer the journal file is finalized ** by truncating it to zero bytes, just as if the connection were ** running in "journal_mode=truncate" mode. *** Journal files that contain master journal pointers cannot be finalized ** simply by overwriting the first journal-header with zeroes, as the ** master journal pointer could interfere with hot-journal rollback of any ** subsequently interrupted transaction that reuses the journal file. *** The flag is cleared as soon as the journal file is finalized (either ** by PagerCommitPhaseTwo or PagerRollback). If an IO error prevents the ** journal file from being successfully finalized, the setMaster flag ** is cleared anyway (and the pager will move to ERROR state). *** doNotSpill *** This variables control the behavior of cache-spills (calls made by ** the pcache module to the pagerStress() routine to write cached data ** to the file-system in order to free up memory). *** When bits SPILLFLAG_OFF or SPILLFLAG_ROLLBACK of doNotSpill are set, ** writing to the database from pagerStress() is disabled altogether. ** The SPILLFLAG_ROLLBACK case is done in a very obscure case that ** comes up during savepoint rollback that requires the pcache module ** to allocate a new page to prevent the journal file from being written ** while it is being traversed by code in pager_playback(). The SPILLFLAG_OFF ** case is a user preference. *** If the SPILLFLAG_NOSYNC bit is set, writing to the database from ** pagerStress() is permitted, but syncing the journal file is not. ** This flag is set by sqlite3PagerWrite() when the file-system sector-size ** is larger than the database page-size in order to prevent a journal sync ** from happening in between the journaling of two pages on the same sector. *** subJInMemory *** This is a boolean variable. If true, then any required sub-journal ** is opened as an in-memory journal file. If false, then in-memory ** sub-journals are only used for in-memory pager files. *** This variable is updated by the upper layer each time a new ** write-transaction is opened. *** dbSize, dbOrigSize, dbFileSize *** Variable dbSize is set to the number of pages in the database file. ** It is valid in PAGER_READER and higher states (all states except for ** OPEN and ERROR). *** dbSize is set based on the size of the database file, which may be ** larger than the size of the database (the value stored at offset ** 28 of the database header by the btree). If the size of the file ** is not an integer multiple of the page-size, the value stored in ** dbSize is rounded down (i.e. a 5KB file with 2K page-size has dbSize==2). ** Except, any file that is greater than 0 bytes in size is considered ** to have at least one page. (i.e. a 1KB file with 2K page-size leads ** to dbSize==1). *** During a write-transaction, if pages with page-numbers greater than ** dbSize are modified in the cache, dbSize is updated accordingly. ** Similarly, if the database is truncated using PagerTruncateImage(), ** dbSize is updated. *** Variables dbOrigSize and dbFileSize are valid in states ** PAGER_WRITER_LOCKED and higher. dbOrigSize is a copy of the dbSize ** variable at the start of the transaction. It is used during rollback, ** and to determine whether or not pages need to be journalled before ** being modified. *** Throughout a write-transaction, dbFileSize contains the size of ** the file on disk in pages. It is set to a copy of dbSize when the ** write-transaction is first opened, and updated when VFS calls are made ** to write or truncate the database file on disk. *** The only reason the dbFileSize variable is required is to suppress ** unnecessary calls to xTruncate() after committing a transaction. If, ** when a transaction is committed, the dbFileSize variable indicates ** that the database file is larger than the database image (Pager.dbSize), ** pager_truncate() is called. The pager_truncate() call uses xFilesize() ** to measure the database file on disk, and then truncates it if required. ** dbFileSize is not used when rolling back a transaction. In this case ** pager_truncate() is called unconditionally (which means there may be ** a call to xFilesize() that is not strictly required). In either case, ** pager_truncate() may cause the file to become smaller or larger. *** dbHintSize *** The dbHintSize variable is used to limit the number of calls made to ** the VFS xFileControl(FCNTL_SIZE_HINT) method. *** dbHintSize is set to a copy of the dbSize variable when a ** write-transaction is opened (at the same time as dbFileSize and ** dbOrigSize). If the xFileControl(FCNTL_SIZE_HINT) method is called, ** dbHintSize is increased to the number of pages that correspond to the ** size-hint passed to the method call. See pager_write_pagelist() for ** details. *** errCode *** The Pager.errCode variable is only ever used in PAGER_ERROR state. It ** is set to zero in all other states. In PAGER_ERROR state, Pager.errCode ** is always set to SQLITE_FULL, SQLITE_IOERR or one of the SQLITE_IOERR_XXX ** sub-codes.

label: code-design

78310. Fall through

78311. If the handle has a write-transaction open, commit the shared-btrees ** transaction and set the shared state to TRANS_READ.

78312. Write the file descriptor here

78313. Context

78314. PagerRollback() is a no-op if called in READER or OPEN state. If ** the pager is already in the ERROR state, the rollback is not ** attempted here. Instead, the error code is returned to the caller.

78315. Index of "the page" in pParent

78316. ** Similar to fts3SqlStmt(). Except, after binding the parameters in ** array apVal[] to the SQL statement identified by eStmt, the statement ** is executed. *** Returns SQLITE_OK if the statement is successfully executed, or an ** SQLite error code otherwise.

78317. Number of bytes in the BLOB

78318. ** Get a VDBE for the given parser context. Create a new one if necessary. ** If an error occurs, return NULL and leave a message in pParse.

78319. The returned text includes up to four fragments of text extracted from ** the data in the current row. The first iteration of the for(...) loop ** below attempts to locate a single fragment of text nToken tokens in ** size that contains at least one instance of all phrases in the query ** expression that appear in the current row. If such a fragment of text ** cannot be found, the second iteration of the loop attempts to locate ** a pair of fragments, and so on.

78320. Name of this index

78321. ***** End of prepare.c *****

78322. Space for Index.aAvgEq[]

78323. The left operand

78324. ** When this function is called, *ppPoslist is assumed to point to the ** start of a column-list. After it returns, *ppPoslist points to the ** to the terminator (POS_COLUMN or POS_END) byte of the column-list. *** A column-list is list of delta-encoded positions for a single column ** within a single document within a doclist. *** The column-list is terminated either by a POS_COLUMN varint (1) or ** a POS_END varint (0). This routine leaves *ppPoslist pointing to ** the POS_COLUMN or POS_END that terminates the column-list. *** If pp is not NULL, then the contents of the column-list are copied ** to *pp. *pp is set to point to the first byte past the last byte copied ** before this function returns. The POS_COLUMN or POS_END terminator ** is not copied into *pp.

78325. ** Advance the snippet iterator to the next candidate snippet.

78326. ** Deallocate internal memory used by a WhereLoop object

78327. ** Return the full pathname of the database file. *** Except, if the pager is in-memory only, then return an empty string if ** nullIfMemDb is true. This routine is called with nullIfMemDb==1 when ** used to report the filename to the user, for compatibility with legacy ** behavior. But when the Btree needs to know the

filename for matching to ** shared cache, it uses nullIfMemDb==0 so that in-memory databases can ** participate in shared-cache.

78328. Name context for sub-vdbe

78329. First try to find another RBU vfs lower down in the vfs stack. If ** one is found, this vfs will operate in pass-through mode. The lower ** level vfs will do the special RBU handling.

78330. **comment:** ** The SELECT statement iterating through the keys for the current object ** (p->objiter.pSelect) currently points to a valid row. This function ** determines the type of operation requested by this row and returns ** one of the following values to indicate the result: *** *** * RBU_INSERT *** * RBU_DELETE *** RBU_IDX_DELETE *** RBU_UPDATE *** * If RBU_UPDATE is returned, then output variable *pzMask is set to ** point to the text value indicating the columns to update. *** * If the rbu_control field contains an invalid value, an error code and *** message are left in the RBU handle and zero returned.

label: code-design

78331. **comment:** This routine is never called after sqlite3RowSetNext()

label: requirement

78332. Context that contains this SELECT

78333. Array of WhereLoop objects implementing this path

78334. Register to hold name of table

78335. List of columns to extract.

78336. Bitmask of db->aDb[] entries referenced

78337. ** Delete all information from the single table that pCur is open on. *** * This routine only work for pCur on an ephemeral table.

78338. '+'

78339. Iterates through set of root pages

78340. Values to insert into the %_idx table

78341. Total number of segments

78342. Template WhereLoop under construction

78343. ** An instance of the following structure keeps track of a mapping ** between VDBE cursor numbers and bits of the bitmasks in WhereTerm. *** * The VDBE cursor numbers are small integers contained in ** SrcList_item.iCursor and Expr.iTable fields. For any given WHERE ** clause, the cursor numbers might not begin with 0 and they might ** contain gaps in the numbering sequence. But we want to make maximum ** use of the bits in our bitmasks. This structure provides a mapping ** from the sparse cursor numbers into consecutive integers beginning ** with 0. *** * If WhereMaskSet.ix[A]==B it means that The A-th bit of a Bitmask ** corresponds VDBE cursor number B. The A-th bit of a bitmask is 1<<A. *** * For example, if the WHERE clause expression used these VDBE ** cursors: 4, 5, 8, 29, 57, 73. Then the WhereMaskSet structure ** would map those cursor numbers into bits 0 through 5. *** * Note that the mapping is not necessarily ordered. In the example ** above, the mapping might go like this: 4->3, 5->1, 8->2, 29->0, ** 57->5, 73->4. Or one of 719 other combinations might be used. It ** does not really matter. What is important is that sparse cursor ** numbers all get mapped into bit numbers that begin with 0 and contain ** no gaps.

78344. Type of test to make on this file

78345. ** Like sqlite3ExprCompare() except COLLATE operators at the top-level ** are ignored.

78346. ** 2009 March 3 *** * The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** * May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains the implementation of the

sqlite3_unlock_notify() ** API method and its associated functionality.

78347. Number of bytes in nKey

78348. ** The sqlite3_mutex_held() and sqlite3_mutex_notheld() routine are ** intended for use only inside assert() statements. On some platforms, ** there might be race conditions that can cause these routines to ** deliver incorrect results. In particular, if pthread_equal() is ** not an atomic operation, then these routines might delivery ** incorrect results. On most platforms, pthread_equal() is a ** comparison of two integers and is therefore atomic. But we are ** told that HPUX is not such a platform. If so, then these routines ** will not always work correctly on HPUX. *** * On those platforms where pthread_equal() is not atomic, SQLite ** should be compiled without -DSQLITE_DEBUG and with -DNDEBUG to ** make sure no assert() statements are evaluated and hence these ** routines are never called.

78349. Number of result rows written here

78350. OUT: Position integer of token

78351. using_opt ::=

78352. The cursor for the index

78353. Original JSON string

78354. ** CAPI3REF: Compare the ages of two snapshot handles. ** EXPERIMENTAL *** * The sqlite3_snapshot_cmp(P1, P2) interface is used to compare the ages ** of two valid snapshot handles. *** * If the two snapshot handles are not associated with the same database ** file, the result of the comparison is undefined. ** Additionally, the result of the comparison is only valid if both of the ** snapshot handles were obtained by calling sqlite3_snapshot_get() since the ** last time the wal file was deleted. The wal file is deleted when the ** database is changed back to rollback mode or when the number of database ** clients drops to zero. If either snapshot handle was obtained before the ** wal file was last deleted, the value returned by this function ** is undefined. *** * Otherwise, this API returns a negative value if P1 refers to an older ** snapshot than P2, zero if the two handles refer to the same database ** snapshot, and a positive value if P1 is a newer snapshot than P2.

78355. Assembled sorter record

78356. ** Apply posix advisory locks for all bytes from ofst through ofst+n-1. *** * Locks block if the mask is exactly UNIX_SHM_C and are non-blocking ** otherwise.

78357. ** Execute the following SQL: *** * DELETE FROM %_data WHERE id BETWEEN \$iFirst AND \$iLast

78358. Increment the nRef value on all name contexts from TopNC up to ** the point where the name matched.

78359. True for "ORDER BY rowid DESC"

78360. Prerequisites of the new entry

78361. Register the result column names for pragmas that return results

78362. **comment:** ** The sqlite3_mutex_alloc() routine allocates a new ** mutex and returns a pointer to it. If it returns NULL ** that means that a mutex could not be allocated. SQLite ** will unwind its stack and return an error. The argument ** to sqlite3_mutex_alloc() is one of these integer constants: *** * ** SQLITE_MUTEX_FAST ** SQLITE_MUTEX_RECURSIVE ** SQLITE_MUTEX_STATIC_MASTER ** SQLITE_MUTEX_STATIC_MEM ** SQLITE_MUTEX_STATIC_OPEN ** SQLITE_MUTEX_STATIC_PRNG ** SQLITE_MUTEX_STATIC_LRU ** SQLITE_MUTEX_STATIC_PMEM ** SQLITE_MUTEX_STATIC_APP1 ** SQLITE_MUTEX_STATIC_APP2 ** SQLITE_MUTEX_STATIC_APP3 ** SQLITE_MUTEX_STATIC_VFS1 ** SQLITE_MUTEX_STATIC_VFS2 ** SQLITE_MUTEX_STATIC_VFS3 ** *** * The first two constants cause sqlite3_mutex_alloc() to create ** a new mutex. The new mutex is recursive when SQLITE_MUTEX_RECURSIVE ** is used but not necessarily so when SQLITE_MUTEX_FAST is used. ** The mutex implementation does not need to make a distinction ** between SQLITE_MUTEX_RECURSIVE and SQLITE_MUTEX_FAST if it does ** not want to. But SQLite will only request a recursive mutex in ** cases where it really needs one. If a faster non-recursive mutex ** implementation is available on the host platform, the mutex subsystem ** might return such a mutex in response to SQLITE_MUTEX_FAST. *** * The other allowed parameters to sqlite3_mutex_alloc() each return ** a pointer to a static preexisting mutex. Six static mutexes are ** used by the current version of SQLite. Future versions of SQLite ** may add additional static mutexes. Static mutexes are for internal ** use by SQLite only. Applications that use SQLite mutexes should ** use only the dynamic mutexes returned by SQLITE_MUTEX_FAST or ** SQLITE_MUTEX_RECURSIVE. *** * Note that if one of the dynamic mutex parameters (SQLITE_MUTEX_FAST ** or SQLITE_MUTEX_RECURSIVE) is used then sqlite3_mutex_alloc() ** returns a different mutex on every call. But for the static ** mutex types, the same mutex is returned on every call that has ** the same type number.

label: code-design

78363. Name of the virtual table

78364. **comment:** ** Return a pointer to the "temporary page" buffer held internally ** by the pager. This is a buffer that is big enough to hold the ** entire content of a database page. This buffer is used internally ** during rollback and will be overwritten whenever a rollback ** occurs. But other modules are free to use it too, as long as ** no rollbacks are happening.

label: code-design

78365. Output cursor

78366. Next and previous elements in the table

78367. If the sorter uses more than one task, then create the top-level ** MergeEngine here. This MergeEngine will read data from exactly ** one PmaReader per sub-task.

78368. ** Designate the PRIMARY KEY for the table. pList is a list of names ** of columns that form the primary key. If pList is NULL, then the ** most recently added column of the table is the primary key. *** ** A table can have at most one primary key. If the table already has ** a primary key (and this is the second primary key) then create an ** error. *** ** If the PRIMARY KEY is on a single column whose datatype is INTEGER, ** then we will try to use that column as the rowid. Set the Table.iPKey ** field of the table under construction to be the index of the ** INTEGER PRIMARY KEY column. Table.iPKey is set to -1 if there is ** no INTEGER PRIMARY KEY. *** ** If the key is not an INTEGER PRIMARY KEY, then create a unique ** index for the key. No index is created for INTEGER PRIMARY KEYS.

78369. If Pager.errCode is set, the contents of the pager cache cannot be ** trusted. Now that there are no outstanding references to the pager, ** it can safely move back to PAGER_OPEN state. This happens in both ** normal and exclusive-locking mode.

78370. True if zBuf is static space

78371. ** Unless an "EXPLAIN QUERY PLAN" command is being processed, this function ** is a no-op. Otherwise, it adds a single row of output to the EQP result, ** where the caption is of one of the two forms: *** ** "COMPOSITE SUBQUERIES iSub1 and iSub2 (op)" ** "COMPOSITE SUBQUERIES iSub1 and iSub2 USING TEMP B-TREE (op)" *** ** where iSub1 and iSub2 are the integers passed as the corresponding ** function parameters, and op is the text representation of the parameter ** of the same name. The parameter "op" must be one of TK_UNION, TK_EXCEPT, ** TK_INTERSECT or TK_ALL. The first form is used if argument bUseTmp is ** false, or the second form if it is true.

78372. If there is data, jump to P2

78373. Should >=0 for apSub element.

78374. !defined(SQLITE_OMIT_FLOATING_POINT)

78375. Skip any leading divider characters.

78376. The column for the index

78377. For interrupt flag

78378. Net deferred immediate constraints

78379. ***** Begin file resolve.c *****

78380. List of docids for full-text queries

78381. ** 2006 June 10 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This file contains code used to help implement virtual tables.

78382. The result set that needs to be DISTINCT

78383. (317) trnm ::= nm

78384. The SELECT statement holding pOrderBy

78385. ** Two versions of the official API. Legacy and new use. In the legacy ** version, the original SQL text is not saved in the prepared statement ** and so if a schema change occurs, SQLITE_SCHEMA is returned by ** sqlite3_step(). In the new version, the original SQL text is retained ** and the statement is automatically recompiled if an schema change ** occurs.

78386. Record best loops here, if not NULL

78387. ifndef SQLITE_AMALGAMATION

78388. Any loop using an application-defined index (or PRIMARY KEY or ** UNIQUE constraint) with one or more == constraints is better ** than an automatic index. Unless it is a skip-scan.

78389. Data for the current row, if all on one page

78390. Expressions of the form: CAST(pLeft AS token)

78391. Current offset within zIn[]

78392. List may have been edited in place by fts3EvalNearTrim()

78393. Array of Vdbe cursors for parent frame

78394. Size of buffer after writing tbl header

78395. Number of valid bytes in zDb

78396. Number of bytes in a varint

78397. Number of valid fields currently in pRec

78398. At this point, both p1 and p2 point to the start of column-lists ** for the same column (the column with index iCol1 and iCol2). ** A column-list is a list of non-negative delta-encoded varints, each ** incremented by 2 before being stored. Each list is terminated by a ** POS_END (0) or POS_COLUMN (1). The following block merges the two lists ** and writes the results to buffer p. p is left pointing to the byte ** after the list written. No terminator (POS_END or POS_COLUMN) is ** written to the output.

78399. Expression of which pNode is a part

78400. 1 byte past EOF for this PmaReader

78401. New Select.pEList for RHS

78402. Free this WITH object at the end of the parse

78403. tab1 must not be a virtual table

78404. Cursor pointing to entry to read from

78405. synopsis: r[P2]=data

78406. True after snippet is shifted

78407. WHERENAMEAFTERREPLACEANDEFAULTAUTOINCREMENTCASTCOLUMNCOMMIT

78408. IMP: R-51689-46548

78409. At this point we have established that the statement is of the ** correct syntactic form to participate in this optimization. Now ** we have to check the semantics.

78410. Thousands separator for %d and %u

78411. Open cursor iTabCur+j if aToOpen[j] is true

78412. Add the entry to the main terms index.

78413. Old rowid

78414. ** Return true if the argument is non-NULL and the WAL module is using ** heap-memory for the wal-index. Otherwise, if the argument is NULL or the ** WAL module is using shared-memory, return false.

78415. ** This function is registered as the module destructor (called when an ** FTS3 enabled database connection is closed). It frees the memory ** allocated for the tokenizer hash table.

78416. ** Return the current pager state

78417. Function call context

78418. ** Maximum pathname length (in bytes) for WinNT. This should normally be ** UNICODE_STRING_MAX_CHARS * sizeof(WCHAR).

78419. ** Generate code for the ANALYZE command. The parser calls this routine ** when it recognizes an ANALYZE command. *** ** ANALYZE -- 1 ** ANALYZE <database> -- 2 ** ANALYZE ?<database>.?<tablename> -- 3 ** ** Form 1 causes all indices in all attached databases to be analyzed. ** Form 2 analyzes all indices the single database named. ** Form 3 analyzes all indices associated with the named table.

78420. ifndef SQLITE_OMIT_SHARED_CACHE

78421. Recursive representation

78422. ** Query to see if the entry in the %_segments table with blockid iEnd is ** NULL. If no error occurs and the entry is NULL, set *pbRes 1 before ** returning. Otherwise, set *pbRes to 0. *** ** Or, if an error occurs while querying the database, return an SQLite ** error code. The final value of *pbRes is undefined in this case. *** ** This is used to test if a segment is an "appendable" segment. If it ** is, then a NULL entry has been inserted into the %_segments table ** with blockid %_segdir.end_block.

78423. True if only the inner-most loop is ordered

78424. EV: R-30323-21917

78425. .xDel =

78426. WHERE_ONEROW would have been helpful

78427. Opcode: Checkpoint P1 P2 P3 * * * ** Checkpoint database P1. This is a no-op if P1 is not currently in ** WAL mode. Parameter P2 is one of SQLITE_CHECKPOINT_PASSIVE, FULL, ** RESTART, or TRUNCATE. Write 1 or 0 into mem[P3] if the checkpoint returns ** SQLITE_BUSY or not,

respectively. Write the number of pages in the ** WAL after the checkpoint into mem[P3+1] and the number of pages ** in the WAL that have been checkpointed after the checkpoint ** completes into mem[P3+2]. However on an error, mem[P3+1] and ** mem[P3+2] are initialized to -1.

78428. Id of this connection within its unixShmNode

78429. ** Write data to the file.

78430. Do an integrity check of the B-Tree ** ** Begin by finding the root pages numbers ** for all tables and indices in the database.

78431. ** Check that the entry in the pointer-map for page iChild maps to ** page iParent, pointer type ptrType. If not, append an error message ** to pCheck.

78432. ** Write page pPg onto the end of the rollback journal.

78433. Use a recursive mutex if it is available

78434. Allocate cursors numbers for Queue and Distinct. The cursor number for ** the Distinct table must be exactly one greater than Queue in order ** for the SRT_DistFifo and SRT_DistQueue destinations to work.

78435. pPager->pLast = 0;

78436. Number of assigned cursor values

78437. Do not save this cursor

78438. Check if the index cursor is past the end of the range.

78439. Check for variable references only

78440. ** Two of the file mapping APIs are different under WinRT. Figure out which ** set we need.

78441. Insert the new freeblock into the freelist

78442. ** Read a varint value from aBuf[] into *piVal. Return the number of ** bytes read.

78443. Top of the IN loop

78444. SQLITE_STMTSTATUS_AUTOINDEX

78445. if there wasn't a hash collision, and this doesn't

78446. Assumed sector size during rollback

78447. Normal case - r-tree scan. Set up the RtreeCursor.aConstraint array ** with the configured constraints.

78448. Source page size

78449. ** The argument is an RtreeCoord. Return the value stored within the RtreeCoord ** formatted as a RtreeDValue (double or int64). This macro assumes that local ** variable pRtree points to the Rtree structure associated with the ** RtreeCoord.

78450. Step 1a

78451. no space allocated, yet.

78452. ... printf arguments

78453. Insert code to test for implied constraints based on transitivity ** of the "==" operator. ** ** Example: If the WHERE clause contains "t1.a=t2.b" and "t2.b=123" ** and we are coding the t1 loop and the t2 loop has not yet coded, ** then we cannot use the "t1.a=t2.b" constraint, but we can code ** the implied "t1.a=123" constraint.

78454. ** Purge the winShmNodeList list of all entries with winShmNode.nRef==0. ** ** This is not a VFS shared-memory method; it is a utility function called ** by VFS shared-memory methods.

78455. 225

78456. Expression to consider

78457. **comment:** ** Internally, each RBU connection uses a separate SQLite database ** connection to access the target and rbu update databases. This ** API allows the application direct access to these database handles. ** ** The first argument passed to this function must be a valid, open, RBU ** handle. The second argument should be passed zero to access the target ** database handle, or non-zero to access the rbu update database handle. ** Accessing the underlying database handles may be useful in the ** following scenarios: ** ** * If any target tables are virtual tables, it may be necessary to ** call sqlite3_create_module() on the target database handle to ** register the required virtual table implementations. ** ** * If the data_xxx tables in the RBU source database are virtual ** tables, the application may need to call sqlite3_create_module() on ** the rbu update db handle to any required virtual table ** implementations. ** ** * If the application uses the "rbu_delta()" feature described above, ** it must use sqlite3_create_function() or similar to register the ** rbu_delta() implementation with the target database handle. ** ** If an error has occurred, either while opening or stepping the RBU object, ** this function may return NULL. The error code and message may be collected ** when sqlite3rbu_close() is called. ** ** Database handles returned by this function remain valid until the next ** call to any sqlite3rbu_xxx() function other than sqlite3rbu_db().

label: code-design

78458. 25

78459. Index to match column of

78460. Name of this CTE

78461. 1st argument

78462. Logarithm of the number of rows in the table

78463. Do not attempt to change the page size for a WAL database

78464. ** Return the number of references to the page supplied as an argument.

78465. If everything went according to plan, link the new VTable structure ** into the linked list headed by pTab->pVTable. Then loop through the ** columns of the table to see if any of them contain the token "hidden". ** If so, set the Column COLFLAG_HIDDEN flag and remove the token from ** the type string.

78466. Opcode: CollSeq P1 * * P4 ** ** P4 is a pointer to a CollSeq object. If the next call to a user function ** or aggregate calls sqlite3GetFuncCollSeq(), this collation sequence will ** be returned. This is used by the built-in min(), max() and nullif() ** functions. ** ** If P1 is not zero, then it is a register that a subsequent min() or ** max() aggregate will set to 1 if the current row is not the minimum or ** maximum. The P1 register is initialized to 0 by this instruction. ** ** The interface used by the implementation of the aforementioned functions ** to retrieve the collation sequence set by this opcode is not available ** publicly. Only built-in functions have access to this feature.

78467. VDBE has already been allocated

78468. IMP: R-06824-28531

78469. p cannot be part of a CHECK constraint

78470. Allocate a cursors for the main database table and for all indices. ** The index cursors might not be used, but if they are used they ** need to occur right after the database cursor. So go ahead and ** allocate enough space, just in case.

78471. OUT: "Score" for this snippet

78472. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, it appends the serialized version of the value stored ** in column iCol of the row that SQL statement pStmt currently points ** to to the buffer.

78473. In-memory list of records

78474. ** standard include files.

78475. mx_payload

78476. ***** Include whereInt.h in the middle of wherecode.c *****

78477. ** Allocate a new page object initially associated with cache pCache.

78478. This subquery can be absorbed into its parent.

78479. OUT: New changeset iterator handle

78480. ** Rtree virtual table module xDisconnect method.

78481. A malloc() failed

78482. True if fts5yymajor has invoked an error

78483. void*, int sz, int N

78484. If the user has inserted a NULL value, this function may be called with ** zText==0. In this case, add zero token entries to the hash table and ** return early.

78485. **comment:** ** Return a pointer to the appropriate hash function given the key class. ** ** The C syntax in this function definition may be unfamiliar to some ** programmers, so we provide the following additional explanation: ** ** The name of the function is "ftsHashFunction". The function takes a ** single parameter "keyClass". The return value of ftsHashFunction() ** is a pointer to another function. Specifically, the return value ** of ftsHashFunction() is a pointer to a function that takes two parameters ** with types "const void*" and "int" and returns an "int".

label: code-design

78486. If a HAVING clause is present, then there must be a GROUP BY clause.

78487. OUT: Height

78488. If an older version of the function with a configured destructor is ** being replaced invoke the destructor function here.
78489. Reg with value of this column. 0 means none.
78490. Previous value read from pgidx
78491. Which cookie to read or write
78492. If a WAL transaction is being committed, there is no point in writing ** any pages with page numbers greater than nTruncate into the WAL file. ** They will never be read by any client. So remove them from the pDirty ** list here.
78493. Page size of main database
78494. Userid for the file
78495. Hour and minutes
78496. READ_LOCK or WRITE_LOCK
78497. ** To compile without implementing sqlite3Hwtime() for your platform, ** you can remove the above #error and use the following ** stub function. You will lose timing support for many ** of the debugging and testing utilities, but it should at ** least compile and run.
78498. If pCte->zCteErr is non-NULL at this point, then this is an illegal ** recursive reference to CTE pCte. Leave an error in pParse and return ** early. If pCte->zCteErr is NULL, then this is not a recursive reference. ** In this case, proceed.
78499. If the following assert() fails on some obscure processor/compiler ** combination, the work-around is to set the correct pointer ** size at compile-time using -DSQLITE_PTRSIZE=n compile-time option
78500. xShrink
78501. True for a lookup only
78502. ** After allocating the Fts3Expr.aMI[] array for each phrase in the ** expression rooted at pExpr, the cursor iterates through all rows matched ** by pExpr, calling this function for each row. This function increments ** the values in Fts3Expr.aMI[] according to the position-list currently ** found in Fts3Expr.pPhrase->doclist.pList for each of the phrase ** expression nodes.
78503. Must match this affinity, if zCollName!=NULL
78504. True if satisfies ORDER BY
78505. !defined(SQLITE_OMIT_SUBQUERY) || !defined(SQLITE_OMIT_VIEW)
78506. Waiting for read-locks to clear
78507. If the child node is now at EOF, so is the parent AND node. Otherwise, ** the child node is guaranteed to have advanced at least as far as ** rowid iLast. So if it is not at exactly iLast, pChild->iRowid is the ** new lastest rowid seen so far.
78508. **comment:** If the buffer currently allocated is too small for this entry, realloc ** the buffer to make it large enough.
label: code-design
78509. Set idxFlags flags for all WHERE clause terms that will be used.
78510. xConnect
78511. Used to iterate through p->aSample[]
78512. Opcode: InitCoroutine P1 P2 P3 * * * * Set up register P1 so that it will Yield to the coroutine ** located at address P3. ** * * If P2!=0 then the coroutine implementation immediately follows ** this opcode. So jump over the coroutine implementation to ** address P2. ** * * See also: EndCoroutine
78513. Hash table
78514. No shared-memory support
78515. **comment:** ** The sqlite3_mutex_leave() routine exits a mutex that was ** previously entered by the same thread. The behavior ** is undefined if the mutex is not currently entered or ** is not currently allocated. SQLite will never do either.
label: code-design
78516. If this is a delete operation to remove a row from a table b-tree, ** invalidate any incrblob cursors open on the row being deleted.
78517. forward declaration
78518. Control jumps to here if an error is encountered above, or upon ** successful coding of the SELECT.
78519. Checkpoint information in wal-index
78520. True if the OP_Affinity operation has been run
78521. ** Find the smallest page number out of all pages held in the WAL that ** has not been returned by any prior invocation of this method on the ** same WalIterator object. Write into *piFrame the frame index where ** that page was last written into the WAL. Write into *piPage the page ** number. ** * * Return 0 on success. If there are no pages in the WAL with a page ** number larger than *piPage, then return 1.
78522. **comment:** The word is too big or too small for the porter stemmer. ** Fallback to the copy stemmer
label: documentation
78523. ** This function is the implementation of both the xConnect and xCreate ** methods of the FTS3 virtual table. ** * * The argv[] array contains the following: ** ** argv[0] -> module name ("fts5") ** argv[1] -> database name ** argv[2] -> table name ** argv[...] -> "column name" and other module argument fields.
78524. OUT: Size of blob data
78525. Initialize sqlite3_vtab_cursor base class
78526. ** For all dirty pages currently in the cache, invoke the specified ** callback. This is only used if the SQLITE_CHECK_PAGES macro is ** defined.
78527. Set to " AND " later on
78528. ** Free the RtreeCursor.aConstraint[] array and its contents.
78529. **comment:** note that the quality of the randomness doesn't matter that much
label: code-design
78530. Position list
78531. ** PRAGMA threads ** PRAGMA threads = N ** * * Configure the maximum number of worker threads. Return the new ** maximum, which might be less than requested.
78532. ** Write a 32-bit integer into a string buffer in big-endian byte order.
78533. Cursor for the canonical data btree
78534. ** Assuming zIn points to the first byte of a UTF-8 character, ** advance zIn to point to the first byte of the next UTF-8 character.
78535. ** This routine sets the busy callback for an Sqlite database to the ** given callback function with the given argument.
78536. True if xText is allocated space
78537. 1460
78538. Prior select in a compound select statement
78539. Global shared lock memory for the file
78540. The isRequirePhrase variable is set to true if a phrase or ** an expression contained in parenthesis is required. If a ** binary operator (AND, OR, NOT or NEAR) is encountered when ** isRequirePhrase is set, this is a syntax error.
78541. Left operand: 0==FALSE, 1==TRUE, 2==UNKNOWN or NULL
78542. ** This assert appears to trigger spuriously on certain ** versions of Windows, possibly due to _beginthreadex() ** and/or CreateThread() not fully setting their thread ** ID parameter before starting the thread.
78543. ** Return true if character 't' may be part of an FTS5 bareword, or false ** otherwise. Characters that may be part of barewords: ** * * * All non-ASCII characters, ** * * The 52 upper and lower case ASCII characters, and ** * The 10 integer ASCII characters. ** * The underscore character "_" (0x5F). ** * The unicode "substitute" character (0x1A).
78544. Copy the data from the source page into the destination page. ** Then clear the Btree layer MemPage.isInit flag. Both this module ** and the pager code use this trick (clearing the first byte ** of the page 'extra' space to invalidate the Btree layers ** cached parse of the page). MemPage.isInit is marked ** "MUST BE FIRST" for this purpose.
78545. True if at End Of Results
78546. ** Argument rc is an SQLite error code. Return true if this error is ** considered fatal if encountered during a backup operation. All errors ** are considered fatal except for SQLITE_BUSY and SQLITE_LOCKED.
78547. SQLITE_DISABLE_SKIPAHEAD_DISTINCT
78548. ** This implementation does not actually create a new thread. It does the ** work of the thread in the main thread, when either the thread is created ** or when it is joined
78549. Size of string aInput in bytes
78550. ** Register a statically linked extension that is automatically ** loaded by every new database connection.

78551. Name of the collating sequence
78552. ** When this global variable is positive, it gets decremented once before ** each instruction in the VDBE. When it reaches zero, the u1.isInterrupted ** field of the sqlite3 structure is set in order to simulate an interrupt. ** ** This facility is used for testing purposes only. It does not function ** in an ordinary build.
78553. ** This array defines hard upper bounds on limit values. The ** initializer must be kept in sync with the SQLITE_LIMIT_* ** #defines in sqlite3.h.
78554. Number of entries in aLTerm[]
78555. Delete and then insert a row
78556. Zero or more SORTFLAG_* bits
78557. Allocate an instance of struct Wal to return.
78558. Add the new segment to the output level
78559. ** Change the "spill" limit on the number of pages in the cache. ** If the number of pages exceeds this limit during a write transaction, ** the pager might attempt to "spill" pages to the journal early in ** order to free up memory. ** ** The value returned is the current spill size. If zero is passed ** as an argument, no changes are made to the spill size setting, so ** using mxPage of 0 is a way to query the current spill size.
78560. Iterator to update aFirst[] array for
78561. One slot for each expression in the list
78562. Parse number
78563. **comment:** ** The xGetLastError() method is designed to return a better ** low-level error message when operating-system problems come up ** during SQLite operation. Only the integer return code is currently ** used.
 label: code-design
78564. ** Check on a Vdbe to make sure it has not been finalized. Log ** an error and return true if it has been finalized (or is otherwise ** invalid). Return false if it is ok.
78565. Mix sz bytes of entropy into p.
78566. ***** End of malloc.c *****
78567. The connection to be closed
78568. End offset of token
78569. ** Formulate and prepare an SQL statement to query table zTab by primary ** key. Assuming the following table structure: *** CREATE TABLE x(a, b, c, d, PRIMARY KEY(a, c)); ** ** The SELECT statement looks like this: *** ** SELECT * FROM x WHERE a = ?1 AND c = ?3 ** ** If successful, SQLITE_OK is returned and SessionApplyCtx.pSelect is left ** pointing to the prepared version of the SQL statement.
78570. Schema format version for this file
78571. Debug EXPLAIN QUERY PLAN
78572. 530
78573. ** Compute the affinity string for table pTab, if it has not already been ** computed. As an optimization, omit trailing SQLITE_AFF_BLOB affinities. ** ** If the affinity exists (if it is not entirely SQLITE_AFF_BLOB values) and ** if iReg>0 then code an OP_Affinity opcode that will set the affinities ** for register iReg and following. Or if affinities exists and iReg==0, ** then just set the P4 operand of the previous opcode (which should be ** an OP_MakeRecord) to the affinity string. ** ** A column affinity string has one character per column: *** ** Character Column affinity ** ----- ** 'A' BLOB ** 'B' TEXT ** 'C' NUMERIC ** 'D' INTEGER ** 'E' REAL
78574. OUT: Global maximum cache size
78575. Quote character
78576. In the normal case (8+3 filenames disabled) the journal filename ** is guaranteed to contain a '-' character.
78577. ** SQL statement pSelect is as generated by the sessionSelectRow() function. ** This function binds the primary key values from the change that changeset ** iterator pIter points to the SELECT and attempts to seek to the table ** entry. If a row is found, the SELECT statement left pointing at the row ** and SQLITE_ROW is returned. Otherwise, if no row is found and no error ** has occurred, the statement is reset and SQLITE_OK is returned. If an ** error occurs, the statement is reset and an SQLite error code is returned. ** ** If this function returns SQLITE_ROW, the caller must eventually reset() ** statement pSelect. If any other value is returned, the statement does ** not require a reset(). ** ** If the iterator currently points to an INSERT record, bind values from the ** new.* record to the SELECT statement. Or, if it points to a DELETE or ** UPDATE, bind values from the old.* record.
78578. ** Return ONEPASS_OFF (0) if an UPDATE or DELETE statement is unable to ** operate directly on the rows returned by a WHERE clause. Return ** ONEPASS_SINGLE (1) if the statement can operate directly because only ** a single row is to be changed. Return ONEPASS_MULTI (2) if the one-pass ** optimization can be used on multiple ** ** If the ONEPASS optimization is used (if this routine returns true) ** then also write the indices of open cursors used by ONEPASS ** into aiCur[0] and aiCur[1]. iaCur[0] gets the cursor of the data ** table and iaCur[1] gets the cursor used by an auxiliary index. ** Either value may be -1, indicating that cursor is not used. ** Any cursors returned will have been opened for writing. ** ** aiCur[0] and aiCur[1] both get -1 if the where-clause logic is ** unable to use the ONEPASS optimization.
78579. Done.
78580. ** Sync the database file to disk. This is a no-op for in-memory databases ** or pages with the Pager.noSync flag set. ** ** If successful, or if called on a pager for which it is a no-op, this ** function returns SQLITE_OK. Otherwise, an IO error code is returned.
78581. Save the left position
78582. Iterator opened on pData/nData
78583. Add one to the old schema cookie
78584. REPLACE
78585. ** This function is a no-op if *pRc is other than SQLITE_OK when it is ** called. Otherwise, append a single varint to the buffer. ** ** If an OOM condition is encountered, set *pRc to SQLITE_NOMEM before ** returning.
78586. The ON clause of a join
78587. QUERY => ID
78588. Source page number
78589. Length of each character in zCharSet
78590. 0==rowid, 1==tbl, 2==rank
78591. xRollbackTo
78592. Column numbers
78593. Sum of nMin for purgeable caches
78594. Zero the file-handle object. If nSpill was passed zero, initialize ** it using the sqlite3OsOpen() function of the underlying VFS. In this ** case none of the code in this module is executed as a result of calls ** made on the journal file-handle.
78595. The VFS to use for actual file I/O
78596. 321
78597. Array of integers that becomes the BLOB
78598. ifdef SQLITE_ENABLE_STAT4
78599. True for a SEARCH. False for SCAN.
78600. ** Delete the given Select structure and all of its substructures.
78601. Gather the complete text of the CREATE INDEX statement into ** the zStmt variable
78602. ** Set the ExprList.a[] .zSpan element of the most recently added item ** on the expression list. ** ** pList might be NULL following an OOM error. But pSpan should never be ** NULL. If a memory allocation fails, the pParse->db->mallocFailed flag ** is set.
78603. ***** Utility routines for dealing with JsonNode and JsonParse objects *****
78604. ** Resize the Vdbe.aOp array so that it is at least nOp elements larger ** than its current size. nOp is guaranteed to be less than or equal ** to 1024/sizeof(Op). ** ** If an out-of-memory error occurs while resizing the array, return ** SQLITE_NOMEM. In this case Vdbe.aOp and Parse.nOpAlloc remain ** unchanged (this is so that any opcodes already allocated can be ** correctly deallocated along with the rest of the Vdbe).
78605. LIMIT
78606. Jump here if malloc fails
78607. Name of the identifier
78608. Malloced string that needs to be freed
78609. Take the requested lock on the conch file and break a stale lock if the ** host id matches.

78610. ** Implementation of bm25() function.
78611. Rowid of the next row to insert
78612. ** A foreign key constraint requires that the key columns in the parent ** table are collectively subject to a UNIQUE or PRIMARY KEY constraint. ** Given that pParent is the parent table for foreign key constraint pFKey, ** search the schema for a unique index on the parent key columns. ** ** If successful, zero is returned. If the parent key is an INTEGER PRIMARY ** KEY column, then output variable *ppIdx is set to NULL. Otherwise, *ppIdx ** is set to point to the unique index. ** ** If the parent key consists of a single column (the foreign key constraint ** is not a composite foreign key), output variable *paiCol is set to NULL. ** Otherwise, it is set to point to an allocated array of size N, where ** N is the number of columns in the parent key. The first element of the ** array is the index of the child table column that is mapped by the FK ** constraint to the parent table column stored in the left-most column ** of index *ppIdx. The second element of the array is the index of the ** child table column that corresponds to the second left-most column of ** *ppIdx, and so on. ** ** If the required index cannot be found, either because: ** ** 1) The named parent key columns do not exist, or ** ** 2) The named parent key columns do exist, but are not subject to a ** UNIQUE or PRIMARY KEY constraint, or ** ** 3) No parent key columns were provided explicitly as part of the ** foreign key definition, and the parent table does not have a ** PRIMARY KEY, or ** ** 4) No parent key columns were provided explicitly as part of the ** foreign key definition, and the PRIMARY KEY of the parent table ** consists of a different number of columns to the child key in ** the child table. ** ** then non-zero is returned, and a "foreign key mismatch" error loaded ** into pParse. If an OOM error occurs, non-zero is returned and the ** pParse->db->mallocFailed flag is set.
78613. Bytes of space allocated at pSpace
78614. True if this is an in-memory file
78615. ** pln is a UTF-8 encoded string, nIn bytes in size. Return the number of ** unicode characters in the string.
78616. Find the firstest rowid any synonym points to.
78617. Bytes content per ovfl page
78618. ** The select statement passed as the first argument is an aggregate query. ** The second argument is the associated aggregate-info object. This ** function tests if the SELECT is of the form: ** ** SELECT count(*) FROM <tbl> ** ** where table is a database table, not a sub-select or view. If the query ** does match this pattern, then a pointer to the Table object representing ** <tbl> is returned. Otherwise, 0 is returned.
78619. Full pathname of this file
78620. **comment:** If this is an xCreate call, create the underlying tables in the ** database. TODO: For xConnect(), it could verify that said tables exist.
label: code-design
78621. the FROM clause -- which tables to scan
78622. Write to the last byte of each newly allocated or extended page
78623. The virtual table
78624. ** Tokenization callback used by integrity check.
78625. Add the current term to the parent node. The term added to the ** parent must: ** ** a) be greater than the largest term on the leaf node just written ** to the database (still available in pLeaf->key), and ** ** b) be less than or equal to the term about to be added to the new ** leaf node (zTerm/nTerm). ** ** In other words, it must be the prefix of zTerm 1 byte longer than ** the common prefix (if any) of zTerm and pWriter->zTerm.
78626. ** If the STATIC_MEM mutex is not already held, obtain it now. The mutex ** will already be held (obtained by code in malloc.c) if ** sqlite3GlobalConfig.bMemStat is true.
78627. Check the following statements are true: ** ** (a) Exactly one of the READWRITE and READONLY flags must be set, and ** (b) if CREATE is set, then READWRITE must also be set, and ** (c) if EXCLUSIVE is set, then CREATE must also be set. ** (d) if DELETEONCLOSE is set, then CREATE must also be set.
78628. Fts5 table configuration
78629. Delta to apply to the pattern
78630. ** Iterate through all phrase nodes in an FTS3 query, except those that ** are part of a sub-tree that is the right-hand-side of a NOT operator. ** For each phrase node found, the supplied callback function is invoked. ** ** If the callback function returns anything other than SQLITE_OK, ** the iteration is abandoned and the error code returned immediately. ** Otherwise, SQLITE_OK is returned after a callback has been made for ** all eligible phrase nodes.
78631. Flags for sqlite3_vfs.xOpen()
78632. First index cursor
78633. An index being inserted or updated
78634. Keywords and unquoted identifiers
78635. 61
78636. !defined(SQLITE_CORE) || defined(SQLITE_ENABLE_RBU)
78637. ** 2003 October 31 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains the C functions that implement date and time **
functions for SQLite. ** ** There is only one exported symbol in this file - the function ** sqlite3RegisterDateTimeFunctions() found at the bottom of the file. **
All other code has file scope. ** ** SQLite processes all times and dates as Julian day numbers. The ** dates and times are stored as the number of days since
noon ** in Greenwich on November 24, 4714 B.C. according to the Gregorian ** calendar system. *** 1970-01-01 00:00:00 is JD 2440587.5 ** 2000-01-01
00:00:00 is JD 2451544.5 ** ** This implementation requires years to be expressed as a 4-digit number ** which means that only dates between 0000-01-01 and
9999-12-31 can ** be represented, even though Julian day numbers allow a much wider ** range of dates. ** ** The Gregorian calendar system is used for all
dates and times, ** even those that predate the Gregorian calendar. Historians usually ** use the Julian calendar for dates prior to 1582-10-15 and for some **
dates afterwards, depending on locale. Beware of this difference. ** ** The conversion algorithms are implemented based on descriptions ** in the following text:
*** Jean Meeus ** Astronomical Algorithms, 2nd Edition, 1998 ** ISBN 0-943396-61-1 ** Willmann-Bell, Inc ** Richmond, Virginia (USA)
78638. Largest docid for all iterators
78639. Literal "new" token
78640. 264
78641. #ifndef SQLITE_OMIT_TRACE
78642. The value of the INTEGER PRIMARY KEY column is always a NULL. ** Whenever this column is read, the rowid will be substituted ** in its place. Hence, fill
this column with a NULL to avoid ** taking up data space with information that will never be used. ** As there may be shallow copies of this value, make it a
soft-NULL
78643. Number of fields in the header
78644. SQLITE_JOCAP_POWERSAFE_OVERWRITE
78645. 253
78646. Opcode: CursorHint P1 ** P4 * *** ** Provide a hint to cursor P1 that it only needs to return rows that ** satisfy the Expr in P4. TK_REGISTER terms in the P4
expression refer ** to values currently held in registers. TK_COLUMN terms in the P4 ** expression refer to columns in the b-tree to which cursor P1 is pointing.
78647. Number of bytes in rhs input
78648. ** Allocate a new, empty, sqlite3_changegroup.
78649. The table that is indexed
78650. ** Return the amount cell p would grow by if it were unioned with pCell.
78651. Serial type to deserialize
78652. Database page-size in bytes
78653. Opcode: Variable P1 P2 * P4 * *** Synopsis: r[P2]=parameter(P1,P4) *** Transfer the values of bound parameter P1 into register P2 *** If the parameter is
named, then its name appears in P4. ** The P4 value is used by sqlite3_bind_parameter_name().
78654. ***** End of sqlite3rbu.c *****
78655. Name of this column, \000, then the type
78656. Unsorted cost of (pFrom+pWLoop)
78657. Output variables. aPoslist==0 at EOF
78658. FTS5_STRING or FTS5_TERM node
78659. Database that contains the table
78660. True if incr-vacuum is enabled
78661. 0x00..0x0F

78662. ** Compute the union of two position lists. The output written ** into *pp contains all positions of both *pp1 and *pp2 in sorted ** order and with any duplicates removed. All pointers are ** updated appropriately. The caller is responsible for insuring ** that there is enough space in *pp to hold the complete output.

78663. either the conch didn't match or we need to create a new one

78664. Do not open write-ahead-log files

78665. An engine for executing database bytecode

78666. ** Return the size of the header added by this middleware layer ** in the page-cache hierarchy.

78667. The pH containing "elem"

78668. Index of previous free chunk

78669. if (a) the open flags are not O_RDWR, (b) the conch isn't there, and ** (c) the file system is read-only, then enable no-locking access. ** Ugh, since O_RDONLY==0x0000 we test for !O_RDWR since unixOpen asserts ** that openFlags will have only one of O_RDONLY or O_RDWR.

78670. Result of strlen(zFts3)

78671. ** Lock an rbuVfs-file.

78672. Name of child table

78673. xRollback

78674. **comment:** ** There are various methods for file locking used for concurrency ** control: ** ** 1. POSIX locking (the default), ** 2. No locking, ** 3. Dot-file locking, ** 4. flock() locking, ** 5. AFP locking (OSX only), ** 6. Named POSIX semaphores (VXWorks only), ** 7. proxy locking. (OSX only) ** ** Styles 4, 5, and 7 are only available of SQLITE_ENABLE_LOCKING_STYLE ** is defined to 1. The SQLITE_ENABLE_LOCKING_STYLE also enables automatic ** selection of the appropriate locking style based on the filesystem ** where the database is located.
label: code-design

78675. Cursor number of the source table

78676. trigger_event ::= UPDATE OF idlist

78677. Check that the leaf page indicated by the iterator really does ** contain the rowid suggested by the same.

78678. The following are used by the fts3_eval.c module.

78679. OUT: Rank function arguments

78680. Caution: pRoot may iterate through docids in ascending or descending ** order. For this reason, even though it seems more defensive, the ** do loop can not be written: *** do {...} while(pRoot->iDocid<iDocid && rc==SQLITE_OK);

78681. List of columns in pTableName to insert into

78682. In-memory chunk-size

78683. If already in the error state, this function is a no-op. But on ** the other hand, this routine is never called if we are already in ** an error state.

78684. values ::= VALUES LP nxprlist RP

78685. ** Read a varint from the stream of data accessed by p. Set *pnOut to ** the value read.

78686. Search the hash table for an existing record for this row.

78687. R-tree structure

78688. ** weekday N ** ** Move the date to the same time on the next occurrence of ** weekday N where 0==Sunday, 1==Monday, and so forth. If the ** date is already on the appropriate weekday, this is a no-op.

78689. Use this text encoding

78690. Steps 2 through 4.

78691. Opcode: RealAffinity P1 * * * * * * * If register P1 holds an integer convert it to a real value. ** ** This opcode is used when extracting information from a column that ** has REAL affinity. Such column values may still be stored as ** integers, for space efficiency, but after extraction we want them ** to have only a real value.

78692. ** Ensure that there is room in the buffer to append nByte bytes of data. ** If not, use sqlite3_realloc() to grow the buffer so that there is. ** ** If successful, return zero. Otherwise, if an OOM condition is encountered, ** set *pRc to SQLITE_NOMEM and return non-zero.

78693. Data to write

78694. ***** End of select.c *****

78695. To fully unlock the database, delete the lock file

78696. trigger_time ::= INSTEAD OF

78697. ** Compute a string for the "comment" field of a Vdbe opcode listing. ** ** The Synopsis: field in comments in the vdbe.c source file gets converted ** to an extra string that is appended to the sqlite3OpcodeName(). In the ** absence of other comments, this synopsis becomes the comment on the opcode. ** Some translation occurs: ** ** "PX" -> "r[X]" ** "PX@PY" -> "r[X..X+Y-1]" or "r[x]" if y is 0 or 1 ** "PX@PY+1" -> "r[X..X+Y]" or "r[x]" if y is 0 ** "PY..PY" -> "r[X..Y]" or "r[x]" if y<=x

78698. Virtual terms from the LIKE optimization

78699. Configure the number of columns. Configure the cursor to ** think that the table has one more column than it really ** does. An OP_Column to retrieve this imaginary column will ** always return an SQL NULL. This is useful because it means ** we can invoke OP_Column to fill in the vdbe cursors type ** and offset cache without causing any IO.

78700. Outputs

78701. SELECT statement or RHS of INSERT INTO SELECT ...

78702. Figure out whether to use 1, 2, 4, 6 or 8 bytes.

78703. sqlite3_config() does not allow otherwise

78704. Source of content

78705. expr ::= expr AND expr

78706. Table object to append header for

78707. Register for ephem table rowid

78708. Level of segments

78709. ** Create new user functions.

78710. Counter to limit the number of searches

78711. exprlist ::= exprlist cnearset

78712. Generate code to handle the case of A<B

78713. ***** Interface to automatically generated code in fts5_unicode2.c.

78714. Context for returning result/error

78715. Ensure that page 0 of the wal-index (the page that contains the ** wal-index header) is mapped. Return early if an error occurs here.

78716. Used to iterate through deferred tokens

78717. ** Lists of free blocks. aiFreelist[0] is a list of free blocks of ** size mem5.szAtom. aiFreelist[1] holds blocks of size szAtom*2. ** aiFreelist[2] holds free blocks of size szAtom*4. and so forth.

78718. ** Functions for accessing sqlite3_vfs methods

78719. same as TK_LSHIFT, synopsis: r[P3]=r[P2]<r[P1]

78720. try to match the database file R/W permissions, ignore failure

78721. DELETE_DOCSIZE

78722. Vdbe instruction address of the start of the loop

78723. The size of a lookaside slot after ROUNDOWN8 needs to be larger ** than a pointer to be useful.

78724. If the database is zero pages in size, that means that either (1) the ** journal is a remnant from a prior database with the same name where ** the database file but not the journal was deleted, or (2) the initial ** transaction that populates a new database is being rolled back. ** In either case, the journal file can be deleted. However, take care ** not to delete the journal file if it is already open due to ** journal_mode=PERSIST.

78725. ** Return the name of the first system call after zName. If zName==NULL ** then return the name of the first system call. Return NULL if zName ** is the last system call or if zName is not the name of a valid ** system call.

78726. Changeset blob

78727. pagetype

78728. ** Open the shared-memory area associated with database file pDbFd. ** ** When opening a new shared-memory file, if no other instances of that ** file are currently open, in this process or in other processes, then ** the file must be truncated to zero length or have its header cleared.

78729. ** CAPI3REF: Number Of Columns In A Result Set ** METHOD: sqlite3_stmt *** ^Return the number of columns in the result set returned by the ** [prepared statement]. ^If this routine returns 0, that means the ** [prepared statement] returns no data (for example an [UPDATE]). ** ^However, just because this routine returns a positive number does not ** mean that one or more rows of data will be returned. ^A SELECT statement ** will always have a positive sqlite3_column_count() but depending on the ** WHERE clause constraints and the table content, it might return no rows. ** ** See also: [sqlite3_data_count()]

78730. ***** The following three functions, found below: *** ** rbuDeltaGetInt()

** rbuDeltaChecksum() ** rbuDeltaApply() ** ** are lifted from the fossil source code (<http://fossil-scm.org>). They ** are used to implement the scalar SQL function rbu_fossil_delta().

78731. ** Round up a request size to the next valid allocation size. If ** the allocation is too large to be handled by this allocation system, ** return 0. ** ** All allocations must be a power of two and must be expressed by a ** 32-bit signed integer. Hence the largest allocation is 0x40000000 ** or 1073741824 bytes.

78732. Append the prefix-compressed term and doclist to the buffer.

78733. Every page has an associated PCache

78734. EVIDENCE-OF: R-58015-48175 The two-byte integer at offset 5 designates ** the start of the cell content area. A zero value for this integer is ** interpreted as 65536.

78735. 73

78736. Journal omitted.

78737. ** Return the current wal-index header checksum for the target database ** as a 64-bit integer. ** ** The checksum is store in the first page of xShmMap memory as an 8-byte ** blob starting at byte offset 40.

78738. ** Step the RBU object.

78739. ***** Begin file parse.h *****

78740. Initialize the contents of the SnippetIter object. Then iterate through ** the set of phrases in the expression to populate the aPhrase[] array.

78741. Set the output variables and return.

78742. Number of bytes in aRec

78743. A virtual table that is constrained by an IN clause may not ** consume the ORDER BY clause because (1) the order of IN terms ** is not necessarily related to the order of output terms and ** (2) Multiple outputs from a single IN value will not merge ** together.

78744. Mask off the SQLITE_SYNC_* values

78745. ** This routine "expands" a SELECT statement and all of its subqueries. ** For additional information on what it means to "expand" a SELECT ** statement, see the comment on the selectExpand worker callback above. ** ** Expanding a SELECT statement is the first step in processing a ** SELECT statement. The SELECT statement must be expanded before ** name resolution is performed. ** ** If anything goes wrong, an error message is written into pParse. ** The calling function can detect the problem by looking at pParse->nErr ** and/or pParse->db->mallocFailed.

78746. 245

78747. ** Create a new index for an SQL table. pName1.pName2 is the name of the index ** and pTblList is the name of the table that is to be indexed. Both will ** be NULL for a primary key or an index that is created to satisfy a ** UNIQUE constraint. If pTable and pIndex are NULL, use pParse->pNewTable ** as the table to be indexed. pParse->pNewTable is a table that is ** currently being constructed by a CREATE TABLE statement. ** ** pList is a list of columns to be indexed. pList will be NULL if this ** is a primary key or unique-constraint on the most recent column added ** to the table currently under construction.

78748. Set the *pmSeen output variable.

78749. Query the schema of the main database. Create a mirror schema ** in the temporary database.

78750. We are trying for an exclusive lock but another thread in this ** same process is still holding a shared lock.

78751. xUnpin

78752. ** The StrAccum "p" is not large enough to accept N new bytes of z[]. ** So enlarge if first, then do the append. ** ** This is a helper routine to sqlite3StrAccumAppend() that does special-case ** work (enlarging the buffer) using tail recursion, so that the ** sqlite3StrAccumAppend() routine can use fast calling semantics.

78753. CONTINUE and BREAK addresses

78754. Name of the table we are looking for

78755. ***** End of fts3_tokenize_vtab.c *****

78756. Allocate space for the new multi-seg-iterator.

78757. If this is the end of a transaction, then we might need to pad ** the transaction and/or sync the WAL file. ** ** Padding and syncing only occur if this set of frames complete a ** transaction and if PRAGMA synchronous=FULL. If synchronous==NORMAL ** or synchronous==OFF, then no padding or syncing are needed. ** ** If SQLITE_IOCAP_POWERSAFE_OVERWRITE is defined, then padding is not ** needed and only the sync is done. If padding is needed, then the ** final frame is repeated (with its commit mark) until the next sector ** boundary is crossed. Only the part of the WAL prior to the last ** sector boundary is synced; the part of the last frame that extends ** past the sector boundary is written after the sync.

78758. Only allow tracing if SQLITE_DEBUG is defined.

78759. (274) cmdlist ::= ecmd (OPTIMIZED OUT)

78760. Number of buffers currently checked out

78761. Symbolic name of this SELECT use for debugging

78762. Delete a row from an aux. index b-tree

78763. ** Return the number of bytes required to encode v as a varint

78764. Code the SELECTs to our left into temporary table "tab1".

78765. ** 2008 May 26 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This header file is used by programs that want to link against the ** RTREE library. All it does is declare the sqlite3RtreeInit() interface.

78766. Number of bytes of the string text to include in output

78767. **comment:** ** This is called once for each leaf page except the first that contains ** at least one term. Argument (nTerm/pTerm) is the split-key - a term that ** is larger than all terms written to earlier leaves, and equal to or ** smaller than the first term on the new leaf. ** ** If an error occurs, an error code is left in Fts5Index.rc. If an error ** has already occurred when this function is called, it is a no-op.

label: code-design

78768. Buffer to assemble frame-headers in

78769. Reports from the internet are that performance is always ** better if FILE_FLAG_RANDOM_ACCESS is used. Ticket #2699.

78770. The auto-commit flag is true, the vdbe program was successful ** or hit an 'OR FAIL' constraint and there are no deferred foreign ** key constraints to hold up the transaction. This means a commit ** is required.

78771. Cell index within the node

78772. end-if not one-row

78773. ** Each page of the wal-index mapping contains a hash-table made up of ** an array of HASHTABLE_NSLOT elements of the following type.

78774. If the P3 value could not be converted into an integer without ** loss of information, then special processing is required...

78775. Input buffer or stream

78776. Mask of ORDER BY terms that need reversing

78777. EVIDENCE-OF: R-51873-39618 The page size for a database file is ** determined by the 2-byte integer located at an offset of 16 bytes from ** the beginning of the database file.

78778. 9,10,11

78779. First argument for conflict handler

78780. Expression (pLeft != pRight)

78781. If the root node was just loaded, set pRtree->iDepth to the height ** of the r-tree structure. A height of zero means all data is stored on ** the root node. A height of one means the children of the root node ** are the leaves, and so on. If the depth as specified on the root node ** is greater than RTREE_MAX_DEPTH, the r-tree structure must be corrupt.

78782. The x>=y term

78783. Replace with JsonNode.u.iReplace

78784. Address of the OP_IfNot opcode

78785. Minimum local payload in a LEAFDATA table

78786. Create a SrcList structure containing the child table. We need the ** child table as a SrcList for sqlite3WhereBegin()

78787. SQLITE3EXT_H

78788. ** Remove the page supplied as an argument from the hash table ** (PCache1.apHash structure) that it is currently stored in. ** Also free the page if freePage is true. *** The PGroup mutex must be held when this function is called.

78789. Number of UTF-8 characters in zTabName

78790. ** Try to provide a memory barrier operation, needed for initialization ** and also for the implementation of xShmBarrier in the VFS in cases ** where SQLite is compiled without mutexes.

78791. Number of outstanding file locks

78792. Begin a transaction and increment the schema cookie.

78793. ** The following values may be passed as the second argument to ** sqlite3OsLock(). The various locks exhibit the following semantics: *** ** SHARED: Any number of processes may hold a SHARED lock simultaneously. ** RESERVED: A single process may hold a RESERVED lock on a file at ** any time. Other processes may hold and obtain new SHARED locks. ** PENDING: A single process may hold a PENDING lock on a file at ** any one time. Existing SHARED locks may persist, but no new ** SHARED locks may be obtained by other processes. ** EXCLUSIVE: An EXCLUSIVE lock precludes all other locks. *** ** PENDING_LOCK may not be passed directly to sqlite3OsLock(). Instead, a ** process that requests an EXCLUSIVE lock may actually obtain a PENDING ** lock. This can be upgraded to an EXCLUSIVE lock by a subsequent call to ** sqlite3OsLock().

78794. ** Page pParent is an internal (non-leaf) tree page. This function ** asserts that page number iChild is the left-child if the iIdx'th ** cell in page pParent. Or, if iIdx is equal to the total number of ** cells in pParent, that page number iChild is the right-child of ** the page.

78795. Number of aux. indexes on table zTbl

78796. Iterates through aOvfl[]

78797. Figure out if this structure requires optimization. A structure does ** not require optimization if either: *** ** + it consists of fewer than two segments, or ** + all segments are on the same level, or ** + all segments except one are currently inputs to a merge operation. *** ** In the first case, return NULL. In the second, increment the ref-count ** on *pStruct and return a copy of the pointer to it.

78798. ***** Continuing where we left off in sqliteInt.h *****

78799. CURRENT_DATE

78800. **comment:** ** If SQLITE_MALLOC_SOFT_LIMIT is not zero, then try to keep the ** sizes of memory allocations below this value where possible.

label: code-design

78801. **comment:** ** CAPI3REF: Binding Values To Prepared Statements ** KEYWORDS: {host parameter} {host parameters} {host parameter name} ** KEYWORDS: {SQL parameter} {SQL parameters} {parameter binding} ** METHOD: sqlite3_stmt ** ** ^In the SQL statement text input to [sqlite3_prepare_v2()] and its variants, ** literals may be replaced by a [parameter] that matches one of following ** templates: *** ** ** ? ** ? NNN ** :VVV ** @VVV ** \$VVV ** ** ** In the templates above, NNN represents an integer literal, ** and VVV represents an alphanumeric identifier.)** ^The values of these ** parameters (also called "host parameter names" or "SQL parameters") ** can be set using the sqlite3_bind_*() routines defined here. *** ** ^The first argument to the sqlite3_bind_*() routines is always ** a pointer to the [sqlite3_stmt] object returned from ** [sqlite3_prepare_v2()] or its variants. *** ** ^The second argument is the index of the SQL parameter to be set. ** ^The leftmost SQL parameter has an index of 1. ** ^When the same named ** SQL parameter is used more than once, second and subsequent ** occurrences have the same index as the first occurrence. ** ^The index for named parameters can be looked up using the ** [sqlite3_bind_parameter_index()] API if desired. ^The index ** for "?NNN" parameters is the value of NNN. ** ^The NNN value must be between 1 and the [sqlite3_limit()] ** parameter [SQLITE_LIMIT_VARIABLE_NUMBER] (default value: 999). *** ** ^The third argument is the value to bind to the parameter. ** ^If the third parameter to sqlite3_bind_text() or sqlite3_bind_text16() ** or sqlite3_bind_blob() is a NULL pointer then the fourth parameter ** is ignored and the end result is the same as sqlite3_bind_null(). *** ** ^In those routines that have a fourth argument, its value is the ** number of bytes in the parameter. To be clear: the value is the ** number of <u>bytes</u> in the value, not the number of characters.)** ^If the fourth parameter to sqlite3_bind_text() or sqlite3_bind_text16() ** is negative, then the length of the string is ** the number of bytes up to the first zero terminator. ** If the fourth parameter to sqlite3_bind_blob() is negative, then ** the behavior is undefined. ** If a non-negative fourth parameter is provided to sqlite3_bind_text() ** or sqlite3_bind_text16() or sqlite3_bind_text64() then ** that parameter must be the byte offset ** where the NUL terminator would occur assuming the string were NUL ** terminated. If any NUL characters occur at byte offsets less than ** the value of the fourth parameter then the resulting string value will ** contain embedded NULs. The result of expressions involving strings ** with embedded NULs is undefined. *** ** ^The fifth argument to the BLOB and string binding interfaces ** is a destructor used to dispose of the BLOB or ** string after SQLite has finished with it. ^The destructor is called ** to dispose of the BLOB or string even if the call to bind API fails. ** ^If the fifth argument is ** the special value [SQLITE_STATIC], then SQLite assumes that the ** information is in static, unmanaged space and does not need to be freed. ** ^If the fifth argument has the value [SQLITE_TRANSIENT], then ** SQLite makes its own private copy of the data immediately, before ** the sqlite3_bind_*() routine returns. *** ** ^The sixth argument to sqlite3_bind_text64() must be one of ** [SQLITE_UTF8], [SQLITE_UTF16], [SQLITE_UTF16BE], or [SQLITE_UTF16LE] ** to specify the encoding of the text in the third parameter. If ** the sixth argument to sqlite3_bind_text64() is not one of the ** allowed values shown above, or if the text encoding is different ** from the encoding specified by the sixth parameter, then the behavior ** is undefined. *** ** ^The sqlite3_bind_zeroblob() routine binds a BLOB of length N that ** is filled with zeroes. ^A zeroblob uses a fixed amount of memory ** (just an integer to hold its size) while it is being processed. ** Zeroblobs are intended to serve as placeholders for BLOBs whose ** content is later written using ** [sqlite3_blob_open | incremental BLOB I/O] routines. ** ^A negative value for the zeroblob results in a zero-length BLOB. *** ** ^The sqlite3_bind_pointer(S,I,P,T,D) routine causes the I-th parameter in ** [prepared statement] S to have an SQL value of NULL, but to also be ** associated with the pointer P of type T. ^D is either a NULL pointer or ** a pointer to a destructor function for P. ^SQLite will invoke the ** destructor D with a single argument of P when it is finished using ** P. The T parameter should be a static string, preferably a string ** literal. The sqlite3_bind_pointer() routine is part of the ** [pointer passing interface] added for SQLite 3.20.0. *** ** ^If any of the sqlite3_bind_*() routines are called with a NULL pointer ** for the [prepared statement] or with a prepared statement for which ** [sqlite3_step()] has been called more recently than [sqlite3_reset()], ** then the call will return [SQLITE_MISUSE]. If any sqlite3_bind_*() ** routine is passed a [prepared statement] that has been finalized, the ** result is undefined and probably harmful. ** ** ^Bindings are not cleared by the [sqlite3_reset()] routine. ** ^Unbound parameters are interpreted as NULL. *** ** ^The sqlite3_bind_* routines return [SQLITE_OK] on success or an ** [error code] if anything goes wrong. ** ^[SQLITE_TOOBIG] might be returned if the size of a string or BLOB ** exceeds limits imposed by [sqlite3_limit()][SQLITE_LIMIT_LENGTH] or ** [SQLITE_MAX_LENGTH]. ** ^[SQLITE_RANGE] is returned if the parameter ** index is out of range. ^[SQLITE_NOMEM] is returned if malloc() fails. *** ** See also: [sqlite3_bind_parameter_count()], ** [sqlite3_bind_parameter_name()], and [sqlite3_bind_parameter_index()].

label: code-design

78802. The journal file from which to read

78803. Only attempt the conversion to TEXT if there is an integer or real ** representation (blob and NULL do not get converted) but no string ** representation. It would be harmless to repeat the conversion if ** there is already a string rep, but it is pointless to waste those ** CPU cycles.

78804. Filename NULL

78805. Range contraints that come from the LIKE optimization are ** always used in pairs.

78806. Otherwise see if some other process holds it.

78807. End of buffer

78808. refarg ::= MATCH nm

78809. Table Name Column Name

78810. ERROR. return a NULL

78811. Main structure

78812. Phrase and token index, respectively

78813. 5

78814. The next state

78815. ** The following routine only works on pentium-class (or newer) processors. ** It uses the RDTSC opcode to read the cycle count value out of the ** processor and returns that value. This can be used for high-res ** profiling.

78816. Table to index. Use pParse->pNewTable if 0

78817. Generate loop termination code.

78818. Used to iterate through indexes

78819. Size of hash table Rtree.aHash. This hash table is not expected to ** ever contain very many entries, so a fixed number of buckets is ** used.

78820. The x<=z term

78821. ** json_group_array(VALUE) ** ** Return a JSON array composed of all values in the aggregate.

78822. The main static mutex

78823. TABLE
78824. ccons ::= CHECK LP expr RP
78825. Number of entries in aRoot[]
78826. Space to hold the labels
78827. If nTruncate is non-zero, this is a commit record.
78828. 0x00. Next integer will be a docid.
78829. ** A pointer to this structure is used to communicate information ** from sqlite3Init and OP_ParseSchema into the sqlite3InitCallback.
78830. same as TK_REM, synopsis: r[P3]=r[P2]%-r[P1]
78831. 1 for UPDATE, 0 for DELETE
78832. Index.aSortOrder
78833. True if there exists a hot journal-file
78834. op = IN, EXISTS, SELECT, CASE, FUNCTION, BETWEEN
78835. xLanguageid
78836. 950
78837. ** 2001 September 15 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains routines used for analyzing expressions and ** for generating VDBE code that evaluates expressions in SQLite.
78838. True if ORDER BY rank
78839. Remember the size of the LHS in iTable so that we can check that ** the RHS and LHS sizes match during code generation.
78840. Create the DELETE statement to write to the target PK b-tree. ** Because it only performs INSERT operations, this is not required for ** an rbu vacuum handle.
78841. BUSY not possible when useWal==1
78842. Argument to pass to xBusyHandler
78843. VFS only
78844. 0x13
78845. It's odd to simulate an io-error here, but really this is just ** using the io-error infrastructure to test that SQLite handles this ** function failing. This function could fail if, for example, the ** current working directory has been unlinked.
78846. ** Argument pCsr must be a cursor opened for writing on an ** INTKEY table currently pointing at a valid table entry. ** This function modifies the data stored as part of that entry. ** Only the data content may only be modified, it is not possible to ** change the length of the data stored. If this function is called with ** parameters that attempt to write past the end of the existing data, ** no modifications are made and SQLITE_CORRUPT is returned.
78847. ROW
78848. **comment:** ** Remove an entry from the %_segdir table. This involves running the ** following two statements: ** ** DELETE FROM %_segdir WHERE level = :iAbsLevel AND idx = :iIdx ** UPDATE %_segdir SET idx = idx - 1 WHERE level = :iAbsLevel AND idx > :iIdx ** ** The DELETE statement removes the specific %_segdir level. The UPDATE ** statement ensures that the remaining segments have contiguously allocated ** idx values.
label: code-design
78849. ** Return true if the given BtCursor is valid. A valid cursor is one ** that is currently pointing to a row in a (non-empty) table. ** This is a verification routine is used only within assert() statements.
78850. ** The pre-update hook implementations.
78851. Register holding an index record
78852. nByte must be a positive
78853. **comment:** ** A macro to hint to the compiler that a function should not be ** inlined.
label: documentation
78854. Retrieve the current mutex implementation
78855. Call SetAutoVacuum() to set initialize the internal auto and ** incr-vacuum flags. This is required in case this connection ** creates the database file. It is important that it is created ** as an auto-vacuum capable db.
78856. ** Compile the UTF-8 encoded SQL statement zSql into a statement handle.
78857. Number of full pages read from DB
78858. ** Analyze the pExpr expression looking for aggregate functions and ** for variables that need to be added to AggInfo object that pNC->pAggInfo ** points to. Additional entries are made on the AggInfo object as ** necessary. ** ** This routine should only be called after the expression has been ** analyzed by sqlite3ResolveExprNames().
78859. Write results here
78860. How to write to Queue
78861. Schema used for this query
78862. ***** End of ft3_unicode.c *****
78863. WhereTerms used
78864. (297) conslist ::= conslist tconscomma tcons
78865. IN/OUT: Position list
78866. Size of doclist in bytes
78867. Allocate the new vtab object and parse the configuration
78868. ** Implementation of the xBestIndex method for FTS3 tables. There ** are three possible strategies, in order of preference: ** ** 1. Direct lookup by rowid or docid. ** 2. Full-text search using a MATCH operator on a non-docid column. ** 3. Linear scan of %_content table.
78869. If this was an INSERT, UPDATE or DELETE and no statement transaction ** has been rolled back, update the database connection change-counter.
78870. Copy of the SELECT that implements the view
78871. virtual table
78872. Error code from journal finalization operation
78873. Tokenizer implementations will typically add additional fields
78874. ** Add a new CHECK constraint to the table currently under construction.
78875. Position of token in phrase
78876. Constraint type
78877. ** This function is the implementation of both the xConnect and xCreate ** methods of the FTS3 virtual table. ** ** The argv[] array contains the following: ** ** argv[0] -> module name ("fts3" or "fts4") ** argv[1] -> database name ** argv[2] -> table name ** argv[...] -> "column name" and other module argument fields.
78878. Incrementally read one PMA
78879. sqlite3ParserTrace(stdout, "parser: ");
78880. ** 2012 May 24 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Implementation of the "unicode" full-text-search tokenizer.
78881. Input character index
78882. If the left hand side of the regexp operator is NULL, ** then the result is also NULL.
78883. PRAGMA synchronous=EXTRA
78884. FROM is not a subquery or view
78885. ** Recursively apply colset pColset to expression node pNode and all of ** its descendants. If (*ppFree) is not NULL, it contains a spare copy ** of pColset. This function may use the spare copy and set (*ppFree) to ** zero, or it may create copies of pColset using fts5CloneColset().
78886. Escape ' and enclose in '...'
78887. The entire poslist will fit on the current leaf. So copy ** it in one go.
78888. Opcode: SorterData P1 P2 P3 * * * Synopsis: r[P2]=data ** ** Write into register P2 the current sorter data for sorter cursor P1. ** Then clear the column header cache on cursor P3. ** ** This opcode is normally use to move a record out of the sorter and into ** a register that is the source for a pseudo-table cursor created

using ** OpenPseudo. That pseudo-table cursor is the one that is identified by ** parameter P3. Clearing the P3 column cache as part of this opcode saves ** us from having to issue a separate NullRow instruction to clear that cache.

78889. Array indicating modified columns

78890. Index of term>=? value in apVal

78891. someone else might have it reserved

78892. Different sort orders

78893. ** This is the xExprCallback for a tree walker. It is used to ** implement sqlite3ExprAnalyzeAggregates(). See sqlite3ExprAnalyzeAggregates ** for additional information.

78894. Bulk memory used by pcache-local

78895. A flag to indicate when processing is finished

78896. **comment:** ** All of the static variables used by this module are collected ** into a single structure named "mem3". This is to keep the ** static variables organized and to reduce namespace pollution ** when this module is combined with other in the amalgamation.

label: code-design

78897. Jump here to skip the current record

78898. Append position iPos to the output

78899. ***** End of os_win.h *****

78900. assert(!db->mallocFailed); // not true with SQLITE_USE_ALLOCA

78901. The word being stemmed (Reversed)

78902. term ::= CTIME_KW

78903. IMP: R-61949-35727

78904. True if the node needs to be written to disk

78905. Originates in ON/USING clause of outer join

78906. Initial static space for a[]

78907. Start label for the block-output subroutine

78908. **comment:** The recursive SELECT to be coded

label: code-design

78909. xClose method

78910. Check if an existing function is being overridden or deleted. If so, ** and there are active VMs, then return SQLITE_BUSY. If a function ** is being overridden/deleted but there are no active VMs, allow the ** operation to continue but invalidate all precompiled statements.

78911. Insert on an aux. index b-tree

78912. Use the current thread to populate aFile[1], even though this ** PmaReader is multi-threaded. If this is an INCRINIT_TASK object, ** then this function is already running in background thread ** pIncr->pTask->thread. *** If this is the INCRINIT_ROOT object, then it is running in the ** main Vdbe thread. But that is Ok, as that thread cannot return ** control to the Vdbe or proceed with anything useful until the ** first results are ready from this merger object anyway.

78913. Number of bytes of available data

78914. ** CAPI3REF: Number of columns in a result set ** METHOD: sqlite3_stmt ** ** ^The sqlite3_data_count(P) interface returns the number of columns in the ** current row of the result set of [prepared statement] P. ** ^If prepared statement P does not have results ready to return ** (via calls to the [sqlite3_column_int | sqlite3_column_*()]) of ** interfaces) then sqlite3_data_count(P) returns 0. ** ^The sqlite3_data_count(P) routine also returns 0 if P is a NULL pointer. ** ^The sqlite3_data_count(P) routine returns 0 if the previous call to ** [sqlite3_step](P) returned [SQLITE_DONE]. ^The sqlite3_data_count(P) ** will return non-zero if previous call to [sqlite3_step](P) returned ** [SQLITE_ROW], except in the case of the [PRAGMA incremental_vacuum] ** where it always returns zero since each step of that multi-step ** pragma returns 0 columns of data. *** See also: [sqlite3_column_count()]

78915. First Mem of result set

78916. 33

78917. True if inside sqlite3_declare_vtab()

78918. _SQLITE_OS_H

78919. ** Open a shared-memory area associated with open database file pDbFd. ** This particular implementation uses mmapped files. *** The file used to implement shared-memory is in the same directory ** as the open database file and has the same name as the open database ** file with the "-shm" suffix added. For example, if the database file ** is "/home/user1/config.db" then the file that is created and mmapped ** for shared memory will be called "/home/user1/config.db-shm". ** ** Another approach to is to use files in /dev/shm or /dev/tmp or an ** some other tmpfs mount. But if a file in a different directory ** from the database file is used, then differing access permissions ** or a chroot() might cause two different processes on the same ** database to end up using different files for shared memory - ** meaning that their memory would not really be shared - resulting ** in database corruption. Nevertheless, this tmpfs file usage ** can be enabled at compile-time using -DSQLITE_SHM_DIRECTORY="/dev/shm" ** or the equivalent. The use of the SQLITE_SHM_DIRECTORY compile-time ** option results in an incompatible build of SQLite; builds of SQLite ** that with differing SQLITE_SHM_DIRECTORY settings attempt to use the ** same database file at the same time, database corruption will likely ** result. The SQLITE_SHM_DIRECTORY compile-time option is considered ** "unsupported" and may go away in a future SQLite release. *** When opening a new shared-memory file, if no other instances of that ** file are currently open, in this process or in other processes, then ** the file must be truncated to zero length or have its header cleared. *** If the original database file (pDbFd) is using the "unix-excl" VFS ** that means that an exclusive lock is held on the database file and ** that no other processes are able to read or write the database. In ** that case, we do not really need shared memory. No shared memory ** file is created. The shared memory will be simulated with heap memory.

78920. Set to true if out of memory

78921. .Malloc =

78922. ** Open the sqlite_master table stored in database number iDb for ** writing. The table is opened using cursor 0.

78923. "write-version" value for main db files

78924. Bonus points if the text encoding matches

78925. This is a delete. Set the delete flag.

78926. Table that the trigger fires off of

78927. Initialize the Vdbe program

78928. No foreign keys against expression indexes

78929. The expression to extract a value from

78930. Size of extra space for each page

78931. Flush all currently outstanding nodes to disk.

78932. For use by application

78933. Steps 3, 4, and 5 implemented by this subroutine

78934. Number of bytes to lock or unlock

78935. Write new cell into this node

78936. Test all UNIQUE constraints by creating entries for each UNIQUE ** index and making sure that duplicate entries do not already exist. ** Compute the revised record entries for indices as we go. *** This loop also handles the case of the PRIMARY KEY index for a ** WITHOUT ROWID table.

78937. Constant "b" from BM25 formula

78938. Size of each object in the array

78939. Forward declaration of methods

78940. Address of a freeblock within pPage->aData[]

78941. **comment:** ** Set up SQL objects in database db used to access the contents of ** the hash table pointed to by argument pHash. The hash table must ** been initialized to use string keys, and to take a private copy ** of the key when a value is inserted. i.e. by a call similar to: ** ** sqlite3Fts3HashInit(pHash, FTS3_HASH_STRING, 1); ** ** This function adds a scalar function (see header comment above ** fts3TokenizerFunc() in this file for details) and, if ENABLE_TABLE is ** defined at compilation time, a temporary virtual table (see header ** comment above struct HashTableVtab) to the database schema. Both ** provide read/write access to the contents of *pHash. *** The third argument to this function, zName, is used as the name ** of both the scalar and, if created, the virtual table.

label: code-design

78942. If a new wal-index header was loaded before the checkpoint was ** performed, then the pager-cache associated with pWal is now ** out of date. So zero the cached wal-index header to ensure that ** next time the pager opens a snapshot on this database it knows that ** the cache needs to be reset.

78943. The new cursor
78944. ***** End of icu.c *****
78945. Page-type flag for the root page of new table
78946. **comment:** TODO(shess) Only used for SQLITE_OK and SQLITE_DONE at this time. ** If tokenizers are to be allowed to call sqlite3_*() functions, then ** we will need a way to register the API consistently.
label: code-design
78947. For the purposes of the EP_ConstFunc flag, date and time ** functions and other functions that change slowly are considered ** constant because they are constant for the duration of one query
78948. ** Add an opcode that includes the p4 value as a pointer.
78949. ** IN/OUT parameter (*pa) points to a position list n bytes in size. If ** the position list contains entries for column iCol, then (*pa) is set ** to point to the sub-position-list for that column and the number of ** bytes in it returned. Or, if the argument position list does not ** contain any entries for column iCol, return 0.
78950. Pointer to %_segdir.root buffer
78951. Check if a thread in this process holds such a lock
78952. ** Change the pMem->zMalloc allocation to be at least szNew bytes. ** If pMem->zMalloc already meets or exceeds the requested size, this ** routine is a no-op.
 *** Any prior string or blob content in the pMem object may be discarded. ** The pMem->xDel destructor is called, if it exists. Though MEM_Str ** and MEM_Blob values may be discarded, MEM_Int, MEM_Real, and MEM_Null ** values are preserved. *** Return SQLITE_OK on success or an error code (probably SQLITE_NOMEM) ** if unable to complete the resizing.
78953. **comment:** TODO: SQLite does something special to deal with mixed-endian ** floating point values (e.g. ARM7). This code probably should ** too.
label: code-design
78954. Malformed header - probably all zeros
78955. ***** End of callback.c *****
78956. ** 2006 September 30 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ***
***** Implementation of the full-text-search tokenizer that implements **
a Porter stemmer.
78957. ** Free a MatchinfoBuffer object allocated using fts3MIBufferNew()
78958. OUT: Segment id
78959. Btree handle holding this lock
78960. Buffer in which to assemble pgidx
78961. ** The following functions are used to register the module with SQLite. If ** this module is being built as part of the SQLite core (SQLITE_CORE is ** defined), then sqlite3_open() will call sqlite3Fts5Init() directly. ** ** Or, if this module is being built as a loadable extension, ** sqlite3Fts5Init() is omitted and the two standard entry points ** sqlite3_fts_init() and sqlite3_fts5_init() defined instead.
78962. sqlite3_pcache_methods2*
78963. ** Return the size of the database in pages (or zero, if unknown).
78964. Number of columns in table
78965. OUT: Normalized text for token
78966. Preserve the text encoding
78967. True if copy of key made on insert
78968. The file descriptor for the journal file
78969. ** The implementation of the rbu_target_name() SQL function. This function ** accepts one or two arguments. The first argument is the name of a table - ** the name of a table in the RBU database. The second, if it is present, is 1 ** for a view or 0 for a table. *** For a non-vacuum RBU handle, if the table name matches the pattern: ** ** data[0-9]_<name> ** ** where <name> is any sequence of 1 or more characters, <name> is returned. ** Otherwise, if the only argument does not match the above pattern, an SQL ** NULL is returned. *** "data_t1" -> "t1" ** "data0123_t2" -> "t2" ** "dataAB_t3" -> NULL *** For an rbu vacuum handle, a copy of the first argument is returned if ** the second argument is either missing or 0 (not a view).
78970. ** Return the size of the header added to each page by this module.
78971. Fx
78972. Aggregate function with DISTINCT keyword
78973. Opcode: VFilter P1 P2 P3 P4 * ** Synopsis: iplan=r[P3] zplan='P4' ** ** P1 is a cursor opened using VOpen. P2 is an address to jump to if ** the filtered result set is empty. ** ** P4 is either NULL or a string that was generated by the xBestIndex ** method of the module. The interpretation of the P4 string is left ** to the module implementation. ** ** This opcode invokes the xFilter method on the virtual table specified ** by P1. The integer query plan parameter to xFilter is stored in register ** P3. Register P3+1 stores the argc parameter to be passed to the ** xFilter method. Registers P3+2..P3+1+argc are the argc ** additional parameters which are passed to ** xFilter as argv. Register P3+2 becomes argv[0] when passed to xFilter. *** A jump is made to P2 if the result set after filtering would be empty.
78974. ** The maximum legal page number is (2^31 - 1).
78975. Number of bytes of the blob to include in output
78976. CHECK
78977. Reload the table, index and permanent trigger schemas.
78978. IMP: R-24470-31136 This function is an SQL wrapper around the ** sqlite3_sourceid() C interface.
78979. Database is always well-formed
78980. ** Prepare to begin tokenizing a particular string. The input ** string to be tokenized is pInput[0..nBytes-1]. A cursor ** used to incrementally tokenize this string is returned in ** *ppCursor.
78981. Number of iterators in alter not at EOF
78982. ** Given an allocation, find the MemBlockHdr for that allocation. *** This routine checks the guards at either end of the allocation and ** if they are incorrect it asserts.
78983. ** Try to convert a value into a numeric representation if we can ** do so without loss of information. In other words, if the string ** looks like a number, convert it into a number. If it does not ** look like a number, leave it alone. *** If the bTryForInt flag is true, then extra effort is made to give ** an integer representation. Strings that look like floating point ** values but which have no fractional component (example: '48.00') ** will have a MEM_Int representation when bTryForInt is true. *** If bTryForInt is false, then if the input string contains a decimal ** point or exponential notation, the result is only MEM_Real, even ** if there is an exact integer representation of the quantity.
78984. ***** Begin file sqlite3rtree.h *****
78985. Token that describes the complete CREATE TRIGGER
78986. aOrder array
78987. Trace this connection
78988. Make sure zData points to enough of the record to cover the header.
78989. At this point pNew->nOut is set to the number of rows expected to ** be visited by the index scan before considering term pTerm, or the ** values of nIn and nInMul. In other words, assuming that all ** "x IN(...)" terms are replaced with "x = ?". This block updates ** the value of pNew->nOut to account for pTerm (but not nIn/nInMul).
78990. 19-24
78991. Number of slots allocated for aLTerm[]
78992. **comment:** ** Characters that may appear in the second argument to matchinfo().
label: code-design
78993. Extra content
78994. Name of the database containing table, or NULL
78995. EOF
78996. Precision of the current field
78997. ** Populate buffer zOut with the full canonical pathname corresponding ** to the pathname in zPath. zOut is guaranteed to point to a buffer ** of at least (DEVSYM_MAX_PATHNAME+1) bytes.
78998. 460

78999. ** Set the SQLITE_PTRSIZE macro to the number of bytes in a pointer
79000. ** The sqlite3_strlike() interface. Return 0 on a match and non-zero for ** a miss - like strcmp().
79001. Unless an error has occurred, register numbers are always positive.
79002. OUT: New cursor object
79003. largest root page number
79004. ** next_row: ** regChng = 0 ** if(idx(0) != regPrev(0)) goto chng_addr_0 ** regChng = 1 ** if(idx(1) != regPrev(1)) goto chng_addr_1 ** ... ** regChng = N
 ** goto endDistinctTest
79005. Number of labels used
79006. An implied index of the table
79007. xOpen
79008. Loop counter 0..nSnippet-1
79009. Write the payload into the local Cell and any extra into overflow pages
79010. ** Normally, a PmaReader object iterates through an existing PMA stored ** within a temp file. However, if the PmaReader.pIncr variable points to ** an object of the following type, it may be used to iterate/merge through ** multiple PMAs simultaneously. ** ** There are two types of IncrMerger object - single (bUseThread==0) and ** multi-threaded (bUseThread==1). ** ** A multi-threaded IncrMerger object uses two temporary files - aFile[0] ** and aFile[1]. Neither file is allowed to grow to more than mxSz bytes in ** size. When the IncrMerger is initialized, it reads enough data from ** pMerger to populate aFile[0]. It then sets variables within the ** corresponding PmaReader object to read from that file and kicks off ** a background thread to populate aFile[1] with the next mxSz bytes of ** sorted record data from pMerger. ** ** When the PmaReader reaches the end of aFile[0], it blocks until the ** background thread has finished populating aFile[1]. It then exchanges ** the contents of the aFile[0] and aFile[1] variables within this structure, ** sets the PmaReader fields to read from the new aFile[0] and kicks off ** another background thread to populate the new aFile[1]. And so on, until ** the contents of pMerger are exhausted. ** ** A single-threaded IncrMerger does not open any temporary files of its ** own. Instead, it has exclusive access to mxSz bytes of space beginning ** at offset iStartOff of file pTask->file2. And instead of using a ** background thread to prepare data for the PmaReader, with a single ** threaded IncrMerger the allocate part of pTask->file2 is "refilled" with ** keys from pMerger by the calling thread whenever the PmaReader runs out ** of data.
79011. ** Open a blob handle.
79012. Mask of OLD.* columns in use
79013. Magic number for sanity checking
79014. Make an entry in the sqlite_master table
79015. Opcode: SorterNext P1 P2 * * P5 ** ** This opcode works just like OP_Next except that P1 must be a ** sorter object for which the OP_SorterSort opcode has been ** invoked. This opcode advances the cursor to the next sorted ** record, or jumps to P2 if there are no more sorted records.
79016. ***** End of parse.c *****
79017. **comment:** ** The following array holds FuncDef structures for all of the functions ** defined in this file. ** ** The array cannot be constant since changes are made to the ** FuncDef.pHash elements at start-time. The elements of this array ** are read-only after initialization is complete. ** ** For peak efficiency, put the most frequently used function last.
label: code-design
79018. Absolute level to delete from
79019. Comparison affinity
79020. expr ::= MINUS expr
79021. Ignore wal frames before this one
79022. ** Fill the Index.aiRowEst[] array with default information - information ** to be used when we have not run the ANALYZE command. ** ** aiRowEst[0] is supposed to contain the number of elements in the index. ** Since we do not know, guess 1 million. aiRowEst[1] is an estimate of the ** number of rows in the table that match any particular value of the ** first column of the index. aiRowEst[2] is an estimate of the number ** of rows that match any particular combination of the first 2 columns ** of the index. And so forth. It must always be the case that * ** aiRowEst[N]<=aiRowEst[N-1] ** aiRowEst[N]>=1 ** ** Apart from that, we have little to go on besides intuition as to ** how aiRowEst[] should be initialized. The numbers generated here ** are based on typical values found in actual indices.
79023. ** 2008 August 18 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil.
 ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains routines used for walking the parser tree and ** resolve all identifiers by associating them with a particular ** table and column.
79024. Number of bytes in text at z
79025. synopsis: if r[P3]=null halt
79026. Right subnode
79027. If there are other active statements that belong to this database ** handle, downgrade to a read-only transaction. The other statements ** may still be reading from the database.
79028. ** This function is called on every node of an expression tree used as an ** argument to the OP_CursorHint instruction. If the node is a TK_COLUMN ** that accesses any table other than the one identified by ** CCurHint.iTabCur, then do the following: ** ** 1) allocate a register and code an OP_Column instruction to read ** the specified column into the new register, and ** ** 2) transform the expression node to a TK_REGISTER node that reads ** from the newly populated register. ** ** Also, if the node is a TK_COLUMN that does access the table identified ** by pCCurHint.iTabCur, and an index is being used (which we will ** know because CCurHint.pIdx!=0) then transform the TK_COLUMN into ** an access of the index rather than the original table.
79029. ifexists :=
79030. ** Walk the parse trees associated with all subqueries in the ** FROM clause of SELECT statement p. Do not invoke the select ** callback on p, but do invoke it on each FROM clause subquery ** and on any subqueries further down in the tree. Return ** WRC_Abort or WRC_Continue;
79031. The new rowid is not NULL (in this case the rowid will be ** automatically assigned and there is no chance of a conflict), and ** the statement is either an INSERT or an UPDATE that modifies the ** rowid column. So if the conflict mode is REPLACE, then delete any ** existing row with rowid=pNewRowid. ** ** Or, if the conflict mode is not REPLACE, insert the new record into ** the %_content table. If we hit the duplicate rowid constraint (or any ** other error) while doing so, return immediately. ** ** This branch may also run if pNewRowid contains a value that cannot ** be losslessly converted to an integer. In this case, the eventual ** call to fts3InsertData() (either just below or further on in this ** function) will return SQLITE_MISMATCH. If fts3DeleteByRowid is ** invoked, it will delete zero rows (since no row will have ** docid=\$pNewRowid if \$pNewRowid is not an integer value).
79032. Value of k is out of range. Database corruption
79033. downgrading to a shared lock on NFS involves clearing the write lock ** before establishing the readlock - to avoid a race condition we downgrade ** the lock in 2 blocks, so that part of the range will be covered by a ** write lock until the rest is covered by a read lock: ** 1: [WWWWW] ** 2: [...W] ** 3: [RRRRW] ** 4: [RRR.]
79034. ** Return the index of the cell containing a pointer to node pNode ** in its parent. If pNode is the root node, return -1.
79035. ** Unlink the given table from the hash tables and the delete the ** table structure with all its indices and foreign keys.
79036. Current parse context
79037. end loop over pSrcList
79038. **comment:** ** Some malloc failures are only possible if SQLITE_TEST_REALLOC_STRESS is ** defined. We need to defend against those failures when testing with ** SQLITE_TEST_REALLOC_STRESS, but we don't want the unreachable branches ** during a normal build. The following macro can be used to disable tests ** that are always false except when SQLITE_TEST_REALLOC_STRESS is set.
label: test
79039. Score of best snippet
79040. Both tables must have the same INTEGER PRIMARY KEY
79041. This block ensures that all data required to recreate the original ** database has been stored in the journal for pDestPager and the ** journal synced to disk. So at this point we may safely modify ** the database file in any way, knowing that if a power failure ** occurs, the original database will be reconstructed from the ** journal file.
79042. Memory allocation function
79043. ** Invalidate the current position list for phrase pPhrase.
79044. Create a new change object containing all the old values (if ** this is an SQLITE_UPDATE or SQLITE_DELETE), or just the PK ** values (if this is an INSERT).

79045. Matching left table
79046. ** Generate code for the start of the iLevel-th loop in the WHERE clause ** implementation described by pWInfo.
79047. 199
79048. True if currently busy
79049. Document store
79050. Parse context (for malloc() and error reporting)
79051. Mark the table as Ephemeral prior to deleting it, so that the ** sqlite3DeleteTable() routine will know that it is not stored in ** the schema.
79052. **comment:** ** Make copies of relevant WHERE clause terms of the outer query into ** the WHERE clause of subquery. Example: ** * SELECT * FROM (SELECT a AS x, c-d AS y FROM t1) WHERE x=5 AND y=10; ** ** Transformed into: ** * SELECT * FROM (SELECT a AS x, c-d AS y FROM t1 WHERE a=5 AND c-d=10) ** WHERE x=5 AND y=10; ** ** The hope is that the terms added to the inner query will make it more ** efficient. ** ** Do not attempt this optimization if: ** ** (1) The inner query is an aggregate. (In that case, we'd really want ** to copy the outer WHERE-clause terms onto the HAVING clause of the ** inner query. But they probably won't help there so do not bother.) ** ** (2) The inner query is the recursive part of a common table expression. ** ** (3) The inner query has a LIMIT clause (since the changes to the WHERE ** close would change the meaning of the LIMIT). ** ** (4) The inner query is the right operand of a LEFT JOIN. (The caller ** enforces this restriction since this routine does not have enough ** information to know.) ** ** (5) The WHERE clause expression originates in the ON or USING clause ** of a LEFT JOIN. ** ** Return 0 if no changes are made and non-zero if one or more WHERE clause ** terms are duplicated into the subquery.
label: code-design
79053. If this statement was prepared using saved SQL and an ** error has occurred, then return the error code in p->rc to the ** caller. Set the error code in the database handle to the same value.
79054. Estimated number of rows returned
79055. **comment:** ** CAPI3REF: Destroy A Prepared Statement Object ** DESTRUCTOR: sqlite3_stmt ** ** ^The sqlite3_finalize() function is called to delete a [prepared statement]. ** ^If the most recent evaluation of the statement encountered no errors ** or if the statement is never been evaluated, then sqlite3_finalize() returns ** SQLITE_OK. ^If the most recent evaluation of statement S failed, then ** sqlite3_finalize(S) returns the appropriate [error code] or ** [extended error code]. ** ** ^The sqlite3_finalize(S) routine can be called at any point during ** the life cycle of [prepared statement] S: ** before statement S is ever evaluated, after ** one or more calls to [sqlite3_reset()], or after any call ** to [sqlite3_step()] regardless of whether or not the statement has ** completed execution. ** ** ^Invoking sqlite3_finalize() on a NULL pointer is a harmless no-op. ** ** The application must finalize every [prepared statement] in order to avoid ** resource leaks. It is a grievous error for the application to try to use ** a prepared statement after it has been finalized. Any use of a prepared ** statement after it has been finalized can result in undefined and ** undesirable behavior such as segfaults and heap corruption.
label: code-design
79056. The database to store the trigger in
79057. ** Buffer (a/n) is assumed to contain a list of serialized varints. Read ** each varint and append its string representation to buffer pBuf. Return ** after either the input buffer is exhausted or a 0 value is read. ** ** The return value is the number of bytes read from the input buffer.
79058. Checksum value to return
79059. COMMA => nothing
79060. **comment:** Omit unused tables in joins
label: code-design
79061. ** Unregister a VFS so that it is no longer accessible.
79062. The connection to be opened
79063. Function to reinitialize pages
79064. Index within level, or 0x7FFFFFFF for PT
79065. True if wrong number of arguments
79066. ** Continue the search on cursor pCur until the front of the queue ** contains an entry suitable for returning as a result-set row, ** or until the RtreeSearchPoint queue is empty, indicating that the ** query has completed.
79067. X costs more than Y
79068. Implemented as a co-routine
79069. ** Flags that make up the mask passed to sqlite3PagerGet().
79070. 272
79071. ** This is a Walker.xSelectCallback callback for the sqlite3SelectTypeInfo() ** interface. ** ** For each FROM-clause subquery, add Column.zType and Column.zColl ** information to the Table structure that represents the result set ** of that subquery. ** ** The Table structure that represents the result set was constructed ** by selectExpander() but the type and collation information was omitted ** at that point because identifiers had not yet been resolved. This ** routine is called after identifier resolution.
79072. Opcode: SeekGE P1 P2 P3 P4 * * * Synopsis: key=r[P3@P4] ** ** If cursor P1 refers to an SQL table (B-Tree that uses integer keys), ** use the value in register P3 as the key. If cursor P1 refers ** to an SQL index, then P3 is the first in an array of P4 registers ** that are used as an unpacked index key. ** ** Reposition cursor P1 so that it points to the smallest entry that ** is greater than or equal to the key value. If there are no records ** greater than or equal to the key and P2 is not zero, then jump to P2. ** ** If the cursor P1 was opened using the OPFLAG_SEEKEQ flag, then this ** opcode will always land on a record that equally equals the key, or ** else jump immediately to P2. When the cursor is OPFLAG_SEEKEQ, this ** opcode must be followed by an IdxLE opcode with the same arguments. ** The IdxLE opcode will be skipped if this opcode succeeds, but the ** IdxLE opcode will be used on subsequent loop iterations. ** ** This opcode leaves the cursor configured to move in forward order, ** from the beginning toward the end. In other words, the cursor is ** configured to use Next, not Prev. ** ** See also: Found, NotFound, SeekLt, SeekGt, SeekLe
79073. Opcode: DropTrigger P1 * * P4 * * * Remove the internal (in-memory) data structures that describe ** the trigger named P4 in database P1. This is called after a trigger ** is dropped from disk (using the Destroy opcode) in order to keep ** the internal representation of the ** schema consistent with what is on disk.
79074. File name
79075. xFullPathname
79076. ** A trigger is either a BEFORE or an AFTER trigger. The following constants ** determine which. ** ** If there are multiple triggers, you might of some BEFORE and some AFTER. ** In that cases, the constants below can be ORed together.
79077. **comment:** The term of the WHERE clause to be coded
label: code-design
79078. Second argument to matchinfo() function
79079. 740
79080. ** Tokenize some text using the ascii tokenizer.
79081. Make sure all the indices are constructed correctly.
79082. JOIN
79083. 390
79084. This routine constructs a binary expression node out of two ExprSpan ** objects and uses the result to populate a new ExprSpan object.
79085. ** Return TRUE if the cursor is not pointing at an entry of the table. ** ** TRUE will be returned after a call to sqlite3BtreeNext() moves ** past the last entry in the table or sqlite3BtreePrev() moves past ** the first entry. TRUE is also returned if the table is empty.
79086. Append the non-PK part of the WHERE clause
79087. Address where integer counter is initialized
79088. ** Make sure the given Mem is \u0000 terminated.
79089. ** Check to see if the CreateFileMappingA function is supported on the ** target system. It is unavailable when using "mincore.lib" on Win10. ** When compiling for Windows 10, always assume "mincore.lib" is in use.
79090. For an INSERT, memory cell p->iNewReg contains the serialized record ** that is being inserted. Deserialize it.
79091. **comment:** A base allocation. Not from malloc.
label: code-design
79092. NDEBUG
79093. If this is a new term, query for it. Update cksum3 with the results.
79094. Previous row content
79095. insert_cmd ::= INSERT orconf

79096. Bytes of new data
79097. Document text to extract snippet from
79098. The table to rename.
79099. ** Advance the iterator to the next coalesced phrase instance. Return ** an SQLite error code if an error occurs, or SQLITE_OK otherwise.
79100. Handle for sqlite3DbMallocRawNN()
79101. One of the CURTYPE_* values above
79102. Index.aiRowLogEst
79103. Next input in pMerge
79104. Support sqlite3WhereIsSorted()
79105. orconf ::=
79106. ** "val" is a double such that $0.1 \leq *val < 10.0$ ** Return the ascii code for the leading digit of *val, then ** multiply "*val" by 10.0 to renormalize. ***
Example: ** input: *val = 3.14159 ** output: *val = 1.4159 function return = '3' *** ** The counter *cnt is incremented each time. After counter exceeds *** 16
(the number of significant digits in a 64-bit float) '0' is ** always returned.
79107. Parsing & code generating context
79108. A "special" INSERT op. These are handled separately.
79109. Index of WhereLoop in pPath being processed
79110. Remove all entries from a hash table. Reclaim all memory. ** Call this routine to delete a hash table or to reset a hash table ** to the empty state.
79111. SQL used to find output index
79112. Page data
79113. same as TK_BITAND, in1, in2, out3
79114. FTS5_NOT node to advance
79115. SQLITE_MAX_MMAP_SIZE>0
79116. Fts3 cursor for current query
79117. ** Free memory.
79118. Name of the open file
79119. JNODE flags
79120. Pointer to changeset buffer
79121. True if a m-j name has been written to jrn1
79122. Segment to delete
79123. ** CAPI3REF: Flush caches to disk mid-transaction *** ** ^If a write-transaction is open on [database connection] D when the ** [sqlite3_db_cacheflush(D)] interface invoked, any dirty ** pages in the pager-cache that are not currently in use are written out ** to disk. A dirty page may be in use if a database cursor created by an ** active SQL statement is reading from it, or if it is page 1 of a database ** file (page 1 is always "in use"). ^The [sqlite3_db_cacheflush(D)] ** interface flushes caches for all schemas - "main", "temp", and ** any [attached] databases. *** ** ^If this function needs to obtain extra database locks before dirty pages ** can be flushed to disk, it does so. ^If those locks cannot be obtained ** immediately and there is a busy-handler callback configured, it is invoked ** in the usual manner. ^If the required lock still cannot be obtained, then ** the database is skipped and an attempt made to flush any dirty pages ** belonging to the next (if any) database. ^If any databases are skipped ** because locks cannot be obtained, but no other error occurs, this ** function returns SQLITE_BUSY. *** ** ^If any other error occurs while flushing dirty pages to disk (for ** example an IO error or out-of-memory condition), then processing is ** abandoned and an SQLite [error code] is returned to the caller immediately. *** ** ^Otherwise, if no error occurs, [sqlite3_db_cacheflush()] returns SQLITE_OK. *** ** ^This function does not set the database handle error code or message ** returned by the [sqlite3_errcode()] and [sqlite3_errmsg()] functions.
79124. Column number of matching column on the left
79125. SQLITE_CHECKPOINT_* value
79126. ** 2008 Nov 28 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This module contains code that implements a parser for
fts3 query strings ** (the right-hand argument to the MATCH operator). Because the supported ** syntax is relatively simple, the whole tokenizer/parser system is
** hand-coded.
79127. 0 = main program, 1 = first sub-program etc.
79128. True when aMatchinfo[] needs filling in
79129. Open file descriptor
79130. Error code from db file unlock operation
79131. ** Examine a WherePath (with the addition of the extra WhereLoop of the 6th ** parameters) to see if it outputs rows in the requested ORDER BY ** (or GROUP BY) without requiring a separate sort operation. Return N: ** ** N>0: N terms of the ORDER BY clause are satisfied ** N==0: No terms of the ORDER BY clause are satisfied ** N<0: Unknown yet how many terms of ORDER BY might be satisfied. *** ** Note that processing for WHERE_GROUPBY and WHERE_DISTINCTBY is not as ** strict. With GROUP BY and DISTINCT the only requirement is that ** equivalent rows appear immediately adjacent to one another. GROUP BY ** and DISTINCT do not require rows to appear in any particular order as long ** as equivalent rows are grouped together. Thus for GROUP BY and DISTINCT ** the pOrderBy terms can be matched in any order. With ORDER BY, the ** pOrderBy terms must be matched in strict left-to-right order.
79132. ** Copy the contents of object (*pFrom) into (*pTo).
79133. Opcode: SeekLT P1 P2 P3 P4 * ** Synopsis: key=r[P3@P4] ** ** If cursor P1 refers to an SQL table (B-Tree that uses integer keys), ** use the value in register P3 as a key. If cursor P1 refers ** to an SQL index, then P3 is the first in an array of P4 registers ** that are used as an unpacked index key. *** ** Reposition cursor P1 so that it points to the largest entry that ** is less than the key value. If there are no records less than ** the key and P2 is not zero, then jump to P2. **
** This opcode leaves the cursor configured to move in reverse order, ** from the end toward the beginning. In other words, the cursor is ** configured to use Prev, not Next. ** ** See also: Found, NotFound, SeekGt, SeekGe, SeekLe
79134. Before doing anything else, call the xSync() callback for any ** virtual module tables written in this transaction. This has to ** be done before determining whether a master journal file is ** required, as an xSync() callback may add an attached database ** to the transaction.
79135. Result from sqlite3Reprepare()
79136. Index.aiRowEst may already be set here if there are duplicate ** sqlite_stat1 entries for this index. In that case just clobber ** the old data with the new instead of allocating a new array.
79137. ** The parser calls this routine in order to create a new VIEW
79138. Parent table of foreign key pFKey
79139. (324) vtabarglist ::= vtabarglist COMMA vtabarg
79140. ** Instances of this structure are used to build strings or binary records.
79141. Opcode: MakeRecord P1 P2 P3 P4 * ** Synopsis: r[P3]=mkrec(r[P1@P2]) ** ** Convert P2 registers beginning with P1 into the [record format] ** use as a data record in a database table or as a key ** in an index. The OP_Column opcode can decode the record later. *** ** P4 may be a string that is P2 characters long. The N-th character of the ** string indicates the column affinity that should be used for the N-th ** field of the index key. *** ** The mapping from character to affinity is given by the SQLITE_AFF_ ** macros defined in sqliteInt.h. *** ** If P4 is NULL then all index fields have the affinity BLOB.
79142. SELECTTRACE_ENABLED
79143. Number of pages in the database according to hdr
79144. Insert an implicit AND operator.
79145. ***** Begin file mutex_noop.c *****
79146. ON DELETE and ON UPDATE actions, respectively
79147. same as TK_LE, jump, in1, in3
79148. Docid to read size data for
79149. ifndef SQLITE OMIT_AUTOVACUUM
79150. (term <= \$zLeTerm) paramater, or NULL
79151. How to dispose of the results
79152. synopsis: r[P2]=P4
79153. Number of pages in pList
79154. Table structure

79155. The main DB, main journal, WAL file and master journal are never ** automatically deleted. Nor are they ever temporary files.
79156. Size of term suffix in bytes
79157. Adjust downward, as appropriate
79158. OUT: Bytes stored locally
79159. If this instruction implements a COMMIT and other VMs are writing ** return an error indicating that the other VMs must complete first.
79160. Parsing contexts
79161. Type of the RHS
79162. Cursor open on pIdx
79163. ** PRAGMA wal_autocheckpoint ** PRAGMA wal_autocheckpoint = N ** ** Configure a database connection to automatically checkpoint a database ** after accumulating N frames in the log. Or query for the current value ** of N.
79164. FTS3 table cursor
79165. Number of columns in the result set
79166. ** sqlite3_test_control(SQLITE_TESTCTRL_PENDING_BYTE, unsigned int X) ** ** Set the PENDING byte to the value in the argument, if X>0. ** Make no changes if X==0. Return the value of the pending byte ** as it existing before this routine was called. ** ** IMPORTANT: Changing the PENDING byte from 0x40000000 results in ** an incompatible database file format. Changing the PENDING byte ** while any database connection is open results in undefined and ** deleterious behavior.
79167. Used to iterate through zArg
79168. One-time setup cost (ex: create transient index)
79169. If an EXCLUSIVE lock can be obtained on the database file (using the ** ordinary, rollback-mode locking methods, this guarantees that the ** connection associated with this log file is the only connection to ** the database. In this case checkpoint the database and unlink both ** the wal and wal-index files. ** ** The EXCLUSIVE lock is not released before returning.
79170. **comment:** Clean up before exiting
label: code-design
79171. ** This is the xFilter implementation for the virtual table.
79172. Collision detection may be omitted if all of the following are true: ** (1) The conflict resolution algorithm is REPLACE ** (2) The table is a WITHOUT ROWID table ** (3) There are no secondary indexes on the table ** (4) No delete triggers need to be fired if there is a conflict ** (5) No FK constraint counters need to be updated if a conflict occurs.
79173. Pointer to buffer containing changeset B
79174. Register containing index name
79175. 440
79176. ** Internal types used by SQLite.
79177. Number of trailing arguments for rank()
79178. Number of entries currently in hash
79179. **comment:** ** Growing our own isspace() routine this way is twice as fast as ** the library isspace() function, resulting in a 7% overall performance ** increase for the parser. (Ubuntu14.10 gcc 4.8.4 x64 with -Os).
label: code-design
79180. **comment:** ** Close a file descriptor. ** ** We assume that close() almost always works, since it is only in a ** very sick application or on a very sick platform that it might fail. ** If it does fail, simply leak the file descriptor, but do log the ** error. ** ** Note that it is not safe to retry close() after EINTR since the ** file descriptor might have already been reused by another thread. ** So we don't even try to recover from an EINTR. Just log the error ** and move on.
label: code-design
79181. Flags this file descriptor was opened with
79182. ** The makefile scans the vdbe.c source file and creates the "opcodes.h" ** header file that defines a number for each opcode used by the VDBE.
79183. flags = BTREE_INTKEY;
79184. ** Parameters z and n contain a pointer to and length of a buffer containing ** an fts3 query expression, respectively. This function attempts to parse the ** query expression and create a tree of Fts3Expr structures representing the ** parsed expression. If successful, *ppExpr is set to point to the head ** of the parsed expression tree and SQLITE_OK is returned. If an error ** occurs, either SQLITE_NOMEM (out-of-memory error) or SQLITE_ERROR (parse ** error) is returned and *ppExpr is set to 0. ** ** If parameter n is a negative number, then z is assumed to point to a ** nul-terminated string and the length is determined using strlen(). ** ** The first parameter, pTokenizer, is passed the fts3 tokenizer module to ** use to normalize query tokens while parsing the expression. The azCol[] ** array, which is assumed to contain nCol entries, should contain the names ** of each column in the target fts3 table, in order from left to right. ** Column names must be nul-terminated strings. ** ** The iDefaultCol parameter should be passed the index of the table column ** that appears on the left-hand-side of the MATCH operator (the default ** column to match against for tokens for which a column name is not explicitly ** specified as part of the query string), or -1 if tokens may by default ** match any table column.
79185. Set iRoot to the index in pWriter->aNodeWriter[] of the output segment ** root node. If the segment fits entirely on a single leaf node, iRoot ** will be set to 0. If the root node is the parent of the leaves, iRoot ** will be 1. And so on.
79186. 228
79187. .flags =
79188. ***** End of os_win.c *****
79189. CPU clock count at start of opcode
79190. Full scan via index
79191. **comment:** If the cursor is currently on the last row and we are appending a ** new row onto the end, set the "loc" to avoid an unnecessary ** btreeMoveto() call
label: code-design
79192. If no data has been written to disk, then do not do so now. Instead, ** sort the VdbeSorter.pRecord list. The vdbe layer will read data directly ** from the in-memory list.
79193. FROM clause item
79194. Bytes required to hold database name
79195. Count the number of possible WHERE clause constraints referring ** to this virtual table
79196. Iterates through snippet candidates
79197. True if last 0x00 counts
79198. Authorization denied
79199. ** The journal file must be open when this is called. A journal header file ** (JOURNAL_HDR_SZ bytes) is read from the current location in the journal ** file. The current location in the journal file is given by ** pPager->journalOff. See comments above function writeJournalHdr() for ** a description of the journal header format. ** ** If the header is read successfully, *pNRec is set to the number of ** page records following this header and *pDbSize is set to the size of the ** database before the transaction began, in pages. Also, pPager->cksumInit ** is set to the value read from the journal header. SQLITE_OK is returned ** in this case. ** ** If the journal header file appears to be corrupted, SQLITE_DONE is ** returned and *pNRec and *pDbSize are undefined. If JOURNAL_HDR_SZ bytes ** cannot be read from the journal file an error code is returned.
79200. Write up to this many output leaves
79201. List of triggers on table pTab
79202. X not a subset of Y since term X[i] not used by Y
79203. ** An instance of this structure is used to iterate through the terms on ** a contiguous set of segment b-tree leaf nodes. Although the details of ** this structure are only manipulated by code in this file, opaque handles ** of type Fts3SegReader* are also used by code in fts3.c to iterate through ** terms when querying the full-text index. See functions: ** ** sqlite3Fts3SegReaderNew() ** sqlite3Fts3SegReaderFree() ** sqlite3Fts3SegReaderIterate() ** ** Methods used to manipulate Fts3SegReader structures: ** ** fts3SegReaderNext() ** fts3SegReaderFirstDocid() ** fts3SegReaderNextDocid()
79204. Flush the hash table to disk if required
79205. If the NEAR expression does not match any rows, zero the doclist for ** all phrases involved in the NEAR. This is because the snippet(), ** offsets() and matchinfo() functions are not supposed to recognize ** any instances of phrases that are part of unmatched NEAR queries. ** For example if this expression: ** ** ... MATCH 'a OR (b NEAR c)' ** ** is matched against a row containing: ** ** 'a b d e' ** ** then any snippet() should only highlight the "a" term, not the "b" ** (as "b" is part of a non-matching NEAR clause).
79206. 320

79207. ** Enlarge a memory allocation. If an out-of-memory allocation occurs, ** then free the old allocation.
79208. The path to search
79209. **comment:** ** CAPI3REF: Test To See If The Library Is Threadsafe *** ^The sqlite3_threadsafe() function returns zero if and only if ** SQLite was compiled with mutexing code omitted due to the ** [SQLITE_THREADS] compile-time option being set to 0. ** ** SQLite can be compiled with or without mutexes. When ** the [SQLITE_THREADS] C preprocessor macro is 1 or 2, mutexes ** are enabled and SQLite is threadsafe. When the ** [SQLITE_THREADS] macro is 0, ** the mutexes are omitted. Without the mutexes, it is not safe ** to use SQLite concurrently from more than one thread. ** ** Enabling mutexes incurs a measurable performance penalty. ** So if speed is of utmost importance, it makes sense to disable ** the mutexes. But for maximum safety, mutexes should be enabled. ** ^The default behavior is for mutexes to be enabled. ** ** This interface can be used by an application to make sure that the ** version of SQLite that it is linking against was compiled with ** the desired setting of the [SQLITE_THREADS] macro. ** ** This interface only reports on the compile-time mutex setting ** of the [SQLITE_THREADS] flag. If SQLite is compiled with ** SQLITE_THREADS=1 or =2 then mutexes are enabled by default but ** can be fully or partially disabled using a call to [sqlite3_config()] ** with the verbs [SQLITE_CONFIG_SINGLETHREAD], [SQLITE_CONFIG_MULTITHREAD], ** or [SQLITE_CONFIG_SERIALIZED]. ^The return value of the ** sqlite3_threadsafe() function shows only the compile-time setting of ** thread safety, not any run-time changes to that setting made by ** sqlite3_config(). In other words, the return value from sqlite3_threadsafe() ** is unchanged by calls to sqlite3_config().^ ** ** See the [threading mode] documentation for additional information.
label: requirement
79210. Hash table buckets
79211. ** End a write transaction. The commit has already been done. This ** routine merely releases the lock.
79212. Number of index columns w/o the pk/rowid
79213. Used by VdbeSorterCompare()
79214. Array of phrase freq. for current row
79215. **comment:** ** Generate a human-readable description of a WITH clause.
label: code-design
79216. IMP: R-20520-54086
79217. Set the error code of the destination database handle.
79218. ** Functions to serialize a 16 bit integer, 32 bit real number and ** 64 bit integer. The value returned is the number of bytes written ** to the argument buffer (always 2, 4 and 8 respectively).
79219. value_type() of apVal[0]
79220. True if info.nKey is valid
79221. Columns for imposter table
79222. This routine implements an SQL function that returns the "depth" parameter ** from the front of a blob that is an r-tree node. For example: ** ** SELECT rtreedepth(data) FROM rt_node WHERE nodeno=1; ** ** The depth value is 0 for all nodes other than the root node, and the root ** node always has nodeno=1, so the example above is the primary use for this ** routine. This routine is intended for testing and analysis only.
79223. sortorder ::= DESC
79224. rowid <= ? expression (or NULL)
79225. If this path name begins with "/X:", where "X" is any alphabetic ** character, discard the initial "/" from the pathname.
79226. same as TK_STAR, synopsis: r[P3]=r[P1]*r[P2]
79227. Number of key columns
79228. Zero the contents of pRoot. Then install pChild as the right-child.
79229. Index to consider domain of
79230. porter rule condition: (m > 1)
79231. Name of the trigger
79232. Make sure the database schema is loaded if the pragma requires that
79233. Tokenization callback context object
79234. UTF-8 encoded SQL statement.
79235. Byte offset to previous journal header
79236. column list if this is an UPDATE OF trigger
79237. Next element of the cell pointer array
79238. (pKey1/nKey1) is a blob
79239. Index of function argument.
79240. True to omit %_dsize table
79241. OUT: New db size (or 0 if not commit)
79242. ** Given a wildcard parameter name, return the index of the variable ** with that name. If there is no variable with the given name, ** return 0.
79243. Add additional columns needed to make the automatic index into ** a covering index
79244. The phrase extracted from pExpr
79245. #include <stdio.h>
79246. ** Set up a cursor object for iterating through a full-text index or a ** single level therein.
79247. Main database page size
79248. It is not possible to release (commit) a savepoint if there are ** active write statements.
79249. Language id for current row
79250. Mask of trigger times
79251. A single level in pWInfo->a[]
79252. Index.nSample is non-zero at this point if data has already been ** loaded from the stat4 table. In this case ignore stat3 data.
79253. Node number to load
79254. SQLITE_PROTOCOL? SQLITE_MISUSE?
79255. Number of columns in index being sampled
79256. An ORDER/GROUP BY clause of more than 63 terms cannot be optimized
79257. Output array
79258. Size of previous chunk in Mem3Block elements
79259. Value of julianday('now') for this statement
79260. Max database page to write
79261. ** Exported version of applyAffinity(). This one works on sqlite3_value*, ** not the internal Mem* type.
79262. ** Insert an OP_CursorHint instruction if it is appropriate to do so.
79263. Output of stat() on database file
79264. ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** Memory allocation functions used throughout sqlite.
79265. ** Read the first N bytes from the beginning of the file into memory ** that pDest points to. ** ** If the pager was opened on a transient file (zFilename==""), or ** opened on a file less than N bytes in size, the output buffer is ** zeroed and SQLITE_OK returned. The rationale for this is that this ** function is used to read database headers, and a new transient or ** zero sized database has a header that consists entirely of zeroes. ** ** If any IO error apart from SQLITE_IOERR_SHORT_READ is encountered, ** the error code is returned to the caller and the contents of the ** output buffer undefined.
79266. The result set must have exactly one column
79267. Eponymous table for this module
79268. An array to map all upper-case characters into their corresponding ** lower-case character. ** ** SQLite only considers US-ASCII (or EBCDIC) characters. We do not ** handle case conversions for the UTF character set since the tables ** involved are nearly as big or bigger than SQLite itself.
79269. Token (or prefix) to query for
79270. Fix the pointer map entries associated with the right-child of each ** sibling page. All other pointer map entries have already been taken ** care of.
79271. ** Unless this is an in-memory or temporary database, clear the pager cache.

79272. (2a)

79273. *** Tokenize the text passed via the second and third arguments. *** The callback is invoked once for each token in the input text. The ** arguments passed to it are, in order: *** void *pCtx // Copy of 4th argument to sqlite3Fts5Tokenize() *** const char *pToken // Pointer to buffer containing token *** int nToken // Size of token in bytes *** int iStart // Byte offset of start of token within input text *** int iEnd // Byte offset of end of token within input text *** int iPos // Position of token in input (first token is 0) *** If the callback returns a non-zero value the tokenization is abandoned *** and no further callbacks are issued. *** This function returns SQLITE_OK if successful or an SQLite error code *** if an error occurs. If the tokenization was abandoned early because *** the callback returned SQLITE_DONE, this is not an error and this function ** still returns SQLITE_OK. Or, if the tokenization was abandoned early *** because the callback returned another non-zero value, it is assumed *** to be an SQLite error code and returned to the caller.

79274. True for xCreate, false for xConnect

79275. It is possible to append to this segment. Set up the IncrmergeWriter ** object to do so.

79276. ** Return TRUE if the named file is really a directory. Return false if ** it is something other than a directory, or if there is any kind of memory ** allocation failure.

79277. cmd ::= COMMIT|END trans_opt

79278. a

79279. Added by the optimizer. Do not code

79280. defined(_MSC_VER) && _MSC_VER>=1400

79281. Loop counter: Next table in schema

79282. iArg:

79283. The following variables are used by fts3SegReaderNextDocid() to iterate ** through the current doclist (aDoclist/nDoclist).

79284. Wal frame to lock to

79285. 31

79286. **comment:** *** Iterator pIter currently points to the first rowid of a doclist. ** There is a doclist-index associated with the final term on the current ** page. If the current term is the last term on the page, load the ** doclist-index from disk and initialize an iterator at (pIter->pDlidx).
label: code-design

79287. Following a rollback, the database file should be back in its original ** state prior to the start of the transaction, so invoke the ** SQLITE_FCNTL_DB_UNCHANGED file-control method to disable the ** assertion that the transaction counter was modified.

79288. payload

79289. ** Allocate and populate an UnpackedRecord structure based on the serialized ** record in nKey/pKey. Return a pointer to the new UnpackedRecord structure ** if successful, or a NULL pointer if an OOM error is encountered.

79290. List of explicit column names, or NULL

79291. Size of the zMalloc allocation

79292. 100

79293. GROUPBY cover of ORDERBY

79294. Do not delete the table until the reference count reaches zero.

79295. Query the WAL sub-system for the database size. The WalDbsize() ** function returns zero if the WAL is not open (i.e. Pager.pWal==0), or ** if the database size is not available. The database size is not ** available from the WAL sub-system if the log file is empty or ** contains no valid committed transactions.

79296. Loop over all indices

79297. ** Object used to represent an shared memory buffer. *** When multiple threads all reference the same wal-index, each thread ** has its own unixShm object, but they all point to a single instance ** of this unixShmNode object. In other words, each wal-index is opened ** only once per process. *** Each unixShmNode object is connected to a single unixInodeInfo object. ** We could coalesce this object into unixInodeInfo, but that would mean ** every open file that does not use shared memory (in other words, most ** open files) would have to carry around this extra information. So ** the unixInodeInfo object contains a pointer to this unixShmNode object ** and the unixShmNode object is created only when needed. *** unixMutexHeld() must be true when creating or destroying ** this object or while reading or writing the following fields: *** nRef *** The following fields are read-only after the object is created: *** fid ** zFilename *** Either unixShmNode.mutex must be held or unixShmNode.nRef==0 and ** unixMutexHeld() is true when reading or writing any other field ** in this structure.

79298. ** Test a register to see if it exceeds the current maximum blob size. ** If it does, record the new maximum blob size.

79299. number of rows in the table

79300. ** Macros to Set(), Clear() and Test() cursor flags.

79301. SELECT may not have an ORDER BY clause

79302. Inequality constraint at range end

79303. True if OP_Affinity has been run already

79304. Opcode: HaltIfNull P1 P2 P3 P4 P5 ** Synopsis: if r[P3]=null halt *** Check the value in register P3. If it is NULL then Halt using ** parameter P1, P2, and P4 as if this were a Halt instruction. If the ** value in register P3 is not NULL, then this routine is a no-op. ** The P5 parameter should be 1.

79305. Current index for apPage[iPage]

79306. Continue rolling back records out of the main journal starting at ** the first journal header seen and continuing until the effective end ** of the main journal file. Continue to skip out-of-range pages and ** continue adding pages rolled back to pDone.

79307. This virtual table always delivers results in ascending order of ** the "term" column (column 0). So if the user has requested this ** specifically - "ORDER BY term" or "ORDER BY term ASC" - set the ** sqlite3_index_info.orderByConsumed flag to tell the core the results ** are already in sorted order.

79308. Comparing signed and unsigned

79309. ** Generate code for a boolean expression such that a jump is made ** to the label "dest" if the expression is false but execution ** continues straight thru if the expression is true. *** If the expression evaluates to NULL (neither true nor false) then ** jump if jumpIfNull is SQLITE_JUMPIFNULL or fall through if jumpIfNull ** is 0.

79310. ** Register all built-in tokenizers with FTS5.

79311. If a journal file exists, and there is no RESERVED lock on the ** database file, then it either needs to be played back or deleted.

79312. Bytes of space to allocate

79313. sqlite3_create_collation only

79314. Document being tokenized

79315. Unless i==Idx->nSample, indicating that pRec is larger than ** all samples in the aSample[] array, pRec must be smaller than the ** (iCol+1) field prefix of sample i.

79316. ** Maximum depth of recursion for triggers. *** A value of 1 means that a trigger program will not be able to itself ** fire any triggers. A value of 0 means that no trigger programs at all ** may be executed.

79317. Code an OP_Transaction and OP_TableLock for <table>.

79318. ** Call sqlite3WalkExpr() for every expression in Select statement p. ** Invoke sqlite3WalkSelect() for subqueries in the FROM clause and ** on the compound select chain, p->pPrior. *** If it is not NULL, the xSelectCallback() callback is invoked before ** the walk of the expressions and FROM clause. The xSelectCallback2() ** method is invoked following the walk of the expressions and FROM clause, ** but only if both xSelectCallback and xSelectCallback2 are both non-NULL ** and if the expressions and FROM clause both return WRC_Continue; ** ** Return WRC_Continue under normal conditions. Return WRC_Abort if ** there is an abort request. *** If the Walker does not have an xSelectCallback() then this routine ** is a no-op returning WRC_Continue.

79319. Return code from subroutines

79320. If pPrior is part of the data area of pPage, then make sure pPage ** is still writeable

79321. Path to this page

79322. If an expression is an integer literal that fits in a signed 32-bit ** integer, then the EP_IntValue flag will have already been set

79323. Used for rounding floating point values

79324. ** Return information regarding the PRIMARY KEY and number of columns in ** the database table affected by the change that pIter currently points ** to. This function may only be called after changeset_next() returns ** SQLITE_ROW.

79325. Trigger Name Table Name

79326. ** These macros can be used to test, set, or clear bits in the ** Db.pSchema->flags field.

79327. Call the progress callback if it is configured and the required number ** of VDBE ops have been executed (either since this invocation of ** sqlite3VdbeExec() or since last time the progress callback was called). ** If the progress callback returns non-zero, exit the virtual machine with ** a return code SQLITE_ABORT.

79328. 178
79329. Vdbe to add code to
79330. "nlt" column of stat[34] entry
79331. ** pEList is the SET clause of an UPDATE statement. Each entry ** in pEList is of the format <id>=<expr>. If any of the entries ** in pEList have an <id> which matches an identifier in pIdList, ** then return TRUE. If pIdList==NULL, then it is considered a ** wildcard that matches anything. Likewise if pEList==NULL then ** it matches anything so always return true. Return false only ** if there is no match.
79332. ** Create an iterator used to iterate through the contents of a changeset.
79333. Open the target, RBU and state databases
79334. ***** Begin file prepare.c *****
79335. Built-in affinity() function
79336. The longest decimal representation of a 32 bit integer is 10 digits: ** ** 1234567890 ** 2^31 -> 2147483648
79337. page-size bytes of space for parent ovfl
79338. Parent table of FK pFKey
79339. Rowid of last value written
79340. All (named) indices indexed by name
79341. **comment:** ** A call to this routine tells the pager that it is not necessary to ** write the information on page pPg back to the disk, even though ** that page might be marked as dirty. This happens, for example, when ** the page has been added as a leaf of the freelist and so its ** content no longer matters. ** ** The overlying software layer calls this routine when all of the data ** on the given page is unused. The pager marks the page as clean so ** that it does not get written to disk. ** ** Tests show that this optimization can quadruple the speed of large ** DELETE operations. ** ** This optimization cannot be used with a temp-file, as the page may ** have been dirty at the start of the transaction. In that case, if ** memory pressure forces page pPg out of the cache, the data does need ** to be written out to disk so that it may be read back in if the ** current transaction is rolled back.
label: code-design
79342. ***** End of vdbe.c *****
79343. ** CAPI3REF: Collation Needed Callbacks ** METHOD: sqlite3 *** ^To avoid having to register all collation sequences before a database ** can be used, a single callback function may be registered with the ** [database connection] to be invoked whenever an undefined collation ** sequence is required. ** ** ^If the function is registered using the sqlite3_collation_needed() API, ** then it is passed the names of undefined collation sequences as strings ** encoded in UTF-8. ^If sqlite3_collation_needed16() is used, ** the names are passed as UTF-16 in machine native byte order. ** ^A call to either function replaces the existing collation-needed callback. ** ** ^When the callback is invoked, the first argument passed is a copy ** of the second argument to sqlite3_collation_needed() or ** sqlite3_collation_needed16(). The second argument is the database ** connection. The third argument is one of [SQLITE_UTF8], [SQLITE_UTF16BE], ** or [SQLITE_UTF16LE], indicating the most desirable form of the collation ** sequence function required. The fourth parameter is the name of the ** required collation sequence.)^ ** ** The callback function should register the desired collation using ** [sqlite3_create_collation()], [sqlite3_create_collation16()], or ** [sqlite3_create_collation_v2()].
79344. Open the sub-journal, if it has not already been opened
79345. Size of the free slot
79346. Result codes from subroutines
79347. same as TK_LSHIFT, in1, in2, out3
79348. Operational flags
79349. To get here, there need to be 2^(N_SORT_BUCKET) elements in ** the input list. But that is impossible.
79350. Key for ARRAY objects in json_tree()
79351. Even though the Mem structure contains an element ** of type i64, on certain architectures (x86) with certain compiler ** switches (-Os), gcc may align this Mem object on a 4-byte boundary ** instead of an 8-byte one. This all works fine, except that when ** running with SQLITE_DEBUG defined the SQLite code sometimes assert(s) ** that a Mem structure is located on an 8-byte boundary. To prevent ** these assert(s) from failing, when building with SQLITE_DEBUG defined ** using gcc, we force nullMem to be 8-byte aligned using the magical ** __attribute__((aligned(8))) macro.
79352. ** Return an expression that can be used in a WHERE clause to match the ** primary key of the current table. For example, if the table is: ** ** CREATE TABLE t1(a, b, c, PRIMARY KEY(b, c)); ** ** Return the string: ** ** "b = ?1 AND c = ?2"
79353. case_else ::=
79354. ***** Above is constant between recursions. Below is reset before and after ** each recursion. The boundary between these two regions is determined ** using offsetof(Parse,sLastToken) so the sLastToken field must be the ** first field in the recursive region. *****
79355. Array of all phrases
79356. Find the parent table of this foreign key. Also find a unique index ** on the parent key columns in the parent table. If either of these ** schema items cannot be located, set an error in pParse and return ** early.
79357. ** The following table defines various date transformations of the form *** 'NNN days' *** Where NNN is an arbitrary floating-point number and "days" can be one ** of several units of time.
79358. **comment:** ** The maximum number of terms in a compound SELECT statement. ** The code generator for compound SELECT statements does one ** level of recursion for each term. A stack overflow can result ** if the number of terms is too large. In practice, most SQL ** never has more than 3 or 4 terms. Use a value of 0 to disable ** any limit on the number of terms in a compound SELECT.
label: code-design
79359. Leaf node containing record iDelete
79360. The OFFSET clause. May be null
79361. Where to write results
79362. ***** End of fts3_hash.c *****
79363. ** If the path name starts with a letter and a colon it is either a volume ** relative path or an absolute path. Callers of this function must not ** attempt to treat it as a relative path name (i.e. they should simply use ** it verbatim).
79364. SQLITE3_H
79365. The "table" is actually a sub-select or a view in the FROM clause ** of the SELECT statement. Return the declaration type and origin ** data for the result-set column of the sub-select.
79366. Segment to check internal consistency
79367. Case 2: expr IN (exprlist) *** For each expression, build an index key from the evaluation and ** store it in the temporary table. If <expr> is a column, then use ** that columns affinity when building index keys. If <expr> is not ** a column, use numeric affinity.
79368. TUNING: Cost of a rowid lookup is 10
79369. ** NOTE: We are dealing with a relative path name and the data ** directory has been set. Therefore, use it as the basis ** for converting the relative path name to an absolute ** one by prepending the data directory and a slash.
79370. Checksum failed.
79371. Tables that must be scanned before this one
79372. The major type of the error token
79373. If the index name was unqualified, check if the table ** is a temp table. If so, set the database to 1. Do not do this ** if initialising a database schema.
79374. Minimum entry in aFirst[] to set
79375. Last zStr[] that might match zPattern[]
79376. URI parameter sought
79377. OUT: Filename component of URI
79378. This occurs when the array of context pointers that need to ** be passed to the unlock-notify callback is larger than the ** aStatic[] array allocated on the stack and the attempt to ** allocate a larger array from the heap has failed. ** ** This is a difficult situation to handle. Returning an error ** code to the caller is insufficient, as even if an error code ** is returned the transaction on connection db will still be ** closed and the unlock-notify callbacks on blocked connections ** will go unissued. This might cause the application to wait ** indefinitely for an unlock-notify callback that will never ** arrive. ** ** Instead, invoke the unlock-notify callback with the context ** array already accumulated. We can then clear the array and ** begin accumulating any further context pointers without ** requiring any dynamic allocation. This is sub-optimal because ** it means that instead of one callback with a large array of ** context pointers the application will receive

two or more ** callbacks with smaller arrays of context pointers, which will ** reduce the applications ability to prioritize multiple ** connections. But it is the best that can be done under the ** circumstances.

79379. This call cannot fail. Unless the page for which the page number ** is passed as the second argument is (a) in the cache and ** (b) has an outstanding reference, then xUndo is either a no-op ** (if (a) is false) or simply expels the page from the cache (if (b) ** is false). *** If the upper layer is doing a rollback, it is guaranteed that there ** are no outstanding references to any page other than page 1. And ** page 1 is never written to the log until the transaction is ** committed. As a result, the call to xUndo may not fail.

79380. True once a write-transaction is open on pDest

79381. Invoke the update-hook if required.

79382. ** No-op routine for the parse-tree walker for SELECT statements. ** subquery in the parser tree.

79383. Page number array

79384. A PENDING lock is needed before acquiring a SHARED lock and before ** acquiring an EXCLUSIVE lock. For the SHARED lock, the PENDING will ** be released.

79385. ** This is the xRowid method. The SQLite core calls this routine to ** retrieve the rowid for the current row of the result set. fts5 ** exposes %_content.rowid as the rowid for the virtual table. The ** rowid should be written to *pRowid.

79386. The database being initialized

79387. Do not allow non-admin users to modify the schema arbitrarily

79388. Do not enforce check constraints

79389. **comment:** ** This function is invoked by the vdbe to call the xCreate method ** of the virtual table named zTab in database iDb. *** If an error occurs, *pzErr is set to point to an English language ** description of the error and an SQLITE_XXX error code is returned. ** In this case the caller must call sqlite3DbFree(db,) on *pzErr.

label: code-design

79390. A cursor number

79391. Routine for en/decoding data

79392. 22

79393. OUT: Score of snippet pFragment

79394. If nRec is 0xffffffff, then this journal was created by a process ** working in no-sync mode. This means that the rest of the journal ** file consists of pages, there are no more journal headers. Compute ** the value of nRec based on this assumption.

79395. The expression to be analyzed

79396. aSpare array

79397. The module for this virtual table

79398. The old journal mode

79399. **comment:** ** This is the default collating function named "BINARY" which is always ** available. *** If the padFlag argument is not NULL then space padding at the end ** of strings is ignored. This implements the RTRIM collation.

label: code-design

79400. Use of a hash implementation is OK

79401. ** Close the file.

79402. Rowid to append

79403. Allocate any page <= the parameter

79404. no-op

79405. If there is a table-filter configured, invoke it. If it returns 0, ** do not automatically add the new table.

79406. The following value is the maximum cell size assuming a maximum page ** size give above.

79407. For looping over sibling connections

79408. 0 for min() or 0xffffffff for max()

79409. segdir.end_block

79410. ** Maximum length (in bytes) of the pattern in a LIKE or GLOB ** operator.

79411. term ::= NULL|FLOAT|BLOB

79412. HAVING to WHERE clause ctx

79413. Left key

79414. Default page size

79415. ** Adjust memory usage statistics

79416. OP_Column: Ephemeral output is ok

79417. "PRAGMA encoding = XXX"

79418. NB: seekResult does not distinguish between "no seeks have ever occurred ** on this cursor" and "the most recent seek was an exact match".

79419. cmd ::= ROLLBACK trans_opt TO savepoint_opt nm

79420. Current offset in position list

79421. ** This routine translates a standard POSIX errno code into something ** useful to the clients of the sqlite3 functions. Specifically, it is ** intended to translate a variety of "try again" errors into SQLITE_BUSY ** and a variety of "please close the file descriptor NOW" errors into ** SQLITE_IOERR *** Errors during initialization of locks, or file system support for locks, ** should handle ENOLCK, ENOTSUP, EOPNOTSUPP separately.

79422. 149

79423. ***** Begin file sqlite3ext.h *****

79424. ** Interfaces for opening a shared library, finding entry points ** within the shared library, and closing the shared library.

79425. SQLITE_STMTSTATUS_REPREPARE

79426. Open the backend database driver

79427. ** The pExpr should be a TK_COLUMN expression. The table referred to ** is in pTabList or else it is the NEW or OLD table of a trigger. *** Check to see if it is OK to read this particular column. *** If the auth function returns SQLITE_IGNORE, change the TK_COLUMN ** instruction into a TK_NULL. If the auth function returns SQLITE_DENY, ** then generate an error.

79428. OP_Column/OP_Rowid references to this table

79429. Bytes of space required

79430. Used by: lock_status

79431. Select the column is extracted from

79432. The file is empty (length 0 bytes)

79433. ** Return the device characteristic flags supported by an rbuVfs-file.

79434. ** CAPI3REF: Database Connection For Functions ** METHOD: sqlite3_context *** ^The sqlite3_context_db_handle() interface returns a copy of ** the pointer to the [database connection] (the 1st parameter) ** of the [sqlite3_create_function()] ** and [sqlite3_create_function16()] routines that originally ** registered the application defined function.

79435. Determine how many of the segments actually point to zTerm/nTerm.

79436. SQLITE_MSVC_H

79437. Do not allow EXCLUSIVE locks. Preventing SQLite from taking this ** prevents it from checkpointing the database from sqlite3_close().

79438. The data

79439. TO => nothing

79440. **comment:** ** The SQLITE_*_BKPT macros are substitutes for the error codes with ** the same name but without the _BKPT suffix. These macros invoke ** routines that report the line-number on which the error originated ** using sqlite3_log(). The routines also provide a convenient place ** to set a debugger breakpoint.

label: code-design

79441. Finalize all SQL statements

79442. ** Launch a background thread to populate aFile[1] of pIncr.

79443. ** This function is called before generating code to update or delete a ** row contained in table pTab.

79444. EVIDENCE-OF: R-08308-17224 The default collating function for all ** strings is BINARY.

79445. One bit for each page in the database file
79446. Invoke the stat_init() function. The arguments are: ** (1) the number of columns in the index including the rowid ** (or for a WITHOUT ROWID table, the number of PK columns), ** (2) the number of columns in the key without the rowid/pk ** (3) the number of rows in the index, ** ** ** The third argument is only used for STAT3 and STAT4
79447. ** An instance of the following structure is used by the tree walker ** to determine if an expression can be evaluated by reference to the ** index only, without having to do a search for the corresponding ** table entry. The IdxCover.pIdx field is the index. IdxCover.iCur ** is the cursor for the table.
79448. A b-tree rebalance will be required after deleting this entry. ** Save the cursor key.
79449. Otherwise see if some other process holds it.
79450. The principle used to locate the table name in the CREATE TRIGGER ** statement is that the table name is the first token that is immediately ** preceded by either TK_ON or TK_DOT and immediately followed by one ** of TK_WHEN, TK_BEGIN or TK_FOR.
79451. Length of a field
79452. ** Convert a sorted list of elements (connected by pRight) into a binary ** tree with depth of iDepth. A depth of 1 means the tree contains a single ** node taken from the head of *ppList. A depth of 2 means a tree with ** three nodes. And so forth. ** ** Use as many entries from the input list as required and update the ** *ppList to point to the unused elements of the list. If the input ** list contains too few elements, then construct an incomplete tree ** and leave *ppList set to NULL. ** ** Return a pointer to the root of the constructed binary tree.
79453. ** Private objects used by the sorter
79454. ** Implementation of random(). Return a random integer.
79455. Already thrown the error if VDBE alloc failed
79456. ** PRAGMA [schema.]journal_size_limit ** PRAGMA [schema.]journal_size_limit=N ** ** Get or set the size limit on rollback journal files.
79457. The expression is a sub-select. Return the declaration type and ** origin info for the single column in the result set of the SELECT ** statement.
79458. The strings are already in the correct encoding. Call the ** comparison function directly
79459. VIEW
79460. Parsed information on cells being balanced
79461. colset ::= MINUS LCP colsetlist RCP
79462. restriction (3)
79463. Sequence counter
79464. Opcode: TableLock P1 P2 P3 P4 * ** Synopsis: IDb=P1 root=P2 write=P3 ** ** Obtain a lock on a particular table. This instruction is only used when ** the shared-cache feature is enabled. ** ** P1 is the index of the database in sqlite3.aDb[] of the database ** on which the lock is acquired. A deadlock is obtained if P3==0 or ** a write lock if P3==1. ** ** P2 contains the root-page of the table to lock. ** ** P4 contains a pointer to the name of the table being locked. This is only ** used to generate an error message if the lock cannot be obtained.
79465. Encode the database
79466. logical database name
79467. ** Return the size of the common prefix (if any) shared by zPrev and ** zNext, in bytes. For example, ** ** fts3PrefixCompress("abc", 3, "abcdef", 6) // returns 3
** fts3PrefixCompress("abX", 3, "abcdef", 6) // returns 2 ** fts3PrefixCompress("abX", 3, "Xbcd", 6) // returns 0
79468. EVIDENCE-OF: R-59841-13798 The 4-byte big-endian integer at offset 32 ** stores the page number of the first page of the freelist, or zero if ** the freelist is empty.
79469. **comment:** TODO(shess) Explore whether partially flushing the buffer on ** forced-flush would provide better performance. I suspect that if ** we ordered the doclists by size and flushed the largest until the ** buffer was half empty, that would let the less frequent terms ** generate longer doclists.
label: code-design
79470. Cache size in bytes
79471. Number of values in apVal[] array
79472. Catch mismatch in the declared columns of a view and the number of ** columns in the SELECT on the RHS
79473. The parse tree for this expression
79474. The type of lock held on this fd
79475. Action to take
79476. Non-logarithmic number of rows in the index
79477. (1)
79478. If the page containing the entry to delete is not a leaf page, move ** the cursor to the largest entry in the tree that is smaller than ** the entry being deleted. This cell will replace the cell being deleted ** from the internal node. The 'previous' entry is used for this instead ** of the 'next' entry, as the previous entry is always a part of the ** sub-tree headed by the child page of the cell being deleted. This makes ** balancing the tree following the delete operation easier.
79479. Case 5: Two or more separately indexed terms connected by OR ** ** Example: ** ** CREATE TABLE t1(a,b,c,d); ** CREATE INDEX i1 ON t1(a); **
CREATE INDEX i2 ON t1(b); ** CREATE INDEX i3 ON t1(c); ** ** SELECT * FROM t1 WHERE a=5 OR b=7 OR (c=11 AND d=13) ** ** In the example, there are three indexed terms connected by OR. ** The top of the loop looks like this: ** ** Null 1 # Zero the rowset in reg 1 ** ** Then, for each indexed term, the following. The arguments to ** RowSetTest are such that the rowid of the current row is inserted ** into the RowSet. If it is already present, control skips the ** Gosub opcode and jumps straight to the code generated by WhereEnd(). ** ** sqlite3WhereBegin(<term>) ** RowSetTest # Insert rowid into rowset ** Gosub 2 A ** sqlite3WhereEnd() ** ** Following the above, code to terminate the loop. Label A, the target ** of the Gosub above, jumps to the instruction right after the Goto. ** ** Null 1 # Zero the rowset in reg 1 ** Goto B # The loop is finished. ** ** A: <loop body> # Return data, whatever. ** ** Return 2 # Jump back to the Gosub ** ** B: <after the loop> ** ** Added 2014-05-26: If the table is a WITHOUT ROWID table, then ** use an ephemeral index instead of a RowSet to record the primary ** keys of the rows we have already seen. **
79480. Autovac setting after VACUUM if >=0
79481. ** Make sure that rand_s() is available on Windows systems with MSVC 2005 ** or higher.
79482. EXISTS
79483. LHS of comparison
79484. IN operator used for membership test
79485. 0x07
79486. ** A complete page cache is an instance of this structure. Every ** entry in the cache holds a single page of the database file. The ** btree layer only operates on the cached copy of the database pages. ** ** A page cache entry is "clean" if it exactly matches what is currently ** on disk. A page is "dirty" if it has been modified and needs to be ** persisted to disk. ** ** pDirty, pDirtyTail, pSynced: ** All dirty pages are linked into the doubly linked list using ** PgHdr.pDirtyNext and pDirtyPrev. The list is maintained in LRU order ** such that p was added to the list more recently than p->pDirtyNext. ** PCache.pDirty points to the first (newest) element in the list and ** pDirtyTail to the last (oldest). ** ** The PCache.pSynced variable is used to optimize searching for a dirty ** page to eject from the cache mid-transaction. It is better to eject ** a page that does not require a journal sync than one that does. ** Therefore, pSynced is maintained to that it *almost* always points ** to either the oldest page in the pDirty/pDirtyTail list that has a ** clear PGHDR_NEED_SYNC flag or to a page that is older than this one. ** (so that the right page to eject can be found by following pDirtyPrev ** pointers).
79487. This is the first call to this function. Do initialization.
79488. UNIQUE
79489. FROM clause of the outer query
79490. 2: (end_constraints && bRev && !endEq)
79491. Memory cell holding key of row to be deleted
79492. Database handle.
79493. **comment:** ** Check to make sure we have a valid db pointer. This test is not ** foolproof but it does provide some measure of protection against ** misuse of the interface such as passing in db pointers that are ** NULL or which have been previously closed. If this routine returns ** 1 it means that the db pointer is valid and 0 if it should not be ** dereferenced for any reason. The calling function should invoke ** SQLITE_MISUSE immediately. ** ** sqlite3SafetyCheckOk() requires that the db pointer be valid for ** use. sqlite3SafetyCheckSickOrOk() allows a db pointer that failed to ** open properly and is not fit for general use but which can be ** used as an argument to sqlite3_errmsg() or sqlite3_close().
label: code-design
79494. ** Initialize the iterator object indicated by the final parameter to ** iterate through coalesced phrase instances in column iCol.
79495. b &= (0x7f<<14)|(0x7f);
79496. If pSelect!=0, the id of the sub-select in EQP

79497. Advance Pager.journalOff to the start of the next sector. If the ** journal file is too small for there to be a header stored at this ** point, return SQLITE_DONE.
79498. Opcode: NextIfOpen P1 P2 P3 P4 P5 ** ** This opcode works just like Next except that if cursor P1 is not ** open it behaves a no-op.
79499. Reader whose cursor is to be moved
79500. ** Implementation of xSync() method.
79501. ** Obtain a buffer containing a changeset representing the concatenation ** of all changesets added to the group so far.
79502. ** This is the xEof method of the virtual table. SQLite calls this ** routine to find out if it has reached the end of a result set.
79503. Get the size of the index entry. Only indices entries of less ** than 2GiB are support - anything large must be database corruption. ** Any corruption is detected in sqlite3BtreeParseCellPtr(), though, so ** this code can safely assume that nCellKey is 32-bits
79504. ** This function is called to obtain a shared lock on the database file. ** It is illegal to call sqlite3PagerGet() until after this function ** has been successfully called. If a shared-lock is already held when ** this function is called, it is a no-op. ** ** The following operations are also performed by this function. ** ** 1) If the pager is currently in PAGER_OPEN state (no lock held ** on the database file), then an attempt is made to obtain a ** SHARED lock on the database file. Immediately after obtaining ** the SHARED lock, the file-system is checked for a hot-journal, ** which is played back if present. Following any hot-journal ** rollback, the contents of the cache are validated by checking ** the 'change-counter' field of the database file header and ** discarded if they are found to be invalid. ** ** 2) If the pager is running in exclusive-mode, and there are currently ** no outstanding references to any pages, and is in the error state, ** then an attempt is made to clear the error state by discarding ** the contents of the page cache and rolling back any open journal ** file. ** ** If everything is successful, SQLITE_OK is returned. If an IO error ** occurs while locking the database, checking for a hot-journal file or ** rolling back a journal file, the IO error code is returned.
79505. List of triggers to return
79506. Segment just written
79507. Space characters
79508. 235
79509. cmd ::= create_vtab
79510. **comment:** Disallow the transfer optimization if the destination table contains ** any foreign key constraints. This is more restrictive than necessary. ** But the main beneficiary of the transfer optimization is the VACUUM ** command, and the VACUUM command disables foreign key constraints. So ** the extra complication to make this rule less restrictive is probably ** not worth the effort. Ticket [6284df89debdfa61db8073e062908af0c9b6118e]
label: code-design
79511. Handle for accessing the file
79512. Number of references to this structure
79513. This is a threadsafe build, but strerror_r() is not available.
79514. ** Move the cursor up to the parent page. ** ** pCur->idx is set to the cell index that contains the pointer ** to the page we are coming from. If we are coming from the ** right-most child page then pCur->idx is set to one more than ** the largest cell index.
79515. ** CAPI3REF: Interrupt A Long-Running Query ** METHOD: sqlite3 *** ^This function causes any pending database operation to abort and ** return at its earliest opportunity. This routine is typically ** called in response to a user action such as pressing "Cancel" ** or Ctrl-C where the user wants a long query operation to halt ** immediately. ** ** ^It is safe to call this routine from a thread different from the ** thread that is currently running the database operation. But it ** is not safe to call this routine with a [database connection] that ** is closed or might close before sqlite3_interrupt() returns. ** ** ^If an SQL operation is very nearly finished at the time when ** sqlite3_interrupt() is called, then it might not have an opportunity ** to be interrupted and might continue to completion. ** ** ^An SQL operation that is interrupted will return [SQLITE_INTERRUPT]. ** ^If the interrupted SQL operation is an INSERT, UPDATE, or DELETE ** that is inside an explicit transaction, then the entire transaction ** will be rolled back automatically. ** ** ^The sqlite3_interrupt(D) call is in effect until all currently running ** SQL statements on [database connection] D complete. ^Any new SQL statements ** that are started after the sqlite3_interrupt() call and before the ** running statements reaches zero are interrupted as if they had been ** running prior to the sqlite3_interrupt() call. ^New SQL statements ** that are started after the running statement count reaches zero are ** not effected by the sqlite3_interrupt(). ** ^A call to sqlite3_interrupt(D) that occurs when there are no running ** SQL statements is a no-op and has no effect on SQL statements ** that are started after the sqlite3_interrupt() call returns.
79516. Detach the ORDER BY clause from the compound SELECT
79517. No sort order specified
79518. Also fsync the directory containing the file if the DIRSYNC flag ** is set. This is a one-time occurrence. Many systems (examples: AIX) ** are unable to fsync a directory, so ignore errors on the fsync.
79519. **comment:** ** The testcase() macro is used to aid in coverage testing. When ** doing coverage testing, the condition inside the argument to ** testcase() must be evaluated both true and false in order to ** get full branch coverage. The testcase() macro is inserted ** to help ensure adequate test coverage in places where simple ** condition/decision coverage is inadequate. For example, testcase() ** can be used to make sure boundary values are tested. For ** bitmask tests, testcase() can be used to make sure each bit ** is significant and used at least once. On switch statements ** where multiple cases go to the same block of code, testcase() ** can insure that all cases are evaluated. **
label: test
79520. All OP_Destroy operations occur on the same btree
79521. OUT: Statement handle
79522. The following block updates the change-counter. Exactly how it ** does this depends on whether or not the atomic-update optimization ** was enabled at compile time, and if this transaction meets the ** runtime criteria to use the operation: ** ** * The file-system supports the atomic-write property for ** blocks of size page-size, and ** * This commit is not part of a multi-file transaction, and ** * Exactly one page has been modified and store in the journal file. ** ** If the optimization was not enabled at compile time, then the ** pager_incr_changecounter() function is called to update the change ** counter in 'indirect-mode'. If the optimization is compiled in but ** is not applicable to this transaction, call sqlite3JournalCreate() ** to make sure the journal file has actually been created, then call ** pager_incr_changecounter() to update the change-counter in indirect ** mode. ** ** Otherwise, if the optimization is both enabled and applicable, ** then call pager_incr_changecounter() to update the change-counter ** in 'direct' mode. In this case the journal file will never be ** created for this transaction.
79523. refarg ::= ON INSERT refact
79524. ** Before a virtual table xCreate() or xConnect() method is invoked, the ** sqlite3.pVtabCtx member variable is set to point to an instance of ** this struct allocated on the stack. It is used by the implementation of ** the sqlite3_declare_vtab() and sqlite3_vtab_config() APIs, both of which ** are invoked only from within xCreate and xConnect methods.
79525. Error code from system call errors
79526. Value of the next token
79527. TEMP => ID
79528. Value returned by authorization callback
79529. The BTREE_SEEK_EQ flag is only set on index cursors
79530. Write the SQLite file handle here
79531. Size, in bits, of the bitmap element.
79532. Add the opcode to this VM
79533. ** Deserialize a single record from a buffer in memory. See "RECORD FORMAT" ** for details. ** ** When this function is called, *paChange points to the start of the record ** to deserialize. Assuming no error occurs, *paChange is set to point to ** one byte after the end of the same record before this function returns. ** If the argument abPK is NULL, then the record contains nCol values. Or, ** if abPK is other than NULL, then the record contains only the PK fields ** (in other words, it is a patchset DELETE record). ** ** If successful, each element of the apOut[] array (allocated by the caller) ** is set to point to an sqlite3_value object containing the value read ** from the corresponding position in the record. If that value is not ** included in the record (i.e. because the record is part of an UPDATE change ** and the field was not modified), the corresponding element of apOut[] is ** set to NULL. ** ** It is the responsibility of the caller to free all sqlite_value structures ** using sqlite3_free(). ** ** If an error occurs, an SQLite error code (e.g. SQLITE_NOMEM) is returned. ** The apOut[] array may have been partially populated in this case.
79534. Pointers to phrase objects
79535. ** Macros for troubleshooting. Normally turned off
79536. ** CAPI3REF: Obtain Values For URI Parameters ** ** These are utility routines, useful to VFS implementations, that check ** to see if a database file was a URI that contained a specific query ** parameter, and if so obtains the value of that query parameter. ** ** If F is the database filename pointer passed into the xOpen() method of ** a VFS implementation when the flags parameter to xOpen() has one or ** more of the [SQLITE_OPEN_URI] or [SQLITE_OPEN_MAIN_DB] bits set and ** P is the name of the query parameter, then ** sqlite3_uri_parameter(F,P) returns the value of the P ** parameter if it exists or a NULL pointer if P does not appear as a ** query parameter on F. If P is a query parameter of F ** has no explicit value, then sqlite3_uri_parameter(F,P)

returns ** a pointer to an empty string. *** The sqlite3_uri_boolean(F,P,B) routine assumes that P is a boolean ** parameter and returns true (1) or false (0) according to the value ** of P. The sqlite3_uri_boolean(F,P,B) routine returns true (1) if the ** value of query parameter P is one of "yes", "true", or "on" in any ** case or if the value begins with a non-zero number. The ** sqlite3_uri_boolean(F,P,B) routines returns false (0) if the value of ** query parameter P is one of "no", "false", or "off" in any case or ** if the value begins with a numeric zero. If P is not a query ** parameter on F or if the value of P is does not match any of the ** above, then sqlite3_uri_boolean(F,P,B) returns (B!=0). *** The sqlite3_uri_int64(F,P,D) routine converts the value of P into a ** 64-bit signed integer and returns that integer, or D if P does not ** exist. If the value of P is something other than an integer, then ** zero is returned. *** If F is a NULL pointer, then sqlite3_uri_parameter(F,P) returns NULL and ** sqlite3_uri_boolean(F,P,B) returns B. If F is not a NULL pointer and ** is not a database file pathname pointer that SQLite passed into the xOpen ** VFS method, then the behavior of this routine is undefined and probably ** undesirable.

79537. Add the rowid of the row to be deleted to the RowSet

79538. ** Determine if we are dealing with WinRT, which provides only a subset of ** the full Win32 API.

79539. Bitmask used by OP_Once

79540. ** Enable SQLITE_ENABLE_EXPLAIN_COMMENTS if SQLITE_DEBUG is turned on.

79541. ** Compose a tcl-readable representation of expression pExpr. Return a ** pointer to a buffer containing that representation. It is the ** responsibility of the caller to at some point free the buffer using ** sqlite3_free().

79542. If the database may grow as a result of this checkpoint, hint ** about the eventual size of the db file to the VFS layer.

79543. Pointer to doclist buffer

79544. Size of each free slot

79545. IMP: R-11148-40995

79546. Buffer containing previous term written

79547. 190

79548. IMP: R-63666-48755

79549. **comment:** ** CAPI3REF: Obtain Aggregate Function Context ** METHOD: sqlite3_context ** ** Implementations of aggregate SQL functions use this ** routine to allocate memory for storing their state. *** ^The first time the sqlite3_aggregate_context(C,N) routine is called ** for a particular aggregate function, SQLite ** allocates N of memory, zeroes out that memory, and returns a pointer ** to the new memory. ^On second and subsequent calls to ** sqlite3_aggregate_context() for the same aggregate function instance, ** the same buffer is returned. Sqlite3_aggregate_context() is normally ** called once for each invocation of the xStep callback and then one ** last time when the xFinal callback is invoked. ^When no rows match ** an aggregate query, the xStep() callback of the aggregate function ** implementation is never called and xFinal() is called exactly once. ** In those cases, sqlite3_aggregate_context() might be called for the ** first time from within xFinal().^ ** ** ^The sqlite3_aggregate_context(C,N) routine returns a NULL pointer ** when first called if N is less than or equal to zero or if a memory ** allocate error occurs. *** ^The amount of space allocated by sqlite3_aggregate_context(C,N) is ** determined by the N parameter on first successful call. Changing the ** value of N in subsequent call to sqlite3_aggregate_context() within ** the same aggregate function instance will not resize the memory ** allocation.^ Within the xFinal callback, it is customary to set ** N=0 in calls to sqlite3_aggregate_context(C,N) so that no ** pointless memory allocations occur. *** ^SQLite automatically frees the memory allocated by ** sqlite3_aggregate_context() when the aggregate query concludes. *** The first parameter must be a copy of the ** [sqlite3_context | SQL function context] that is the first parameter ** to the xStep or xFinal callback routine that implements the aggregate ** function. ** This routine must be called from the same thread in which ** the aggregate SQL function is running.

label: code-design

79550. ?'. Numeric SQL variables

79551. Used by vector == or <>

79552. Background thread, if any

79553. True if buffer zTerm must be copied

79554. 1050

79555. ** Return a 64-bit checksum for the FTS index entry specified by the ** arguments to this function.

79556. Index of column zName applies to

79557. No function of the specified name was found. Return 0.

79558. Index of next byte to write to output

79559. Reset a JsonEachCursor back to its original state. Free any memory ** held.

79560. Connection shared memory

79561. If any row exist in the result set, record that fact and abort.

79562. Methods above are valid for version 2

79563. OP_Column only used for typeof()

79564. Schema containing this index

79565. First cell to delete

79566. Pager object

79567. The table in the FROM clause of SELECT

79568. 176

79569. SQLITE_ENABLE_PREUPDATE_HOOK

79570. ** Lower the locking level on file descriptor pFile to eFileLock. eFileLock ** must be either NO_LOCK or SHARED_LOCK. ** ** If the locking level of the file descriptor is already at or below ** the requested locking level, this routine is a no-op.

79571. Position

79572. The virtual file system to use

79573. ** Add the changeset currently stored in buffer pData, size nData bytes, ** to changeset-group p.

79574. ** The sqlite3_mutex_alloc() routine allocates a new ** mutex and returns a pointer to it. If it returns NULL ** that means that a mutex could not be allocated.

79575. First lock to acquire or release

79576. Sector size

79577. Num. columns already in colset object

79578. ** The proxyLockingContext has the path and file structures for the remote ** and local proxy files in it

79579. Remove the sqlite3_vtab* from the aVTrans[] array, if applicable

79580. 1290

79581. ** Proxy locking is only available on MacOSX

79582. Address of rows visited counter

79583. Size of open blob, in bytes

79584. A call to sqlite3BtreeMoveto() is needed

79585. Beginning of keyword text in zKeyText[]

79586. multiselect_op ::= UNION

79587. Allocate new Fts5HashEntry and add it to the hash table.

79588. Generate a subroutine that outputs a single row of the result ** set. This subroutine first looks at the iUseFlag. If iUseFlag ** is less than or equal to zero, the subroutine is a no-op. If ** the processing calls for the query to abort, this subroutine ** increments the iAbortFlag memory location before returning in ** order to signal the caller to abort.

79589. Bias search to the high end

79590. ** Argument pIter is a changeset iterator that has been initialized, but ** not yet passed to sqlite3changeset_next(). This function applies the ** changeset to the main database attached to handle "db". The supplied ** conflict handler callback is invoked to resolve any conflicts encountered ** while applying the change.

79591. Read a page from the write-ahead log, if it is present.

79592. **comment:** This happens if a malloc() inside a call to sqlite3_column_text() or ** sqlite3_column_text16() failed.

label: code-design

79593. **comment:** This is an error condition that can result, for example, when a SELECT ** on the right-hand side of an INSERT contains more result columns than ** there are columns in the table on the left. The error will be caught ** and reported later. But we need to make sure enough memory is allocated ** to avoid other spurious errors in the meantime.

label: code-design

79594. LIKE range processing address

79595. Saved value of the db->flags
79596. Index in the parent table
79597. The left operand, and output
79598. xShmBarrier
79599. Condition (a) is true. Promote the newest segment on level ** iLvl to level iTst.
79600. same as TK_RSHIFT, synopsis: r[P3]=r[P2]>>r[P1]
79601. Update the schema version field in the destination database. This ** is to make sure that the schema-version really does change in ** the case where the source and destination databases have the ** same schema version.
79602. UNIXFILE_* flags
79603. If the YYNOERRORRECOVERY macro is defined, then do not attempt to ** do any kind of error recovery. Instead, simply invoke the syntax ** error routine and continue going as if nothing had happened. *** Applications can set this macro (for example inside %include) if ** they intend to abandon the parse upon the first syntax error seen.
79604. Current term
79605. ** Free the contents of the With object passed as the second argument.
79606. Address of the select-A coroutine
79607. Column of last value written
79608. COMPRESS
79609. expr ::= VARIABLE
79610. Integer primary key is autoincrement
79611. A column-list is terminated by either a 0x01 or 0x00 byte that is ** not part of a multi-byte varint.
79612. Number of characters in zName
79613. ** Unlock the database file to level eLock, which must be either NO_LOCK ** or SHARED_LOCK. Regardless of whether or not the call to xUnlock() ** succeeds, set the Pager.eLock variable to match the (attempted) new lock. *** Except, if Pager.eLock is set to UNKNOWN_LOCK when this function is ** called, do not modify it. See the comment above the #define of ** UNKNOWN_LOCK for an explanation of this.
79614. ** If compiling for a processor that lacks floating point support, ** substitute integer for floating-point
79615. OPTIMIZATION-IF-FALSE
79616. Each page cache is an instance of the following object. Every ** open database file (including each in-memory database and each ** temporary or transient database) has a single page cache which ** is an instance of this object. *** Pointers to structures of this type are cast and returned as ** opaque sqlite3_pcach* handles.
79617. SQLite scalar function context
79618. Obtain authorization to do a recursive query
79619. Register holding root page number for new objects
79620. Make pExpr point to the appropriate pAggInfo->aFunc[] entry
79621. Allocation type code
79622. ** Free memory that might be associated with a particular database ** connection. Calling sqlite3DbFree(D,X) for X==0 is a harmless no-op. ** The sqlite3DbFreeNN(D,X) version requires that X be non-NULL.
79623. ** Free the list of table objects passed as the first argument. The contents ** of the changed-rows hash tables are also deleted.
79624. Context for active pre-update callback
79625. ** Return the size of a memory allocation previously obtained from ** sqlite3Malloc() or sqlite3_malloc().
79626. Address register for select-A coroutine
79627. * The size of the buffer used by sqlite3_win32_write_debug().
79628. 7 END:
79629. **comment:** ** Playback the journal and thus restore the database file to ** the state it was in before we started making changes. *** The journal file format is as follows: *** (1) 8 byte prefix. A copy of aJournalMagic[]. ** (2) 4 byte big-endian integer which is the number of valid page records ** in the journal. If this value is 0xffffffff, then compute the ** number of page records from the journal size. ** (3) 4 byte big-endian integer which is the initial value for the ** sanity checksum. ** (4) 4 byte integer which is the number of pages to truncate the ** database to during a rollback. ** (5) 4 byte big-endian integer which is the sector size. The header ** is this many bytes in size. ** (6) 4 byte big-endian integer which is the page size. ** (7) zero padding out to the next sector size. ** (8) Zero or more pages instances, each as follows: ** + 4 byte page number. ** + pPager->pageSize bytes of data. ** + 4 byte checksum ** ** When we speak of the journal header, we mean the first 7 items above. ** Each entry in the journal is an instance of the 8th item. *** Call the value from the second bullet "nRec". nRec is the number of ** valid page entries in the journal. In most cases, you can compute the ** value of nRec from the size of the journal file. But if a power ** failure occurred while the journal was being written, it could be the ** case that the size of the journal file had already been increased but ** the extra entries had not yet made it safely to disk. In such a case, ** the value of nRec computed from the file size would be too large. For ** that reason, we always use the nRec value in the header. *** If the nRec value is 0xffffffff it means that nRec should be computed ** from the file size. This value is used when the user selects the ** no-sync option for the journal. A power failure could lead to corruption ** in this case. But for things like temporary table (which will be ** deleted when the power is restored) we don't care. *** If the file opened as the journal file is not a well-formed ** journal file then all pages up to the first corrupted page are rolled ** back (or no pages if the journal header is corrupted). The journal file ** is then deleted and SQLITE_OK returned, just as if no corruption had ** been encountered. ** ** If an I/O or malloc() error occurs, the journal-file is not deleted ** and an error code is returned. *** The isHot parameter indicates that we are trying to rollback a journal ** that might be a hot journal. Or, it could be that the journal is ** preserved because of JOURNALMODE_PERSIST or JOURNALMODE_TRUNCATE. ** If the journal really is hot, reset the pager cache prior rolling ** back any content. If the journal is merely persistent, no reset is ** needed.
label: code-design
79630. ** Apply colset pColset to expression node pExpr and all of its descendants.
79631. Functions used to query pager state and configuration.
79632. ** Macro IfNotOmitAV(x) returns (x) if SQLITE_OMIT_AUTOVACUUM is not ** defined, or 0 if it is. For example: *** bIncrVacuum = IfNotOmitAV(pBtShared->incrVacuum);
79633. Insert data into the table of this cursor
79634. OUT: Size of array *pazCol
79635. 137
79636. ** Add 1 to the reference count for page iPage. If this is the second ** reference to the page, add an error message to pCheck->zErrMsg. ** Return 1 if there are 2 or more references to the page and 0 if ** this is the first reference to the page. *** Also check that the page number is in bounds.
79637. 47
79638. Handle open on database file
79639. same as TK_RSHIFT, in1, in2, out3
79640. Check invariants on a PgHdr object
79641. Space to unpack a record
79642. In Firefox (circa 2017-02-08), xRoundup() is remapped to an internal ** implementation of malloc_good_size(), which must be called in debug ** mode and specifically when the DMD "Dark Matter Detector" is enabled ** or else a crash results. Hence, do not attempt to optimize out the ** following xRoundup() call.
79643. Any deferred constraint violations have now been resolved.
79644. WAL frames perhaps written, or maybe not
79645. xBegin
79646. Existing colset object
79647. ** This function is invoked once for each page that has already been ** written into the log file when a WAL transaction is rolled back. ** Parameter iPg is the page number of said page. The pCtx argument ** is actually a pointer to the Pager structure. *** If page iPg is present in the cache, and has no outstanding references, ** it is discarded. Otherwise, if there are one or more outstanding ** references, the page content is reloaded from the database. If the ** attempt to reload content from the database is required and fails, ** return an SQLite error code. Otherwise, SQLITE_OK.
79648. **comment:** If the aStatic[] array is not large enough, allocate a large array ** using sqlite3_malloc(). This approach could be improved upon.
label: code-design
79649. Bytes of leaf data in segment

79650. Checkpoint on this pager
79651. A Sub-term within the pOrWc
79652. ** Translate the P4.pExpr value for an OP_CursorHint opcode into text ** that can be displayed in the P4 column of EXPLAIN output.
79653. Sync all the db files involved in the transaction. The same call ** sets the master journal pointer in each individual journal. If ** an error occurs here, do not delete the master journal file. *** If the error occurs during the first call to ** sqlite3BtreeCommitPhaseOne(), then there is a chance that the ** master journal file will be orphaned. But we cannot delete it, ** in case the master journal file name was written into the journal ** file before the failure occurred.
79654. Number of phrase instances
79655. **comment:** ** Append the output of a printf() style formatting to an existing string.
label: code-design
79656. Must have at least one page for the WAL commit flag. ** Ticket [2d1a5c67dfc2363e44f29d9bb57f] 2011-05-18
79657. Bytes required to hold table name
79658. The ORDER BY clause. May be null
79659. Code the OP_Program opcode in the parent Vdbe. P4 of the OP_Program ** is a pointer to the sub-vdbe containing the trigger program.
79660. Keep track of the maximum allocation request. Even unfulfilled ** requests are counted
79661. 117
79662. xLock method
79663. ** Allocate and zero memory. If the allocation fails, make ** the mallocFailed flag in the connection pointer.
79664. FTS5 global object for this database
79665. ** Return the PRIMARY KEY index of a table
79666. **comment:** Check to see if another process is holding the dead-man switch. ** If not, truncate the file to zero length.
label: code-design
79667. This routine is a no-op if the shared-cache is not enabled
79668. Allocate a buffer large enough for an Fts3Cursor structure. If the ** allocation succeeds, zero it and return SQLITE_OK. Otherwise, ** if the allocation fails, return SQLITE_NOMEM.
79669. szScratch
79670. The pragma
79671. If the journal needs to be sync()'ed before page pPg->pgno can ** be written to, store pPg->pgno in local variable needSyncPgno. ** ** If the isCommit flag is set, there is no need to remember that ** the journal needs to be sync()'ed before database page pPg->pgno ** can be written to. The caller has already promised not to write to it.
79672. ** Properties of opcodes. The OPFLG_INITIALIZER macro is ** created by mkopcodeh.awk during compilation. Data is obtained ** from the comments following the "case OP_XXXX;" statements in ** the vdbe.c file.
79673. **comment:** There are no SHIFTREDUCE actions on nonterminals because the table ** generator has simplified them to pure REDUCE actions.
label: code-design
79674. Memory cell hold array of subprogs
79675. "SELECT * FROM %_segdir WHERE level BETWEEN ? AND ? ORDER BY ..."
79676. Cursor number of the sorting index
79677. End loop over indexes
79678. Usable size of mapping at pMapRegion
79679. Root of current AND/NEAR cluster
79680. Copy the overflow cells from pRoot to pChild
79681. ** Attach subtrees pLeft and pRight to the Expr node pRoot. ** ** If pRoot==NULL that means that a memory allocation error has occurred. ** In that case, delete the subtrees pLeft and pRight.
79682. 3 integers per phrase instance
79683. ** The code in this file only exists if we are not omitting the ** ALTER TABLE logic from the build.
79684. Size of the open journal file in bytes
79685. Name of the database containing pTable
79686. Spacer
79687. Table cursor number
79688. **comment:** At one point the code here called assertTruncateConstraint() to ** ensure that all pages being truncated away by this operation are, ** if one or more savepoints are open, present in the savepoint ** journal so that they can be restored if the savepoint is rolled ** back. This is no longer necessary as this function is now only ** called right before committing a transaction. So although the ** Pager object may still have open savepoints (Pager.nSavepoint!=0), ** they cannot be rolled back. So the assertTruncateConstraint() call ** is no longer correct.
label: code-design
79689. Close with last statement close
79690. No more retries.
79691. ** Enter a mutex on the given BTree object. ** ** If the object is not sharable, then no mutex is ever required ** and this routine is a no-op. The underlying mutex is non-recursive. ** But we keep a reference count in Btree.wantToLock so the behavior ** of this interface is recursive. ** ** To avoid deadlocks, multiple Btrees are locked in the same order ** by all database connections. The p->pNext is a list of other ** Btrees belonging to the same database connection as the p Btree ** which need to be locked after p. If we cannot get a lock on ** p, then first unlock all of the others on p->pNext, then wait ** for the lock to become available on p, then relock all of the ** subsequent Btrees that desire a lock.
79692. The 0x7ffff00 limit term is explained in comments on sqlite3Malloc()
79693. Frame corresponding to aPgno[0]
79694. inProgress
79695. TABLE.COLUMN if no AS clause and is a direct table ref
79696. !defined(SQLITE_OMITDECLTYPE)
79697. Extract the values to be compared.
79698. Pointer to tokenizer hash table
79699. If this process is running as root and if creating a new rollback ** journal or WAL file, set the ownership of the journal or WAL to be ** the same as the original database.
79700. ** 2004 May 22 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code that is specific to Windows.
79701. Bytes at zTerm
79702. Column number of X in "X <op> <expr>"
79703. ** Return a pointer to the column affinity string associated with index ** pIdx. A column affinity string has one character for each column in ** the table, according to the affinity of the column: ** ** Character Column affinity ** ----- ** 'A' BLOB ** 'B' TEXT ** 'C' NUMERIC ** 'D' INTEGER ** 'F' REAL ** ** An extra 'D' is appended to the end of the string to cover the ** rowid that appears as the last column in every index. ** ** Memory for the buffer containing the column index affinity string ** is managed along with the rest of the Index structure. It will be ** released when sqlite3DeleteIndex() is called.
79704. Number of bytes required on leaf page
79705. Cursor number from which to take column data
79706. Write the data read from the journal back into the database file. ** This is usually safe even for an encrypted database - as the data ** was encrypted before it was written to the journal file. The exception ** is if the data was just read from an in-memory sub-journal. In that ** case it must be encrypted here before it is copied into the database ** file.
79707. Journaled and ready to modify
79708. ** Free slots in the allocator used to divide up the global page cache ** buffer provided using the SQLITE_CONFIG_PAGECACHE mechanism.
79709. ***** Continuing where we left off in vdbe.h *****
79710. Database the cursor belongs to, or -1

79711. ** Print a WhereLoop object for debugging purposes
79712. Output variables. zTbl==0 implies EOF.
79713. Registers holding results of a co-routine
79714. The two values have the same sign. Compare using memcmp().
79715. Force exponential buffer size growth as long as it does not overflow, ** to avoid having to call this routine too often
79716. ** Gobble up the first bareword or quoted word from the input buffer zIn. ** Return a pointer to the character immediately following the last in ** the gobbled word if successful, or a NULL pointer otherwise (failed ** to find close-quote character). *** Before returning, set pzOut to point to a new buffer containing a ** nul-terminated, dequoted copy of the gobbled word. If the word was ** quoted, *pbQuoted is also set to 1 before returning. *** If *pRc is other than SQLITE_OK when this function is called, it is ** a no-op (NULL is returned). Otherwise, if an OOM occurs within this ** function, *pRc is set to SQLITE_NOMEM before returning. *pRc is *not* ** set if a parse error (failed to find close quote) occurs.
79717. WHEN
79718. pDestIdx has no corresponding index in pSrc
79719. Implementation of the following: *** Rewind csr ** if eof(csr) goto end_of_scan; ** regChng = 0 ** goto next_push_0; **
79720. FT5 configuration
79721. ** This function outputs the specified (ANSI) string to the Win32 debugger ** (if available).
79722. sqlite3_config() shall return SQLITE_MISUSE if it is invoked while ** the SQLite library is in use.
79723. 142
79724. 400
79725. Round towards zero
79726. 0x09
79727. Populate the argument registers.
79728. DOUB
79729. Collating sequence to on a column
79730. And to delete index entries
79731. Column of index to match
79732. Next in the link-list
79733. If p1 is at EOF
79734. File-name for the master journal
79735. ** CAPI3REF: Declared Datatype Of A Query Result ** METHOD: sqlite3_stmt *** ^The first parameter is a [prepared statement]. ** If this statement is a [SELECT] statement and the Nth column of the ** returned result set of that [SELECT] is a table column (not an ** expression or subquery) then the declared type of the table ** column is returned.)^ If the Nth column of the result set is an ** expression or subquery, then a NULL pointer is returned. ** ^The returned string is always UTF-8 encoded. *** ^For example, given the database schema: *** CREATE TABLE t1(c1 VARIANT); *** and the following statement to be compiled: *** SELECT c1 + 1, c1 FROM t1; ** this routine would return the string "VARIANT" for the second result ** column (i==1), and a NULL pointer for the first result column (i==0).)^ ** ^SQLite uses dynamic run-time typing. ^So just because a column ** is declared to contain a particular type does not mean that the ** data stored in that column is of the declared type. SQLite is ** strongly typed, but the typing is dynamic not static. ^Type ** is associated with individual values, not with the containers ** used to hold those values.
79736. sync directory after journal delete
79737. **comment:** True in "cleanup" state
 label: code-design
79738. Number of '?' variables seen in the SQL so far
79739. ** Remove all data from the FT53 table. Clear the hash table containing ** pending terms.
79740. turn off proxy locking - already off - NOOP
79741. IN/OUT: Right/output doclist
79742. True if errors are benign
79743. ***** Begin file threads.c *****
79744. ** This function is used to resize the hash table used by the cache passed ** as the first argument. *** The PCache mutex must be held when this function is called.
79745. **comment:** ** Move an existing blob handle to point to a different row of the same ** database table. *** If an error occurs, or if the specified row does not exist or does not ** contain a blob or text value, then an error code is returned and the ** database handle error code and message set. If this happens, then all ** subsequent calls to sqlite3_blob_xxx() functions (except blob_close()) ** immediately return SQLITE_ABORT.
 label: code-design
79746. Opcode: SetCookie P1 P2 P3 *** Write the integer value P3 into cookie number P2 of database P1. ** P2==1 is the schema version. P2==2 is the database format. ** P2==3 is the recommended pager cache ** size, and so forth. P1==0 is the main database file and P1==1 is the ** database file used to store temporary tables. *** A transaction must be started before executing this opcode.
79747. OUT: Tokenizer (if applicable)
79748. Tokens to the left of first highlight
79749. Readr data from the file. Return early if an error occurs.
79750. Extra space on the end of pIdx
79751. Name of WAL file
79752. Bytes of space to allocate here
79753. When sqlite_stat3 histogram data is available an operator of the ** form "x IS NOT NULL" can sometimes be evaluated more efficiently ** as "x>NULL" if x is not an INTEGER PRIMARY KEY. So construct a ** virtual term of that form. *** Note that the virtual term must be tagged with TERM_VNULL.
79754. 7x
79755. Skip docid varint
79756. ** CAPI3REF: Start a read transaction on an historical snapshot ** EXPERIMENTAL *** ^The [sqlite3_snapshot_open(D,S,P)] interface starts a ** read transaction for schema S of ** [database connection] D such that the read transaction ** refers to historical [snapshot] P, rather than the most ** recent change to the database. ** ^The [sqlite3_snapshot_open()] interface returns SQLITE_OK on success ** or an appropriate [error code] if it fails. *** ^In order to succeed, a call to [sqlite3_snapshot_open(D,S,P)] must be ** the first operation following the [BEGIN] that takes the schema S ** out of [autocommit mode]. ** ^In other words, schema S must not currently be in ** a transaction for [sqlite3_snapshot_open(D,S,P)] to work, but the ** database connection D must be out of [autocommit mode]. ** ^A [snapshot] will fail to open if it has been overwritten by a ** [checkpoint]. ** ^A call to [sqlite3_snapshot_open(D,S,P)] will fail if the ** database connection D does not know that the database file for ** schema S is in [WAL mode]. A database connection might not know ** that the database file is in [WAL mode] if there has been no prior ** I/O on that database connection, or if the database entered [WAL mode] ** after the most recent I/O on the database connection.)*** (Hint: Run "[PRAGMA application_id]" against a newly opened ** database connection in order to make it ready to use snapshots.) *** The [sqlite3_snapshot_open()] interface is only available when the ** SQLITE_ENABLE_SNAPSHOT compile-time option is used.
79757. SQLITE_STMTSTATUS_SORT
79758. Write the referenced table cursor & column here
79759. Combined MEM_* flags from both inputs
79760. The overall R-Tree
79761. same as TK_NE, jump, in1, in3
79762. expr ::= ID|INDEXED LP STAR RP
79763. Either "ORDER" or "GROUP", as appropriate
79764. ** Free a page object allocated by pcache1AllocPage().
79765. Desired snippet length
79766. If no pending lock has been acquired, then acquire it
79767. Write the comparison result here
79768. zName:
79769. Read the PK of the current row into an array of registers. In ** ONEPASS_OFF mode, serialize the array into a record and store it in ** the ephemeral table. Or, in ONEPASS_SINGLE or MULTI mode, change ** the OP_OpenEphemeral instruction to a Noop (the ephemeral table ** is not required) and leave the PK fields in

the array of registers.

79770. Highlight-mask for snippet

79771. The remaining transformations only apply to b-tree tables, not to ** virtual tables

79772. 3 registers to hold a result row

79773. Start a search on a new JSON string

79774. ** This is a wrapper around "sqlite3_mprintf(zFmt, ...)". If an OOM occurs, ** an error code is stored in the RBU handle passed as the first argument. *** If an error has already occurred (p->rc is already set to something other ** than SQLITE_OK), then this function returns NULL without modifying the ** stored error code. In this case it still calls sqlite3_free() on any ** printf() parameters associated with %z conversions.

79775. Store SELECT results in intermediate table

79776. 1370

79777. **comment:** Do not search for an unused file descriptor on vxworks. Not because ** vxworks would not benefit from the change (it might, we're not sure), ** but because no way to test it is currently available. It is better ** not to risk breaking vxworks support for the sake of such an obscure ** feature.

label: code-design

79778. OUT: Pointer to output buffer

79779. The function name

79780. ** Set the Expr.nHeight variable in the structure passed as an ** argument. An expression with no children, Expr.pList or ** Expr.pSelect member has a height of 1. Any other expression ** has a height equal to the maximum height of any other ** referenced Expr plus one. *** Also propagate EP_Propagate flags up from Expr.x.pList to Expr.flags, ** if appropriate.

79781. Get low-level information about the file that we can used to ** create a unique name for the file.

79782. If this statement has violated immediate foreign key constraints, do ** not return the number of rows modified. And do not RELEASE the statement ** transaction. It needs to be rolled back.

79783. Array of parsed cells

79784. restriction 5

79785. ** Turn a relative pathname into a full pathname. Write the full ** pathname into zOut[]. zOut[] will be at least pVfs->mxPathname ** bytes in size.

79786. Parent VFS

79787. if !defined(SQLITE_ENABLE_COLUMN_METADATA)

79788. ** Detect compound SELECT statements that use an ORDER BY clause with ** an alternative collating sequence. *** SELECT ... FROM t1 EXCEPT
SELECT ... FROM t2 ORDER BY .. COLLATE ... *** These are rewritten as a subquery: *** SELECT * FROM (SELECT ... FROM t1 EXCEPT SELECT ...
FROM t2) ** ORDER BY ... COLLATE ... *** This transformation is necessary because the multiSelectOrderBy() routine ** above that generates the code for a
compound SELECT with an ORDER BY clause ** uses a merge algorithm that requires the same collating sequence on the ** result columns as on the ORDER
BY clause. See ticket ** http://www.sqlite.org/src/info/6709574d2a *** This transformation is only needed for EXCEPT, INTERSECT, and UNION. ** The
UNION ALL operator works fine with multiSelectOrderBy() even when ** there are COLLATE terms in the ORDER BY.

79789. If there are any triggers to fire, allocate a range of registers to ** use for the old.* references in the triggers.

79790. Pointer to token

79791. Table list this loop refers to

79792. #include "sqlite3.h" ** Required for error code definitions **

79793. 119

79794. Like mgt0 above except we are looking for a value of m>1 instead ** or m>0

79795. ** CAPI3REF: Concatenate Two Changeset Objects *** This function is used to concatenate two changesets, A and B, into a ** single changeset. The result is a
changeset equivalent to applying ** changeset A followed by changeset B. *** This function combines the two input changesets using an **
sqlite3_changegroup object. Calling it produces similar results as the ** following code fragment: *** sqlite3_changegroup *pGrp; ** rc =
sqlite3_changegroup_new(&pGrp); ** if(rc==SQLITE_OK) rc = sqlite3changegroup_add(pGrp, nA, pA); ** if(rc==SQLITE_OK) rc =
sqlite3changegroup_add(pGrp, nB, pB); ** if(rc==SQLITE_OK){ ** rc = sqlite3changegroup_output(pGrp, pnOut, ppOut); ** }else{ *** *ppOut = 0; *** *pnOut
= 0; *** } *** Refer to the sqlite3_changegroup documentation below for details.

79796. Statement to return all language-ids

79797. ** Load content from the sqlite_stat4 and sqlite_stat3 tables into ** the Index.aSample[] arrays of all indices.

79798. Delete any TriggerPrg structures allocated while parsing this statement.

79799. Table with INTEGER PRIMARY KEY and nothing else

79800. ** Invoke either the xSavepoint, xRollbackTo or xRelease method of all ** virtual tables that currently have an open transaction. Pass iSavepoint ** as the second
argument to the virtual table method invoked. *** If op is SAVEPOINT_BEGIN, the xSavepoint method is invoked. If it is ** SAVEPOINT_ROLLBACK, the
xRollbackTo method. Otherwise, if op is ** SAVEPOINT_RELEASE, then the xRelease method of each virtual table with ** an open transaction is invoked. **
** If any virtual table method returns an error code other than SQLITE_OK, ** processing is abandoned and the error returned to the caller of this ** function
immediately. If all calls to virtual table methods are successful, ** SQLITE_OK is returned.

79801. Pointer to interior tree structure

79802. f8..ff

79803. The LIMIT clause. May be null

79804. Opcode to access the value of the IN constraint

79805. New rowid value (for a rowid UPDATE)

79806. SQL_SELECT_SEGDIR ** Read a single entry from the %_segdir table. The entry from absolute ** level :1 with index value .2.

79807. True to auto-attach tables

79808. xRename

79809. ** Terminate the current execution of an SQL statement and reset it ** back to its starting state so that it can be reused. A success code from ** the prior execution
is returned. *** This routine sets the error code and string returned by ** sqlite3_errcode(), sqlite3errmsg() and sqlite3errmsg16().

79810. Do FK processing. This call checks that any FK constraints that ** refer to this table (i.e. constraints attached to other tables) ** are not violated by deleting this
row.

79811. szPage

79812. Adjust pTemplate cost downward so that it is cheaper than its ** subset p.

79813. If this is a converted compound query, move the ORDER BY clause from ** the sub-query back to the parent query. At this point each term ** within the ORDER
BY clause has been transformed to an integer value. ** These integers will be replaced by copies of the corresponding result ** set expressions by the call to
resolveOrderGroupBy() below.

79814. ** 2005-07-08 *** The author claims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This file contains code associated with the ANALYZE command.

*** The ANALYZE command gather statistics about the content of tables ** and indices. These statistics are made available to the query planner ** to help it
make better decisions about how to perform queries. *** The following system tables are or have been supported: *** CREATE TABLE sqlite_stat1(tbl, idx,
stat); ** CREATE TABLE sqlite_stat2(tbl, idx, samplen, sample); ** CREATE TABLE sqlite_stat3(tbl, idx, nEq, nLt, nDLt, sample); ** CREATE TABLE
sqlite_stat4(tbl, idx, nEq, nLt, nDLt, sample); *** Additional tables might be added in future releases of SQLite. ** The sqlite_stat2 table is not created or used
unless the SQLite version ** is between 3.6.18 and 3.7.8, inclusive, and unless SQLite is compiled ** with SQLITE_ENABLE_STAT2. The sqlite_stat2 table is
deprecated. ** The sqlite_stat2 table is superseded by sqlite_stat3, which is only ** created and used by SQLite versions 3.7.9 and later and with **

SQLITE_ENABLE_STAT3 defined. The functionality of sqlite_stat3 ** is a superset of sqlite_stat2. The sqlite_stat4 is an enhanced ** version of sqlite_stat3 and
is only available when compiled with ** SQLITE_ENABLE_STAT4 and in SQLite versions 3.8.1 and later. It is ** not possible to enable both STAT3 and STAT4
at the same time. If they ** are both enabled, then STAT4 takes precedence. *** For most applications, sqlite_stat1 provides all the statistics required ** for the
query planner to make good choices. *** Format of sqlite_stat1: *** There is normally one row per index, with the index identified by the ** name in the idx
column. The tbl column is the name of the table to ** which the index belongs. In each such row, the stat column will be ** a string consisting of a list of integers.
The first integer in this ** list is the number of rows in the index. (This is the same as the ** number of rows in the table, except for partial indices.) The second **
integer is the average number of rows in the index that have the same ** value in the first column of the index. The third integer is the average ** number of rows
in the index that have the same value for the first two ** columns. The N-th integer (for N>1) is the average number of rows in ** the index which have the same
value for the first N-1 columns. For ** a K-column index, there will be K+1 integers in the stat column. If ** the index is unique, then the last integer will be 1. **

** The list of integers in the stat column can optionally be followed ** by the keyword "unordered". The "unordered" keyword, if it is present, ** must be separated from the last integer by a single space. If the ** "unordered" keyword is present, then the query planner assumes that ** the index is unordered and will not use the index for a range query. *** If the sqlite_stat1.idx column is NULL, then the sqlite_stat1.stat ** column contains a single integer which is the (estimated) number of ** rows in the table identified by sqlite_stat1.tbl. *** Format of sqlite_stat2: *** The sqlite_stat2 is only created and is only used if SQLite is compiled ** with SQLITE_ENABLE_STAT2 and if the SQLite version number is between ** 3.6.18 and 3.7.8. The "stat2" table contains additional information ** about the distribution of keys within an index. The index is identified by ** the "idx" column and the "tbl" column is the name of the table to which ** the index belongs. There are usually 10 rows in the sqlite_stat2 ** table for each index. *** The sqlite_stat2 entries for an index that have samples between 0 and 9 ** inclusive are samples of the left-most key value in the index taken at ** evenly spaced points along the index. Let the number of samples be S ** (10 in the standard build) and let C be the number of rows in the index. ** Then the sampled rows are given by: *** rownumber = (i*C*2 + C)/(S*2) *** For i between 0 and S-1. Conceptually, the index space is divided into ** S uniform buckets and the samples are the middle row from each bucket. *** The format for sqlite_stat2 is recorded here for legacy reference. This ** version of SQLite does not support sqlite_stat2. It neither reads nor ** writes the sqlite_stat2 table. This version of SQLite only supports ** sqlite_stat3. *** Format for sqlite_stat3: *** The sqlite_stat3 format is a subset of sqlite_stat4. Hence, the ** sqlite_stat4 format will be described first. Further information ** about sqlite_stat3 follows the sqlite_stat4 description. *** Format for sqlite_stat4: *** As with sqlite_stat2, the sqlite_stat4 table contains histogram data ** to aid the query planner in choosing good indices based on the values ** that indexed columns are compared against in the WHERE clauses of ** queries. *** The sqlite_stat4 table contains multiple entries for each index. ** The idx column names the index and the tbl column is the table of the ** index. If the idx and tbl columns are the same, then the sample is ** of the INTEGER PRIMARY KEY. The sample column is a blob which is the ** binary encoding of a key from the index. The nEq column is a ** list of integers. The first integer is the approximate number ** of entries in the index whose left-most column exactly matches ** the left-most column of the sample. The second integer in nEq ** is the approximate number of entries in the index where the ** first two columns match the first two columns of the sample. ** And so forth. nLt is another list of integers that show the approximate ** number of entries that are strictly less than the sample. The first ** integer in nLt contains the number of entries in the index where the ** left-most column is less than the left-most column of the sample. ** The K-th integer in the nLt entry is the number of index entries ** where the first K columns are less than the first K columns of the ** sample. The nDLt column is like nLt except that it contains the ** number of distinct entries in the index that are less than the ** sample. *** There can be an arbitrary number of sqlite_stat4 entries per index. ** The ANALYZE command will typically generate sqlite_stat4 tables ** that contain between 10 and 40 samples which are distributed across ** the key space, though not uniformly, and which include samples with ** large nEq values. *** Format for sqlite_stat3 redux: *** The sqlite_stat3 table is like sqlite_stat4 except that it only ** looks at the left-most column of the index. The sqlite_stat3.sample ** column contains the actual value of the left-most column instead ** of a blob encoding of the complete index key as is found in ** sqlite_stat4.sample. The nEq, nLt, and nDLt entries of sqlite_stat3 ** all contain just a single integer which is the same as the first ** integer in the equivalent columns in sqlite_stat4.

79815. Minimum PMA size, in bytes

79816. ** Append a string to the string-buffer passed as the first argument. *** If nAppend is negative, then the length of the string zAppend is ** determined using strlen().

79817. Tracing routines for the RtreeSearchPoint queue

79818. Turn bulk memory into a hash table object by initializing the ** fields of the Hash structure. *** "pNew" is a pointer to the hash table that is to be initialized. ** keyClass is one of the constants ** FTS3_HASH_BINARY or FTS3_HASH_STRING. The value of keyClass ** determines what kind of key the hash table will use. "copyKey" is ** true if the hash table should make its own private copy of keys and ** false if it should just use the supplied pointer.

79819. Version 3.7.16 and later

79820. ** Delete an entire expression list.

79821. expr ::= NOT expr

79822. ** The input to this routine is an WhereTerm structure with only the ** "pExpr" field filled in. The job of this routine is to analyze the ** subexpression and populate all the other fields of the WhereTerm ** structure. *** If the expression is of the form "<expr> <op> X" it gets commuted ** to the standard form of "X <op> <expr>". *** If the expression is of the form "X <op> Y" where both X and Y are ** columns, then the original expression is unchanged and a new virtual ** term of the form "Y <op> X" is added to the WHERE clause and ** analyzed separately. The original term is marked with TERM_COPIED ** and the new term is marked with TERM_DYNAMIC (because it's pExpr ** needs to be freed with the WhereClause) and TERM_VIRTUAL (because it ** is a commuted copy of a prior term.) The original term has nChild=1 ** and the copy has idxParent set to the index of the original term.

79823. The transaction counter has changed

79824. Array of column names

79825. ** A structure defining how to do GLOB-style comparisons.

79826. ** Move the cursor down to the right-most leaf entry beneath the ** page to which it is currently pointing. Notice the difference ** between moveToLeftmost() and moveToRightmost(). moveToLeftmost() ** finds the left-most entry beneath the *entry* whereas moveToRightmost() ** finds the right-most entry beneath the *page*. *** The right-most entry is the one with the largest key - the last ** key in ascending order.

79827. New temp space

79828. Write the MergeEngine here

79829. ** Transfer content from the second pLoop into the first.

79830. **comment:** One of the FTSQUERY_XXX values defined below

label: code-design

79831. The IN, SELECT, or EXISTS operator

79832. Number of bytes in buffer zTerm

79833. Merge-writer object

79834. Loop termination flag

79835. Input: Mask of columns used by statement

79836. Size of input list

79837. ** Attempt to apply the change that the iterator passed as the first argument ** currently points to to the database. If a conflict is encountered, invoke ** the conflict handler callback. *** The difference between this function and sessionApplyOne() is that this ** function handles the case where the conflict-handler is invoked and ** returns SQLITE_CHANGESET_REPLACE - indicating that the change should be ** retried in some manner.

79838. True if this constraint is on docid

79839. Value configured by 'automerge'

79840. Prepared statement for which info desired

79841. True to disable triggers

79842. Database page size

79843. True to dequote pCollName

79844. ** CAPI3REF: Generate A Patchset From A Session Object ** ** The differences between a patchset and a changeset are that: *** ** DELETE records consist of the primary key fields only. The ** original values of other fields are omitted. ** The original values of any modified fields are omitted from ** UPDATE records. ** ** ** A patchset blob may be used with up to date versions of all ** sqlite3changeset_xxx API functions except for sqlite3changeset_invert(), ** which returns SQLITE_CORRUPT if it is passed a patchset. Similarly, ** attempting to use a patchset blob with old versions of the ** sqlite3changeset_xxx APIs also provokes an SQLITE_CORRUPT error. *** Because the non-primary key "old."** fields are omitted, no ** SQLITE_CHANGESET_DATA conflicts can be detected or reported if a patchset ** is passed to the sqlite3changeset_apply() API. Other conflict types work ** in the same way as for changesets. ** ** Changes within a patchset are ordered in the same way as for changesets ** generated by the sqlite3session_changeset() function (i.e. all changes for ** a single table are grouped together, tables appear in the order in which ** they were attached to the session object).

79845. Stat4Accum.anEq

79846. Boolean array - true for PK columns

79847. ** SQLite will invoke this method one or more times while planning a query ** that uses the stmt virtual table. This routine needs to create ** a query plan for each invocation and compute an estimated cost for that ** plan.

79848. moved CSE1 up

79849. Used by: foreign_key_check

79850. for AFP simulated shared lock

79851. If a NULL pointer was passed as the collate function, fall through ** to the blob case and use memcmp().

79852. ** Advance the iterator pDL to the next entry in pDL->aAll/nAll. Set *pbEof ** to true if EOF is reached.

79853. Determine the type of a table. *** peType is of type (int*), a pointer to an output parameter of type ** (int). This call sets the output parameter as follows, depending ** on the type of the table specified by parameters dbName and zTbl. *** RBU_PK_NOTABLE: No such table. *** RBU_PK_NONE: Table has an implicit rowid. *** RBU_PK_IPK: Table has an explicit IPK column. *** RBU_PK_EXTERNAL: Table has an external PK index. *** RBU_PK_WITHOUT_ROWID: Table is WITHOUT ROWID. *** RBU_PK_VTAB: Table is a virtual table. *** Argument *piPk is also of type (int*), and also points to an output ** parameter. Unless the table has an external primary key index ** (i.e. unless *peType is set to 3), then *piPk is set to zero. Or, ** if the table does have an external primary key index, then *piPk ** is set to the root page number of the primary key index before ** returning. *** ALGORITHM: *** *** if(no entry exists in sqlite_master){ ** return RBU_PK_NOTABLE ** } else if(sql for the entry starts with "CREATE VIRTUAL"){ ** return RBU_PK_VTAB ** } else if("PRAGMA index_list()" for the table contains a "pk" index){ ** if(the index that is the pk exists in sqlite_master) { *** *piPK = rootpage of that index. *** return RBU_PK_EXTERNAL ** } else{ *** return RBU_PK_WITHOUT_ROWID ** } *** } else if("PRAGMA table_info()" lists one or more "pk" columns){ *** return RBU_PK_IPK ** } else{ *** return RBU_PK_NONE ** } }

79854. Length of TOP vector

79855. ** Open a file descriptor to the directory containing file zFilename. ** If successful, *pFd is set to the opened file descriptor and ** SQLITE_OK is returned. If an error occurs, either SQLITE_NOMEM ** or SQLITE_CANTOPEN is returned and *pFd is set to an undefined ** value. *** ** The directory file descriptor is used for only one thing - to ** fsync() a directory to make sure file creation and deletion events ** are flushed to disk. Such fsyncs are not needed on newer ** journaling filesystems, but are required on older filesystems. *** ** This routine can be overridden using the xSetSysCall interface. ** The ability to override this routine was added in support of the ** chromium sandbox. Opening a directory is a security risk (we are ** told) so making it overrideable allows the chromium sandbox to ** replace this routine with a harmless no-op. To make this routine ** a no-op, replace it with a stub that returns SQLITE_OK but leaves ** *pFd set to a negative number. *** ** If SQLITE_OK is returned, the caller is responsible for closing ** the file descriptor *pFd using close().

79856. Argument to the progress callback

79857. EVIDENCE-OF: R-06866-39125 Freeblocks are always connected in order of ** increasing offset.

79858. expr ::= expr BITAND|BITOR|LSHIFT|RSHIFT expr

79859. ** Register a callback to be invoked each time a row is updated, ** inserted or deleted using this database connection.

79860. Thread that will run pMerger

79861. Change to compare to

79862. **comment:** 0x2000 not currently used
label: code-design

79863. Local cache of vfs pointer

79864. The table whose column is desired

79865. ** CAPI3REF: Virtual Table Constraint Operator Codes *** ** These macros defined the allowed values for the ** [sqlite3_index_info].aConstraint[].op field. Each value represents ** an operator that is part of a constraint term in the WHERE clause of ** a query that uses a [virtual table].

79866. ** This is the xSetSystemCall() method of sqlite3_vfs for all of the ** "win32" VFSes. Return SQLITE_OK upon successfully updating the ** system call pointer, or SQLITE_NOTFOUND if there is no configurable ** system call named zName.

79867. No additional checks needed for this table

79868. Starting offset in main journal

79869. Locate the pragma in the lookup table

79870. Any terms specified as part of the ON(...) clause for any LEFT ** JOIN for which the current table is not the rhs are omitted ** from the cursor-hint. *** ** If this table is the rhs of a LEFT JOIN, "IS" or "IS NULL" terms ** that were specified as part of the WHERE clause must be excluded. ** This is to address the following: ** ** SELECT ... t1 LEFT JOIN t2 ON (t1.a=t2.b) WHERE t2.c IS NULL; ** ** Say there is a single row in t2 that matches (t1.a=t2.b), but its ** t2.c values is not NULL. If the (t2.c IS NULL) constraint is ** pushed down to the cursor, this row is filtered out, causing ** SQLite to synthesize a row of NULL values. Which does match the ** WHERE clause, and so the query returns a row. Which is incorrect. *** ** For the same reason, WHERE terms such as: *** ** WHERE 1 = (t2.c IS NULL) *** are also excluded. See codeCursorHintIsOrFunction() for details.

79871. Space used in zNewRecord[] content

79872. ** 2003 April 6 *** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code used to implement the ATTACH and DETACH commands.

79873. ** Remove node pNode from the node hash table.

79874. Value passed to FCNTL_SIZE_HINT call

79875. ** One or more segments have just been removed from absolute level iAbsLevel. ** Update the 'idx' values of the remaining segments in the level so that ** the idx values are a contiguous sequence starting from 0.

79876. ** Carve a piece off of the end of the mem3.iMaster free chunk. ** Return a pointer to the new allocation. Or, if the master chunk ** is not large enough, return 0.

79877. Verify condition (1): If cells are moving left, update iPg ** only after iPg-1 has already been updated.

79878. ** Initialize the object pItr to point to term pTerm/nTerm within segment ** pSeg. If there is no such term in the index, the iterator is set to EOF. *** ** If an error occurs, Fts5Index.rc is set to an appropriate error code. If ** an error has already occurred when this function is called, it is a no-op.

79879. 0: off 1: simple 2: cache dumps

79880. Page into which we are copying

79881. ** Query status information for a single database connection

79882. List of all WhereLoop objects

79883. ** An instance of the following structure serves as the key used ** to locate a particular unixInodeInfo object.

79884. ** Add all WhereLoop objects for a table of the join identified by ** pBuilder->pNew->iTab. That table is guaranteed to be a virtual table. *** ** If there are no LEFT or CROSS JOIN joins in the query, both mPrereq and ** mUnusable are set to 0. Otherwise, mPrereq is a mask of all FROM clause ** entries that occur before the virtual table in the FROM clause and are ** separated from it by at least one LEFT or CROSS JOIN. Similarly, the ** mUnusable mask contains all FROM clause entries that occur after the ** virtual table and are separated from it by at least one LEFT or ** CROSS JOIN. *** ** For example, if the query were: *** ... FROM t1, t2 LEFT JOIN t3, t4, vt CROSS JOIN t5, t6; ** ** then mPrereq corresponds to (t1, t2) and mUnusable to (t5, t6). *** ** All the tables in mPrereq must be scanned before the current virtual ** table. So any terms for which all prerequisites are satisfied by ** mPrereq may be specified as "usable" in all calls to xBestIndex. ** Conversely, all tables in mUnusable must be scanned after the current ** virtual table, so any terms for which the prerequisites overlap with ** mUnusable should always be configured as "not-usable" for xBestIndex.

79885. API oddity: If successful, SetFilePointer() returns a dword ** containing the lower 32-bits of the new file-offset. Or, if it fails, ** it returns INVALID_SET_FILE_POINTER. However according to MSDN, ** INVALID_SET_FILE_POINTER may also be a valid new offset. So to determine ** whether an error has actually occurred, it is also necessary to call ** GetLastError().

79886. True if a leaf page

79887. ** CAPI3REF: Error Logging Interface *** ** ^The [sqlite3_log0] interface writes a message into the [error log] ** established by the [SQLITE_CONFIG_LOG] option to [sqlite3_config0]. ** ^If logging is enabled, the zFormat string and subsequent arguments are ** used with [sqlite3_snprintf0] to generate the final output string. *** ** The sqlite3_log0 interface is intended for use by extensions such as ** virtual tables, collating functions, and SQL functions. While there is ** nothing to prevent an application from calling sqlite3_log0, doing so ** is considered bad form. *** ** The zFormat string must not be NULL. *** ** To avoid deadlocks and other threading problems, the sqlite3_log0 routine ** will not use dynamically allocated memory. The log message is stored in ** a fixed-length buffer on the stack. If the log message is longer than ** a few hundred characters, it will be truncated to the length of the ** buffer.

79888. Add number of Cells freed to this counter

79889. ** Begin a write-transaction on the specified pager object. If a ** write-transaction has already been opened, this function is a no-op. *** ** If the exFlag argument is false, then acquire at least a RESERVED ** lock on the database file. If exFlag is true, then acquire at least ** an EXCLUSIVE lock. If such a lock is already held, no locking ** functions need be called. *** ** If the subjInMemory argument is non-zero, then any sub-journal opened ** within this transaction will be opened as an in-memory file. This ** has no effect if the sub-journal is already opened (as it may be when ** running in exclusive mode) or if the transaction does not require a ** sub-journal. If the subjInMemory argument is zero, then any required ** sub-journal is implemented in-memory if pPager is an in-memory database, ** or using a temporary file otherwise.

79890. Allocated size of zTerm buffer

79891. Figure out the rowid of the new row.

79892. ***** Include mutex.h in the middle of sqliteInt.h *****

79893. copy a maximum of nBuf chars to output buffer

79894. The btree
79895. Abandon the tree walk
79896. OUT: Number of leaf pages in b-tree
79897. ** The lower 16-bits of the sqlite3_index_info.idxNum value set by ** the xBestIndex() method contains the Fts3Cursor.eSearch value described ** above. The upper 16-bits contain a combination of the following ** bits, used to describe extra constraints on full-text searches.
79898. ** The caller must be holding a SHARED lock on the database file to call ** this function. ** ** If the pager passed as the first argument is open on a real database ** file (not a temp file or an in-memory database), and the WAL file ** is not already open, make an attempt to open it now. If successful, ** return SQLITE_OK. If an error occurs or the VFS used by the pager does ** not support the xShmXXX() methods, return an error code. *pbOpen is ** not modified in either case. ** ** If the pager is open on a temp-file (or in-memory database), or if ** the WAL file is already open, set *pbOpen to 1 and return SQLITE_OK ** without doing anything.
79899. ** Pragma virtual table module xDisconnect method.
79900. !defined(SQLITE_MUTEX_OMIT)
79901. If this database supports auto-vacuum and iPage is not the last ** page in this overflow list, check that the pointer-map entry for ** the following page matches iPage.
79902. Block number of first allocated block
79903. Destructor for a jsonEachCursor object
79904. The number of notReady tables
79905. Range of registers hold content for pIdx
79906. **comment:** ** Free all resources held by the schema structure. The void* argument points ** at a Schema struct. This function does not call sqlite3DbFree(db,) on the ** pointer itself, it just cleans up subsidiary resources (i.e. the contents ** of the schema hash tables). ** ** The Schema.cache_size variable is not cleared.
label: code-design
79907. ** This function is used to estimate the number of rows that will be visited ** by scanning an index for a range of values. The range may have an upper ** bound, a lower bound, or both. The WHERE clause terms that set the upper ** and lower bounds are represented by pLower and pUpper respectively. For ** example, assuming that index p is on t1(a): ** ** ... FROM t1 WHERE a > ? AND a < ? ... ** |_____|_____|** || ** pLower pUpper ** ** If either of the upper or lower bound is not present, then NULL is passed in ** place of the corresponding WhereTerm. ** ** The value in (pBuilder->pNew->u.btree.nEq) is the number of the index ** column subject to the range constraint. Or, equivalently, the number of ** equality constraints optimized by the proposed index scan. For example, ** assuming index p is on t1(a, b), and the SQL query is: ** ** ... FROM t1 WHERE a = ? AND b > ? AND b < ? ... ** ** then nEq is set to 1 (as the range restricted column, b, is the second ** left-most column of the index). Or, if the query is: ** ** ... FROM t1 WHERE a > ? AND a < ? ... ** ** then nEq is set to 0. ** ** When this function is called, *pnOut is set to the sqlite3LogEst() of the ** number of rows that the index scan is expected to visit without ** considering the range constraints. If nEq is 0, then *pnOut is the number of ** rows in the index. Assuming no error occurs, *pnOut is adjusted (reduced) ** to account for the range constraints pLower and pUpper. ** ** In the absence of sqlite_stat4 ANALYZE data, or if such data cannot be ** used, a single range inequality reduces the search space by a factor of 4. ** and a pair of constraints (x>? AND x<?) reduces the expected number of ** rows visited by a factor of 64.
79908. ** Each node of an expression in the parse tree is an instance ** of this structure. ** ** Expr.op is the opcode. The integer parser token codes are reused ** as opcodes here. For example, the parser defines TK_GE to be an integer ** code representing the ">=" operator. This same integer code is reused ** to represent the greater-than-or-equal-to operator in the expression ** tree. ** ** If the expression is an SQL literal (TK_INTEGER, TK_FLOAT, TK_BLOB, ** or TK_STRING), then Expr.token contains the text of the SQL literal. If ** the expression is a variable (TK_VARIABLE), then Expr.token contains the ** variable name. Finally, if the expression is an SQL function (TK_FUNCTION), ** then Expr.token contains the name of the function. ** ** Expr.pRight and Expr.pLeft are the left and right subexpressions of a ** binary operator. Either or both may be NULL. ** ** Expr.x.pList is a list of arguments if the expression is an SQL function, ** a CASE expression or an IN expression of the form "<lhs> IN (<y>, <z>)". ** Expr.x.pSelect is used if the expression is a sub-select or an expression of ** the form "<lhs> IN (SELECT ...)". If the EP_XIsSelect bit is set in the ** Expr.flags mask, then Expr.x.pSelect is valid. Otherwise, Expr.x.pList is ** valid. ** ** An expression of the form ID or ID.ID refers to a column in a table. ** For such expressions, Expr.op is set to TK_COLUMN and Expr.iTable is ** the integer cursor number of a VDBE cursor pointing to that table and ** Expr.iColumn is the column number for the specific column. If the ** expression is used as a result in an aggregate SELECT, then the ** value is also stored in the Expr.iAgg column in the aggregate so that ** it can be accessed after all aggregates are computed. ** ** If the expression is an unbound variable marker (a question mark ** character '?' in the original SQL) then the Expr.iTable holds the index ** number for that variable. ** ** If the expression is a subquery then Expr.iColumn holds an integer ** register number containing the result of the subquery. If the ** subquery gives a constant result, then iTable is -1. If the subquery ** gives a different answer at different times during statement processing ** then iTable is the address of a subroutine that computes the subquery. ** ** If the Expr is of type OP_Column, and the table it is selecting from ** is a disk table or the "old."** pseudo-table, then pTab points to the ** corresponding table definition. ** ** ALLOCATION NOTES: ** ** Expr objects can use a lot of memory space in database schema. To ** help reduce memory requirements, sometimes an Expr object will be ** truncated. And to reduce the number of memory allocations, sometimes ** two or more Expr objects will be stored in a single memory allocation, ** together with Expr.zToken strings. ** ** If the EP_Reduced and EP_TokenOnly flags are set when ** an Expr object is truncated. When EP_Reduced is set, then all ** the child Expr objects in the Expr.pLeft and Expr.pRight subtrees ** are contained within the same memory allocation. Note, however, that ** the subtrees in Expr.x.pList or Expr.x.pSelect are always separately ** allocated, regardless of whether or not EP_Reduced is set.
79909. Write the number of rows in the result here
79910. Pointer to data part of aFrame buffer
79911. Pointer to next docid
79912. 322
79913. **comment:** unused parameter
label: code-design
79914. File handle from CreateFileMapping
79915. SQLITE_MAX_WORKER_THREADS!=0
79916. cmd ::= ROLLBACK trans_opt
79917. Cookie value for main db files
79918. ** These routines are available for the mem2.c debugging memory allocator ** only. They are used to verify that different "types" of memory ** allocations are properly tracked by the system. ** ** sqlite3MemdebugGetType() sets the "type" of an allocation to one of ** the MEMTYPE_* macros defined below. The type must be a bitmask with ** a single bit set. ** ** sqlite3MemdebugHasType() returns true if any of the bits in its second ** argument match the type set by the previous sqlite3MemdebugGetType(). ** sqlite3MemdebugHasType() is intended for use inside assert() statements. ** ** sqlite3MemdebugNoType() returns true if none of the bits in its second ** argument match the type set by the previous sqlite3MemdebugGetType(). ** ** Perhaps the most important point is the difference between MEMTYPE_HEAP ** and MEMTYPE_LOOKASIDE. If an allocation is MEMTYPE_LOOKASIDE, that means ** it might have been allocated by lookaside, except the allocation was ** too large or lookaside was already full. It is important to verify ** that allocations that might have been satisfied by lookaside are not ** passed back to non-lookaside free() routines. Asserts such as the ** example above are placed on the non-lookaside free() routines to verify ** this constraint. ** ** All of this is no-op for a production build. It only comes into ** play when the SQLITE_MEMDEBUG compile-time option is used.
79919. Valid if eType==FTSQUERY_NEAR
79920. ** This is the xRowid method. The SQLite core calls this routine to ** retrieve the rowid for the current row of the result set. fts3 ** exposes %_content.docid as the rowid for the virtual table. The ** rowid should be written to *pRowid.
79921. Pointer to next record in list
79922. ** Lock a file region.
79923. Generate code for the left and right SELECT statements.
79924. ** Like sqlite3ExprIfFalse() except that a copy is made of pExpr before ** code generation, and that copy is deleted after code generation. This ** ensures that the original pExpr is unchanged.
79925. ** The xShmUnmap method.
79926. Open the pager file.
79927. Used only when flags==MEM_Agg
79928. (287) nm ::= STRING
79929. The prtmapCheckPages() contains assert() statements that verify that ** all pointer map pages are set correctly. This is helpful while ** debugging. This is usually disabled because a corrupt database may ** cause an assert() statement to fail.
79930. ***** Include pager.h in the middle of sqliteInt.h *****

79931. OUT: Temp register to free
79932. ERROR: size integer not terminated by "\n"
79933. If control flows to here, it was not possible to append zTerm to the ** current node. Create a new node (a right-sibling of the current node). ** If this is the first node in the tree, the term is added to it. ** ** Otherwise, the term is not added to the new node, it is left empty for ** now. Instead, the term is inserted into the parent of pTree. If pTree ** has no parent, one is created here.
79934. The new virtual table
79935. cmdx ::= cmd
79936. ** Allowed values of WhereTerm.wtFlags
79937. Skip any separator characters.
79938. Creating a virtual table invokes the authorization callback twice. ** The first invocation, to obtain permission to INSERT a row into the ** sqlite_master table, has already been made by sqlite3StartTable(). ** The second call, to obtain permission to create the table, is made now.
79939. Contains agg functions or a GROUP BY
79940. 11
79941. Initialize the master block.
79942. Value is NULL (or a pointer)
79943. ***** End of pager.c *****
79944. Number of VM memory registers
79945. Floating point value
79946. ***** End of json1.c *****
79947. Node type
79948. Column number
79949. Each cursor uses a memory cell. The first cursor (cursor 0) can ** use aMem[0] which is not otherwise used by the VDBE program. Allocate ** space at the end of aMem[] for cursors 1 and greater. ** See also: allocateCursor().
79950. Largest idx in level (iAbsLevel+1)
79951. 4
79952. ** Parse the fts3 query expression found in buffer z, length n. This function ** returns either when the end of the buffer is reached or an unmatched ** closing bracket - ')' - is encountered. ** ** If successful, SQLITE_OK is returned, *ppExpr is set to point to the ** parsed form of the expression and *pnConsumed is set to the number of ** bytes read from buffer z. Otherwise, *ppExpr is set to 0 and SQLITE_NOMEM ** (out of memory error) or SQLITE_ERROR (parse error) is returned.
79953. Input array
79954. Second doclist
79955. IN/OUT: Probe record
79956. 2nd input operand
79957. Disabled if not using stat3
79958. ** This function is used to populate an allocated Fts3TokenAndCost array. ** ** If *pRc is not SQLITE_OK when this function is called, it is a no-op. ** Otherwise, if an error occurs during execution, *pRc is set to an ** SQLite error code.
79959. ** Unless it is NULL, argument zSql points to a buffer allocated using ** sqlite3_malloc containing an SQL statement. This function prepares the SQL ** statement against database db and frees the buffer. If statement ** compilation is successful, *ppStmt is set to point to the new statement ** handle and SQLITE_OK is returned. ** ** Otherwise, if an error occurs, *ppStmt is set to NULL and an error code ** returned. In this case, *pzErrMsg may also be set to point to an error ** message. It is the responsibility of the caller to free this error message ** buffer using sqlite3_free(). ** ** If argument zSql is NULL, this function assumes that an OOM has occurred. ** In this case SQLITE_NOMEM is returned and *ppStmt set to NULL.
79960. The table cursor. Or the PK cursor for WITHOUT ROWID
79961. Resolve the ORDER BY on a compound SELECT after all terms of ** the compound have been resolved.
79962. same as TK_OR, synopsis: r[P3]=r[P1] || r[P2])
79963. ** Process a "special" query. A special query is identified as one with a ** MATCH expression that begins with a '*' character. The remainder of ** the text passed to the MATCH operator are used as the special query ** parameters.
79964. ** Return a list of all dirty pages in the cache, sorted by page number.
79965. ** file_format==1 Version 3.0.0. ** file_format==2 Version 3.1.3. // ALTER TABLE ADD COLUMN ** file_format==3 Version 3.1.4. // ditto but with non-NULL defaults ** file_format==4 Version 3.3.0. // DESC indices. Boolean constants
79966. ** Flags for sqlite3PagerSetFlags() ** ** Value constraints (enforced via assert()): ** PAGER_FULLFSYNC == SQLITE_FullFSync **
PAGER_CKPT_FULLFSYNC == SQLITE_CkptFullFSync ** PAGER_CACHE_SPILL == SQLITE_CacheSpill
79967. Column names
79968. Tokens to the right of last highlight
79969. One of the SRT_ operations to apply to self
79970. No subqueries or non-deterministic functions allowed
79971. ** Code an OP_Halt that causes the vdbe to return an SQLITE_CONSTRAINT ** error. The onError parameter determines which (if any) of the statement ** and/or current transaction is rolled back.
79972. ** Close a tokenization cursor previously opened by a call to ** simpleOpen() above.
79973. ** TBD: Insert subroutine calls to close cursors on incomplete **** subqueries ***
79974. ** 2001 September 15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** This file contains code to implement a pseudo-random number ** generator (PRNG) for SQLite. ** ** Random numbers are used by some of the database backends in order ** to generate random integer keys for tables or random filenames.
79975. 0x14
79976. INSERT statement
79977. Update the database size and return.
79978. ***** End of loadext.c *****
79979. ** Free the MergeEngine object passed as the only argument.
79980. expr ::= LP select RP
79981. ** This function is used by the matchinfo() module to query a phrase ** expression node for the following information: ** ** 1. The total number of occurrences of the phrase in each column of ** the FTS table (considering all rows), and ** ** 2. For each column, the number of rows in the table for which the ** column contains at least one instance of the phrase. ** ** If no error occurs, SQLITE_OK is returned and the values for each column ** written into the array aiOut as follows: ** ** aiOut[iCol*3 + 1] = Number of occurrences ** aiOut[iCol*3 + 2] = Number of rows containing at least one instance ** ** Caveats: ** ** * If a phrase consists entirely of deferred tokens, then all output ** values are set to the number of documents in the table. In other ** words we assume that very common tokens occur exactly once in each ** column of each row of the table. ** ** * If a phrase contains some deferred tokens (and some non-deferred ** tokens), count the potential occurrence identified by considering ** the non-deferred tokens instead of actual phrase occurrences. ** ** * If the phrase is part of a NEAR expression, then only phrase instances ** that meet the NEAR constraint are included in the counts.
79982. ** Set *pCurrent to the total cache hits or misses encountered by all ** pagers the database handle is connected to. *pHighwater is always set ** to zero.
79983. ** This function is called before modifying the contents of a table ** to invalidate any incrblob cursors that are open on the ** row or one of the rows being modified. ** ** If argument isClearTable is true, then the entire contents of the ** table is about to be deleted. In this case invalidate all incrblob ** cursors open on any row within the table with root-page pgnoRoot. ** ** Otherwise, if argument isClearTable is false, then the row with ** rowid iRow is being replaced or deleted. In this case invalidate ** only those incrblob cursors open on that specific row.
79984. Opcode, P3 & P5 of the opcode that ends the loop
79985. table internal
79986. Bytes remaining to copy
79987. Index in a[] of entry with minimum score

79988. Once all the other databases have been initialized, load the schema ** for the TEMP database. This is loaded last, as the TEMP database ** schema may contain references to objects in other databases.

79989. **comment:** ** CAPI3REF: Streaming Versions of API functions. *** The six streaming API xxx_strm() functions serve similar purposes to the ** corresponding non-streaming API functions: *** <table border=1 style="margin-left:8ex;margin-right:8ex"> ** <tr><th>Streaming function</th><th>Non-streaming equivalent</th> ** <tr><td>sqlite3changeset_apply_str<td>[sqlite3changeset_apply] ** <tr><td>sqlite3changeset_concat_str<td>[sqlite3changeset_concat] ** <tr><td>sqlite3changeset_invert_str<td>[sqlite3changeset_invert] ** <tr><td>sqlite3changeset_start_str<td>[sqlite3changeset_start] ** <tr><td>sqlite3session_changeset_str<td>[sqlite3session_changeset] ** <tr><td>sqlite3session_patchset_str<td>[sqlite3session_patchset] ** </table> *** Non-streaming functions that accept changesets (or patchsets) as input ** require that the entire changeset be stored in a single buffer in memory. ** Similarly, those that return a changeset or patchset do so by returning ** a pointer to a single large buffer allocated using sqlite3_malloc(). ** Normally this is convenient. However, if an application running in a ** low-memory environment is required to handle very large changesets, the ** large contiguous memory allocations required can become onerous. ** In order to avoid this problem, instead of a single large buffer, input ** is passed to a streaming API functions by way of a callback function that ** the sessions module invokes to incrementally request input data as it is ** required. In all cases, a pair of API function parameters such as *** <pre> ** &nbs; int nChangeset, ** &nbs; void *pChangeset, ** </pre> *** Is replaced by: *** <pre> ** &nbs; int (*xInput)(void *pIn, void *pData, int *nData), ** &nbs; void *pIn, ** </pre> *** Each time the xInput callback is invoked by the sessions module, the first ** argument passed is a copy of the supplied pIn context pointer. The second ** argument, pData, points to a buffer (*nData) bytes in size. Assuming no ** error occurs the xInput method should copy up to (*nData) bytes of data ** into the buffer and set (*nData) to the actual number of bytes copied ** before returning SQLITE_OK. If the input is completely exhausted, (*nData) ** should be set to zero to indicate this. Or, if an error occurs, an SQLite ** error code should be returned. In all cases, if an xInput callback returns ** an error, all processing is abandoned and the streaming API function ** returns a copy of the error code to the caller. *** In the case of sqlite3changeset_start_strm(), the xInput callback may be ** invoked by the sessions module at any point during the lifetime of the ** iterator. If such an xInput callback returns an error, the iterator enters ** an error state, whereby all subsequent calls to iterator functions ** immediately fail with the same error code as returned by xInput. ** Similarly, streaming API functions that return changesets (or patchsets) ** return them in chunks by way of a callback function instead of via a ** pointer to a single large buffer. In this case, a pair of parameters such ** as: *** <pre> ** &nbs; int *pnChangeset, ** &nbs; void **ppChangeset, ** </pre> *** Is replaced by: *** <pre> ** &nbs; int (*xOutput)(void *pOut, const void *pData, int nData), ** &nbs; void *pOut ** </pre> *** The xOutput callback is invoked zero or more times to return data to ** the application. The first parameter passed to each call is a copy of the ** pOut pointer supplied by the application. The second parameter, pData, ** points to a buffer nData bytes in size containing the chunk of output ** data being returned. If the xOutput callback successfully processes the ** supplied data, it should return SQLITE_OK to indicate success. Otherwise, ** it should return some other SQLite error code. In this case processing ** is immediately abandoned and the streaming API function returns a copy ** of the xOutput error code to the application. ** The sessions module never invokes an xOutput callback with the third ** parameter set to a value less than or equal to zero. Other than this, ** no guarantees are made as to the size of the chunks of data returned.

label: code-design

79990. ** Compare the 19-character string zNum against the text representation ** value 2^63: 9223372036854775808. Return negative, zero, or positive ** if zNum is less than, equal to, or greater than the string. ** Note that zNum must contain exactly 19 characters. ** Unlike memcmp() this routine is guaranteed to return the difference ** in the values of the last digit if the only difference is in the ** last digit. So, for example, *** compare2pow63("9223372036854775800", 1) ** will return -8.

79991. Changeset iterator to read change from

79992. Register SQL function on this connection

79993. ***** End of vdbeapi.c *****

79994. Insert the i-th overflow cell before the aiOvfl-th ** non-overflow cell

79995. Generate a subroutine that will reset the group-by accumulator

79996. Stmt under construction

79997. ** Routine to transform a unixFile into a proxy-locking unixFile. ** Implementation in the proxy-lock division, but used by unixOpen() ** if SQLITE_PREFER_PROXY_LOCKING is defined.

79998. Write result into this Mem object

79999. New cell becomes the i-th cell of the page

80000. Search for the index that has the lowest scan cost. ** ** (2011-04-15) Do not do a full scan of an unordered index. ** ** (2013-10-03) Do not count the entries in a partial index. ** ** In practice the KeyInfo structure will not be used. It is only ** passed to keep OP_OpenRead happy.

80001. ***** End of whereInt.h *****

80002. Actual size of mapped region

80003. ** Query to see if Btree handle p may obtain a lock of type eLock ** (READ_LOCK or WRITE_LOCK) on the table with root-page iTab. Return ** SQLITE_OK if the lock may be obtained (by calling ** setSharedCacheTableLock()), or SQLITE_LOCKED if not.

80004. Set of best costs

80005. Offset of entry in pointer map

80006. ** Usage: ** ** assert(assert_pager_state(pPager)); ** ** This function runs many asserts to try to find inconsistencies in ** the internal state of the Pager object.

80007. The source or pattern file

80008. Do not issue error messages if true

80009. ** CAPI3REF: Status Parameters ** KEYWORDS: {status parameters} *** These integer constants designate various run-time status parameters ** that can be returned by [sqlite3_status()]. *** <dl> ** [[SQLITE_STATUS_MEMORY_USED]] ^<dt>SQLITE_STATUS_MEMORY_USED</dt> ** <dd>This parameter is the current amount of memory checked out ** using [sqlite3_malloc()], either directly or indirectly. The ** figure includes calls made to [sqlite3_malloc()] by the application ** and internal memory usage by the SQLite library. Scratch memory ** controlled by [SQLITE_CONFIG_SCRATCH] and auxiliary page-cache ** memory controlled by [SQLITE_CONFIG_PAGECACHE] is not included in ** this parameter. The amount returned is the sum of the allocation ** sizes as reported by the xSize method in [sqlite3_mem_methods].</dd>^<dd>[[SQLITE_STATUS_MALLOC_SIZE]] ^<dt>SQLITE_STATUS_MALLOC_SIZE</dt> ** <dd>This parameter records the largest memory allocation request ** handed to [sqlite3_malloc()] or [sqlite3_realloc()] (or their ** internal equivalents). Only the value returned in the ** *pHighwater parameter to [sqlite3_status()] is of interest. ** The value written into the *pCurrent parameter is undefined.</dd>^<dd>[[SQLITE_STATUS_MALLOC_COUNT]] ^<dt>SQLITE_STATUS_MALLOC_COUNT</dt> ** <dd>This parameter records the number of separate memory allocations ** currently checked out.</dd>^<dd>[[SQLITE_STATUS_PAGECACHE_USED]] ^<dt>SQLITE_STATUS_PAGECACHE_USED</dt> ** <dd>This parameter returns the number of pages used out of the ** [pagecache memory allocator] that was configured using ** [SQLITE_CONFIG_PAGECACHE]. The ** value returned is in pages, not in bytes.</dd>^<dd>[[SQLITE_STATUS_PAGECACHE_OVERFLOW]] ^<dt>SQLITE_STATUS_PAGECACHE_OVERFLOW</dt> ** <dd>This parameter returns the number of bytes of page cache ** allocation which could not be satisfied by the [SQLITE_CONFIG_PAGECACHE] ** buffer and where forced to overflow to [sqlite3_malloc()]. The ** returned value includes allocations that overflowed because they ** were too large (they were larger than the "sz" parameter to ** [SQLITE_CONFIG_PAGECACHE]) and allocations that overflowed because ** no space was left in the page cache.</dd>^<dd>[[SQLITE_STATUS_PAGECACHE_SIZE]] ^<dt>SQLITE_STATUS_PAGECACHE_SIZE</dt> ** <dd>This parameter records the largest memory allocation request ** handed to [pagecache memory allocator]. Only the value returned in the ** *pHighwater parameter to [sqlite3_status()] is of interest. ** The value written into the *pCurrent parameter is undefined.</dd>^<dd>[[SQLITE_STATUS_SCRATCH_USED]] ^<dt>SQLITE_STATUS_SCRATCH_USED</dt> ** <dd>This parameter returns the number of allocations used out of the ** [scratch memory allocator] configured using ** [SQLITE_CONFIG_SCRATCH]. The value returned is in allocations, not ** in bytes. Since a single thread may only have one scratch allocation ** outstanding at time, this parameter also reports the number of threads ** using scratch memory at the same time.</dd>^<dd>[[SQLITE_STATUS_SCRATCH_OVERFLOW]] ^<dt>SQLITE_STATUS_SCRATCH_OVERFLOW</dt> ** <dd>This parameter returns the number of bytes of scratch memory ** allocation which could not be satisfied by the [SQLITE_CONFIG_SCRATCH] ** buffer and where forced to overflow to [sqlite3_malloc()]. The values ** returned include overflows because the requested allocation was too ** larger (that is, because the requested allocation was larger than the ** "sz" parameter to [SQLITE_CONFIG_SCRATCH]) and because no scratch buffer ** slots were available. ** </dd>^<dd>[[SQLITE_STATUS_SCRATCH_SIZE]] ^<dt>SQLITE_STATUS_SCRATCH_SIZE</dt> ** <dd>This parameter records the largest memory allocation request ** handed to [scratch memory allocator]. Only the value returned in the ** *pHighwater parameter to [sqlite3_status()] is of interest. ** The value written into the *pCurrent parameter is undefined.</dd>^<dd>[[SQLITE_STATUS_PARSER_STACK]] ^<dt>SQLITE_STATUS_PARSER_STACK</dt> ** <dd>The *pHighwater parameter records the deepest parser stack. ** The *pCurrent value is undefined. The *pHighwater value is only ** meaningful if SQLite is compiled with [YYTRACKMAXSTACKDEPTH].</dd>^<dd>New status parameters may be added from time to time.

80010. 165

80011. **comment:** The print_pager_state() routine is intended to be used by the debugger ** only. We invoke it once here to suppress a compiler warning.

label: code-design

80012. The setup query
80013. Fill the automatic index with content
80014. If the plan produced by the earlier call uses an IN(...) term, call ** xBestIndex again, this time with IN(...) terms disabled.
80015. EVIDENCE-OF: R-49920-60189 If the first pointer (the memory pointer) ** is NULL, then SQLite reverts to using its default memory allocator ** (the system malloc() implementation), undoing any prior invocation of ** SQLITE_CONFIG_MALLOC. *** Setting sqlite3GlobalConfig.m to all zeros will cause malloc to ** revert to its default implementation when sqlite3_initialize() is run
80016. porter rule condition: (m > 0)
80017. '|'. Bitwise OR or concatenate
80018. OUT: Permissions of zFile
80019. SQLITE_NOLFS
80020. ***** End of sqliteicu.h *****
80021. ** SQL is translated into a sequence of instructions to be ** executed by a virtual machine. Each instruction is an instance ** of the following structure.
80022. ***** Scalar SQL function implementations *****
80023. Whether or not a database might need a master journal depends upon ** its journal mode (among other things). This matrix determines which ** journal modes use a master journal and which do not
80024. Number of slots of aNode[] allocated
80025. Database connection
80026. Destination b-tree file
80027. 88
80028. Table being updated
80029. Pointer to cell-pointer area
80030. Doclist (or NULL) to write
80031. ** Verify that the VM passed as the only argument does not contain ** an OP_ResultRow opcode. Fail an assert() if it does. This is used ** by code in pragma.c to ensure that the implementation of certain ** pragmas comports with the flags specified in the mkpragmatab.tcl ** script.
80032. 277
80033. The following block is a no-op unless SQLITE_DEBUG is defined. Its only ** purpose is to execute assert() statements to verify that if the ** PragFlg_NoColumns1 flag is set and the caller specified an argument ** to the PRAGMA, the implementation has not added any OP_ResultRow ** instructions to the VM.
80034. ** Write data from a buffer into a file. Return SQLITE_OK on success ** or some other error code on failure.
80035. Colset to filter on (or NULL)
80036. Pointer to mapping of entire file
80037. ** Return TRUE if the cursor has been moved off of the last ** row of output.
80038. ** Record the fact that we want to lock a table at run-time. *** The table to be locked has root page iTab and is found in database iDb. ** A read or a write lock can be taken depending on isWritelock. *** This routine just records the fact that the lock is desired. The ** code to make the lock occur is generated by a later call to ** codeTableLocks() which occurs during sqlite3FinishCoding().
80039. Allocate exact page if possible
80040. 3 EXPLAIN:
80041. storage for current token
80042. **comment:** No longer used
 label: code-design
80043. Skip the "sqlite_altertab_" prefix on the name
80044. Max pending data before flush to disk
80045. Number of TryBeginRead attempts
80046. Make sure all SELECTs in the statement have the same number of elements ** in their result sets.
80047. OUT: Deserialized object
80048. (294) cagelist ::=
80049. Schema restricts name search if present
80050. cached details from stats0
80051. ***** End %parse_accept code *****
80052. ** Return the number of the wal-index page that contains the hash-table ** and page-number array that contain entries corresponding to WAL frame ** iFrame. The wal-index is broken up into 32KB pages. Wal-index pages ** are numbered starting from 0.
80053. in1, out2, in3
80054. UTF-8 -> UTF-16 Little-endian
80055. Character iIn and iIn+1 form an escaped quote character. Skip ** the input cursor past both and copy a single quote character ** to the output buffer.
80056. Cache used by auxiliary functions xInst() and xInstCount()
80057. Tables marked isRecursive have only a single row that is stored in ** a pseudo-cursor. No need to Rewind or Next such cursors.
80058. If possible, improve on the iUpper estimate using (\$P:\$U).
80059. no-op if possible
80060. Only operate the lookaside when zero
80061. Output variables. Containing the current node entry.
80062. #include "sqlite3.h"
80063. Direct rendering mode means take data directly ** from source tables rather than from accumulators
80064. ***** Begin %stack_overflow code *****
80065. 0: hits. 1: size misses. 2: full misses
80066. ** Serialize and store the "structure" record. *** If an error occurs, leave an error code in the Fts5Index object. If an ** error has already occurred, this function is a no-op.
80067. INTERSECT
80068. ** Load the doclist for phrase p into p->doclist.aAll/nAll. The loaded doclist ** does not take deferred tokens into account. *** SQLITE_OK is returned if no error occurs, otherwise an SQLite error code.
80069. Collect arguments into local variables
80070. paren_explist
80071. Output value
80072. Buffer containing previous term
80073. Current 'pos' value
80074. Masks of old.* , new.* columns accessed
80075. pWLoop is a winner. Add it to the set of best so far
80076. This is not VxWorks.
80077. **comment:** Remove the redundant segments from the input level
 label: code-design
80078. Number of unused bytes at end of each page
80079. Need to read this page properly. It contains some of the ** range of data that is being read (eOp==0) or written (eOp!=0).
80080. SQLITE_ENABLE_COLUMN_METADATA
80081. Comment
80082. Index of free page to move pLastPg to
80083. Flags passed to OsOpen()
80084. Index on parent covering the foreign key
80085. ** Return true if the buffer z[0..n-1] contains all spaces.

80086. ** This version of the memory allocator is only built into the library ** SQLITE_ENABLE_MEMSYS3 is defined. Defining this symbol does not ** mean that the library will use a memory-pool by default, just that ** it is available. The mempool allocator is activated by calling ** sqlite3_config().

80087. ** Ensure these symbols were not defined by some previous header file.

80088. REINDEX

80089. a0..a7

80090. ***** End of walker.c *****

80091. ** This function is called as part of committing a transaction within which ** one or more frames have been overwritten. It updates the checksums for ** all frames written to the wal file by the current transaction starting ** with the earliest to have been overwritten. ** ** SQLITE_OK is returned if successful, or an SQLite error code otherwise.

80092. ** This function is called to merge two changes to the same row together as ** part of an sqlite3changeset_concat() operation. A new change object is ** allocated and a pointer to it stored in *ppNew.

80093. ***** End of btree.h *****

80094. ** Do we need to manually define the Win32 file mapping APIs for use with WAL ** mode or memory mapped files (e.g. these APIs are available in the Windows ** CE SDK; however, they are not present in the header file)?

80095. Link pParent into the free node list. It will be used as an ** internal node of the new tree.

80096. Assume single column records only

80097. Jump destination for skipping partial index entries

80098. YYWILDCARD

80099. Description of the table we are reading from

80100. Fire any BEFORE UPDATE triggers. This happens before constraints are ** verified. One could argue that this is wrong.

80101. ** Allocate space for a file-handle and open a temporary file. If successful, ** set *ppFd to point to the malloc'd file-handle and return SQLITE_OK. ** Otherwise, set *ppFd to 0 and return an SQLite error code.

80102. Contains NOT NULL constraints

80103. temp ::=

80104. Opcode: SequenceTest P1 P2 * * * * Synopsis: if(cursor[P1].ctr++) pc = P2 ** ** P1 is a sorter cursor. If the sequence counter is currently zero, jump ** to P2. Regardless of whether or not the jump is taken, increment the ** sequence value.

80105. 0x87f7f is a mask of SQLITE_OPEN_flags that are valid to be passed ** down into the VFS layer. Some SQLITE_OPEN_flags (for example, ** SQLITE_OPEN_FULLMUTEX or SQLITE_OPEN_SHARED_CACHE) are blocked before ** reaching the VFS.

80106. Cannot be both MEM_Int and MEM_Real at the same time

80107. Insert an element into the hash table pH. The key is pKey,nKey ** and the data is "data". ** ** If no element exists with a matching key, then a new ** element is created. A copy of the key is made if the copyKey ** flag is set. NULL is returned. ** ** If another element already exists with the same key, then the ** new data replaces the old data and the old data is returned. ** The key is not copied in this instance. If a malloc fails, then ** the new data is returned and the hash table is unchanged. ** ** If the "data" parameter to this function is NULL, then the ** element corresponding to "key" is removed from the hash table.

80108. Buffer in which to assemble leaf page

80109. ** An implementation of the UnlockFile API of Windows for CE

80110. ** Run a checkpoint on the Btree passed as the first argument. ** ** Return SQLITE_LOCKED if this or any other connection has an open ** transaction on the shared-cache the argument Btree is connected to. ** ** Parameter eMode is one of SQLITE_CHECKPOINT_PASSIVE, FULL or RESTART.

80111. An element of aFrom[] that we are working on

80112. 600

80113. Register to allocate runtime space

80114. Drop the cell from the parent page. apDiv[i] still points to ** the cell within the parent, even though it has been dropped. ** This is safe because dropping a cell only overwrites the first ** four bytes of it, and this function does not need the first ** four bytes of the divider cell. So the pointer is safe to use ** later on. ** ** But not if we are in secure-delete mode. In secure-delete mode, ** the dropCell() routine will overwrite the entire cell with zeroes. ** In this case, temporarily copy the cell into the aOvflSpace[] ** buffer. It will be copied out again as soon as the aSpace[] buffer ** is allocated.

80115. Right table being joined

80116. Allocated index object

80117. Step 5b

80118. ** CAPI3REF: Write-Ahead Log Commit Hook ** METHOD: sqlite3 ** ** ^The [sqlite3_wal_hook()] function is used to register a callback that ** is invoked each time data is committed to a database in wal mode. ** ** ^The callback is invoked by SQLite after the commit has taken place and ** the associated write-lock on the database released^, so the implementation ** may read, write or [checkpoint] the database as required. ** ** ^The first parameter passed to the callback function when it is invoked ** is a copy of the third parameter passed to sqlite3_wal_hook() when ** registering the callback. ^The second is a copy of the database handle. ** ^The third parameter is the name of the database that was written to - ** either "main" or the name of an [ATTACH]-ed database. ^The fourth parameter ** is the number of pages currently in the write-ahead log file, ** including those that were just committed. ** ** The callback function should normally return [SQLITE_OK]. ^If an error ** code is returned, that error will propagate back up through the ** SQLite code base to cause the statement that provoked the callback ** to report an error, though the commit will have still occurred. If the ** callback returns [SQLITE_ROW] or [SQLITE_DONE], or if it returns a value ** that does not correspond to any valid SQLite error code, the results ** are undefined. ** ** A single database handle may have at most a single write-ahead log callback ** registered at one time. ^Calling [sqlite3_wal_hook()] replaces any ** previously registered write-ahead log callback. ^Note that the ** [sqlite3_wal_autocheckpoint()] interface and the ** [wal_autocheckpoint pragma] both invoke [sqlite3_wal_hook()] and will ** overwrite any prior [sqlite3_wal_hook()] settings.

80119. If this is an RBU vacuum handle and the vacuum has either finished ** successfully or encountered an error, delete the contents of the ** state table. This causes the next call to sqlite3rbu_vacuum() ** specifying the current target and state databases to start a new ** vacuum from scratch.

80120. ** Generate code to return a single text value.

80121. A-overwrites-S

80122. 100 for page 1. 0 otherwise

80123. Top of the delete loop

80124. read the conch content

80125. The following block sets the new values of Mem.z and Mem.xDel. It ** also sets a flag in local variable "flags" to indicate the memory ** management (one of MEM_Dyn or MEM_Static).

80126. The current candidate is no better than any of the mxChoice ** paths currently in the best-so-far buffer. So discard ** this candidate as not viable.

80127. ***** Begin flock Locking *****

***** Use the flock() system call to do file locking. ** ** flock() locking is like dot-file locking in that the various ** fine-grain locking levels supported by SQLite are collapsed into ** a single exclusive lock. In other words, SHARED, RESERVED, and ** PENDING locks are the same thing as an EXCLUSIVE lock. SQLite ** still works when you do this, but concurrency is reduced since ** only a single process can be reading the database at a time. ** ** Omit this section if SQLITE_ENABLE_LOCKING_STYLE is turned off

80128. Allocate and populate a new Parse context to use for coding the ** trigger sub-program.

80129. Verify correct alignment of TK_ and OP_ constants

80130. Set delete-flag @ iSzPoslist

80131. First argument for the conflict handler

80132. True if at end of search

80133. if SystemTimeToFileTime() fails, it returns zero.

80134. Type of value from change record

80135. ** This function is used to access the current position list for phrase ** iPhrase.

80136. Do not allow the total number of threads (main thread + all workers) ** to exceed the maximum merge count

80137. ** Here are all of the sqlite3_io_methods objects for each of the ** locking strategies. Functions that return pointers to these methods ** are also created.

80138. Number of cursors for child frame

80139. Part of a compound query

80140. ** Expr pIn is an IN(...) expression. This function checks that the ** sub-select on the RHS of the IN() operator has the same number of ** columns as the vector on the LHS. Or, if the RHS of the IN() is not ** a sub-query, that the LHS is a vector of size 1.

80141. Number of columns in the sorting index
80142. **comment:** Type of the array "element" for the bitmap representation. ** Should be a power of 2, and ideally, evenly divide into BITVEC_USIZE. ** Setting this to the "natural word" size of your CPU may improve ** performance.
label: code-design
80143. Index number
80144. ** If argument pOrig is NULL, or if (*pRc) is set to anything other than ** SQLITE_OK when this function is called, NULL is returned. ** ** Otherwise, a copy of (*pOrig) is made into memory obtained from ** sqlite3Fts5MallocZero() and a pointer to it returned. If the allocation ** fails, (*pRc) is set to SQLITE_NOMEM and NULL is returned.
80145. ** Mutex to control access to the memory allocation subsystem.
80146. Number of entries in aFirst[]
80147. ***** Below this point is the implementation of the fts5_decode() scalar ** function only.
80148. ** CAPI3REF: Return The Size Of An Open BLOB ** METHOD: sqlite3_blob ** ** ^Returns the size in bytes of the BLOB accessible via the ** successfully opened [BLOB handle] in its only argument. ^The ** incremental blob I/O routines can only read or overwriting existing ** blob content; they cannot change the size of a blob. ** ** This routine only works on a [BLOB handle] which has been created ** by a prior successful call to [sqlite3_blob_open()] and which has not ** been closed by [sqlite3_blob_close()]. Passing any other pointer in ** to this routine results in undefined and probably undesirable behavior.
80149. ** SQLite supports many different ways to resolve a constraint ** error. ROLLBACK processing means that a constraint violation ** causes the operation in process to fail and for the current transaction ** to be rolled back. ABORT processing means the operation in process ** fails and any prior changes from that one operation are backed out, ** but the transaction is not rolled back. FAIL processing means that ** the operation in progress stops and returns an error code. But prior ** changes due to the same operation are not backed out and no rollback ** occurs. IGNORE means that the particular row that caused the constraint ** error is not inserted or updated. Processing continues and no error ** is returned. REPLACE means that preexisting database rows that caused ** a UNIQUE constraint violation are removed so that the new insert or ** update can proceed. Processing continues and no error is reported. ** ** RESTRICT, SETNULL, and CASCADE actions apply only to foreign keys. ** RESTRICT is the same as ABORT for IMMEDIATE foreign keys and the ** same as ROLLBACK for DEFERRED keys. SETNULL means that the foreign ** key is set to NULL. CASCADE means that a DELETE or UPDATE of the ** referenced table row is propagated into the row that holds the ** foreign key. ** ** The following symbolic values are used to record which type ** of action to take.
80150. ** The input pointer currently points to the second byte of a table-header. ** Specifically, to the following: ** ** + number of columns in table (varint) ** + array of PK flags (1 byte per column), ** + table name (nul terminated). ** ** This function decodes the table-header and populates the p->nCol, ** p->zTab and p->abPK[] variables accordingly. The p->apValue[] array is ** also allocated or resized according to the new value of p->nCol. The ** input pointer is left pointing to the byte following the table header. ** ** If successful, SQLITE_OK is returned. Otherwise, an SQLite error code ** is returned and the final values of the various fields enumerated above ** are undefined.
80151. **comment:** Issue SQLITE_READ authorizations with a fake column name for any tables that ** are referenced but from which no values are extracted. Examples of where these ** kinds of null SQLITE_READ authorizations would occur: ** ** SELECT count(*) FROM t1; -- SQLITE_READ t1.**" ** SELECT t1.* FROM t1, t2; -- SQLITE_READ t2."** ** The fake column name is an empty string. It is possible for a table to ** have a column named by the empty string, in which case there is no way to ** distinguish between an unreferenced table and an actual reference to the ** "" column. The original design was for the fake column name to be a NULL, ** which would be unambiguous. But legacy authorization callbacks might ** assume the column name is non-NUL and segfault. The use of an empty string ** for the fake column name seems safer.
label: code-design
80152. Tables that must be scanned after this one
80153. Tokenizer functions
80154. ***** End of rtree.h *****
80155. paren_explist ::= LP exprlist RP
80156. ** 0) SELECT count(*) FROM sqlite_master where name=%Q AND IsVirtual(%Q) ** 1) PRAGMA index_list = ? ** 2) SELECT count(*) FROM sqlite_master where name=%Q ** 3) PRAGMA table_info = ?
80157. OUT: Compiled statement
80158. ** Set the time to the current time reported by the VFS. ** ** Return the number of errors.
80159. If a column from a table in the pSrcList is referenced, then record ** this fact in the pSrcList.a[].colUsed bitmask. Column 0 causes ** bit 0 to be set. Column 1 sets bit 1. And so forth. If the ** column number is greater than the number of bits in the bitmask ** then set the high-order bit of the bitmask.
80160. Non-recursive part
80161. zNum is a 19-digit numbers. Compare it against 9223372036854775808.
80162. The table being dropped does not have the largest root-page ** number in the database. So move the page that does into the ** gap left by the deleted root-page.
80163. ** Load the content of the sqlite_stat1 and sqlite_stat3/4 tables. The ** contents of sqlite_stat1 are used to populate the Index.aiRowEst[] ** arrays. The contents of sqlite_stat3/4 are used to populate the ** Index.aSample[] arrays. ** ** If the sqlite_stat1 table is not present in the database, SQLITE_ERROR ** is returned. In this case, even if SQLITE_ENABLE_STAT3/4 was defined ** during compilation and the sqlite_stat3/4 table is present, no data is ** read from it. ** ** If SQLITE_ENABLE_STAT3/4 was defined during compilation and the ** sqlite_stat4 table is not present in the database, SQLITE_ERROR is ** returned. However, in this case, data is read from the sqlite_stat1 ** table (if it is present) before returning. ** ** If an OOM error occurs, this function always sets db->mallocFailed. ** This means if the caller does not care about other errors, the return ** code may be ignored.
80164. OUT: New value object (or NULL)
80165. **comment:** The database connection used for malloc()
label: code-design
80166. Case 6: There is no usable index. We must do a complete ** scan of the entire table.
80167. wherencode.c:
80168. WHERE clause processing context
80169. Affinity used when eDest==SRT_Set
80170. 0x1f
80171. Locks obtained by this instance of winFile
80172. ** An instance of the following structure is allocated for each active ** savepoint and statement transaction in the system. All such structures ** are stored in the Pager.aSavepoint[] array, which is allocated and ** resized using sqlite3Realloc(). ** ** When a savepoint is created, the PagerSavepoint.iHdrOffset field is ** set to 0. If a journal-header is written into the main journal while ** the savepoint is active, then iHdrOffset is set to the byte offset ** immediately following the last journal record written into the main ** journal before the journal-header. This is required during savepoint ** rollback (see pagerPlaybackSavepoint()).
80173. **comment:** Mark a function parameter as unused, to suppress nuisance compiler ** warnings.
label: code-design
80174. Set the maximum error count
80175. Collating sequence for the result set
80176. Extra data passed into the callback
80177. Creating a new table may probably require moving an existing database ** to make room for the new tables root page. In case this page turns ** out to be an overflow page, delete all overflow page-map caches ** held by open cursors.
80178. Number of integers output by cArg
80179. **comment:** ** This function is used to create a SELECT list (the list of SQL ** expressions that follows a SELECT keyword) for a SELECT statement ** used to read from an data_xxx or rbu_tmp_xxx table while updating the ** index object currently indicated by the iterator object passed as the ** second argument. A "PRAGMA index_xinfo = <idxname>" statement is used ** to obtain the required information. ** ** If the index is of the following form: ** ** CREATE INDEX i1 ON t1(c, b COLLATE nocase); ** ** and "t1" is a table with an explicit INTEGER PRIMARY KEY column ** "ipk", the returned string is: ** ** "c` COLLATE 'BINARY', 'b' COLLATE 'NOCASE', 'ipk' COLLATE 'BINARY'" ** ** As well as the returned string, three other malloc'd strings are ** returned via output parameters. As follows: ** ** pzImposterCols: ... ** pzImposterPk: ... ** pzWhere: ...
label: code-design
80180. ** Used when SQLITE_NO_SYNC is not defined and by the assert() and/or ** OSTRACE() macros.
80181. This page from bulk local storage
80182. Automatically generated. Do not edit

80183. Table structure column is extracted from
80184. zFilename is a temporary or immutable file
80185. Begin flattening the iFrom-th entry of the FROM clause ** in the outer query.
80186. Compute the old pre-UPDATE content of the row being changed, if that ** information is needed
80187. go back to the do {} while start point, try again
80188. In this case the pcache already contains an initialized copy of ** the page. Return without further ado.
80189. This is the bogus content
80190. No. of coordinates
80191. Write any error message here
80192. sqlite3SelectNew() guarantees this
80193. The attached database containing the blob
80194. ** fts3ExprIterate() callback used to collect the "global" matchinfo stats ** for a single query. *** fts3ExprIterate() callback to load the 'global' elements of a ** FTS3_MATCHINFO_HITS matchinfo array. The global stats are those elements ** of the matchinfo array that are constant for all rows returned by the ** current query. *** Argument pCtx is actually a pointer to a struct of type MatchInfo. This ** function populates Matchinfo.aMatchinfo[] as follows: ** for(iCol=0; iCol<nCol; iCol++){ ** aMatchinfo[3*iPhrase*nCol + 3*iCol + 1] = X; ** aMatchinfo[3*iPhrase*nCol + 3*iCol + 2] = Y; ** } *** where X is the number of matches for phrase iPhrase is column iCol of all ** rows of the table. Y is the number of rows for which column iCol contains ** at least one instance of phrase iPhrase. *** If the phrase pExpr consists entirely of deferred tokens, then all X and ** Y values are set to nDoc, where nDoc is the number of documents in the ** file system. This is done because the full-text index doclist is required ** to calculate these values properly, and the full-text index doclist is ** not available for deferred tokens.
80195. Do not load data from disk
80196. READ UNCOMMITTED in shared-cache
80197. Allocate the change object
80198. The PATCH
80199. Parse object
80200. ** Free up as much memory as we can from the given database ** connection.
80201. Create extra terms on the WHERE clause for each column named ** in the USING clause. Example: If the two tables to be joined are ** A and B and the USING clause names X, Y, and Z, then add this ** to the WHERE clause: A.X=B.X AND A.Y=B.Y AND A.Z=B.Z ** Report an error if any column mentioned in the USING clause is ** not contained in both tables to be joined.
80202. pIter now points just past the 0x00 that terminates the position- ** list for document pDL->iDocid. However, if this position-list was ** edited in place by fts3EvalNearTrim(), then pIter may not actually ** point to the start of the next docid value. The following line deals ** with this case by advancing pIter past the zero-padding added by ** fts3EvalNearTrim().
80203. Merge the two position lists.
80204. ** Erase the given database page and all its children. Return ** the page to the freelist.
80205. If SQLITE_UTF16 is specified as the encoding type, transform this ** to one of SQLITE_UTF16LE or SQLITE_UTF16BE using the ** SQLITE_UTF16NATIVE macro. SQLITE_UTF16 is not used internally.
80206. One release per successful fetch. Page is pinned until released. ** Reference counted.
80207. Register to hold key for checking the FK
80208. Increase the leaves written counter
80209. SQLITE_OMIT_TRUNCATE_OPTIMIZATION
80210. which columns to include in the result
80211. Different collating sequences
80212. defined(SQLITE_ENABLE_STAT3_OR_STAT4)
80213. 800
80214. True to use onepass strategy
80215. Index scan is using
80216. ** If pFile holds a lock on a conch file, then release that lock.
80217. ** Implementation of the substr() function. *** substr(x,p1,p2) returns p2 characters of x[] beginning with p1. ** p1 is 1-indexed. So substr(x,1,1) returns the first character ** of x. If x is text, then we actually count UTF-8 characters. ** If x is a blob, then we count bytes. *** If p1 is negative, then we begin abs(p1) from the end of x[]. ** If p2 is negative, return the p2 characters preceding p1.
80218. ** Check that there is no open read-transaction on the b-tree passed as the ** second argument. If there is not, return SQLITE_OK. Otherwise, if there ** is an open read-transaction, return SQLITE_ERROR and leave an error ** message in database handle db.
80219. Alternative data destination
80220. The number of integers to decode
80221. True if cursor is at EOF
80222. ** Compute the number of pages of cache requested. p->szCache is the ** cache size requested by the "PRAGMA cache_size" statement.
80223. Opcode: Prev P1 P2 P3 P4 P5 *** Back up cursor P1 so that it points to the previous key/data pair in its ** table or index. If there is no previous key/value pairs then fall through ** to the following instruction. But if the cursor backup was successful, ** jump immediately to P2. *** The Prev opcode is only valid following an SeekLT, SeekLE, or ** OP_Last opcode used to position the cursor. Prev is not allowed ** to follow SeekGT, SeekGE, or OP_Rewind. *** The P1 cursor must be for a real table, not a pseudo-table. If P1 is ** not open then the behavior is undefined. *** The P3 value is a hint to the btree implementation. If P3==1, that ** means P1 is an SQL index and that this instruction could have been ** omitted if that index had been unique. P3 is usually 0. P3 is ** always either 0 or 1. *** P4 is always of type P4_ADVANCE. The function pointer points to ** sqlite3BtreePrevious(). *** If P5 is positive and the jump is taken, then event counter ** number P5-1 in the prepared statement is incremented.
80224. aXRef[i] is the index in pChanges->a[] of the ** an expression for the i-th column of the table. ** aXRef[i]==-1 if the i-th column is not changed.
80225. **comment:** ** Lookaside malloc is a set of fixed-size buffers that can be used ** to satisfy small transient memory allocation requests for objects ** associated with a particular database connection. The use of ** lookaside malloc provides a significant performance enhancement ** (approx 10%) by avoiding numerous malloc/free requests while parsing ** SQL statements. *** The Lookaside structure holds configuration information about the ** lookaside malloc subsystem. Each available memory allocation in ** the lookaside subsystem is stored on a linked list of LookasideSlot ** objects. *** Lookaside allocations are only allowed for objects that are associated ** with a particular database connection. Hence, schema information cannot ** be stored in lookaside because in shared cache mode the schema information ** is shared by multiple database connections. Therefore, while parsing ** schema information, the Lookaside.bEnabled flag is cleared so that ** lookaside allocations are not used to construct the schema objects.
label: code-design
80226. Changes list for any UPDATE OF triggers
80227. get sign of significand
80228. ** Lock the database file to level eLock, which must be either SHARED_LOCK, ** RESERVED_LOCK or EXCLUSIVE_LOCK. If the caller is successful, set the ** Pager.eLock variable to the new locking state. *** Except, if Pager.eLock is set to UNKNOWN_LOCK when this function is ** called, do not modify it unless the new locking state is EXCLUSIVE_LOCK. ** See the comment above the #define of UNKNOWN_LOCK for an explanation ** of this.
80229. ** The hashing function.
80230. Shortcut for the case when the freelist is empty
80231. ** Read part of the payload for the row at which that cursor pCur is currently ** pointing. "amt" bytes will be transferred into pBuf[]. The transfer ** begins at "offset". ** pCur can be pointing to either a table or an index b-tree. ** If pointing to a table btree, then the content section is read. If ** pCur is pointing to an index b-tree then the key section is read. *** For sqlite3BtreePayload(), the caller must ensure that pCur is pointing ** to a valid row in the table. For sqlite3BtreePayloadChecked(), the ** cursor might be invalid or might need to be restored before being read. *** Return SQLITE_OK on success or an error code if anything goes ** wrong. An error is returned if "offset+amt" is larger than ** the available payload.
80232. ** Function getNextNode(), which is called by fts3ExprParse(), may itself ** call fts3ExprParse(). So this forward declaration is required.
80233. Recursive part
80234. The number of overflow pages to load for this (and therefore all ** subsequent) tokens is greater than the estimated number of pages ** that will be loaded if all subsequent tokens are deferred.
80235. Virtual tables must be handled separately

80236. * The maximum size of the Win32-specific heap. This value may be zero.
80237. ** Check the leaf RTree cell given by pCellData against constraint p. ** If this constraint is not satisfied, set *peWithin to NOT_WITHIN. ** If the constraint is satisfied, leave *peWithin unchanged. ** ** The constraint is of the form: xN op \$val ** ** The op is given by p->op. The xN is p->iCoord-th coordinate in ** pCellData. \$val is given by p->u.rValue.
80238. Create a new ephemeral function definition for the overloaded ** function
80239. ** The variable-length integer encoding is as follows: ** ** KEY: ** A = 0xxxxxxxx 7 bits of data and one flag bit ** B = 1xxxxxxxx 7 bits of data and one flag bit
** C = xxxxxxxx 8 bits of data ** ** 7 bits - A ** 14 bits - BA ** 21 bits - BBA ** 28 bits - BBBA ** 35 bits - BBBBA ** 42 bits - BBBBBA ** 49 bits -
BBBBBBA ** 56 bits - BBBBBBBA ** 64 bits - BBBBBBBBC
80240. **comment:** Consider using a skip-scan if there are no WHERE clause constraints ** available for the left-most terms of the index, and if the average ** number of repeats in the left-most terms is at least 18. ** ** The magic number 18 is selected on the basis that scanning 17 rows ** is almost always quicker than an index seek (even though if the index ** contains fewer than 2^{17} rows we assume otherwise in other parts of ** the code). And, even if it is not, it should not be too much slower. ** On the other hand, the extra seeks could end up being significantly ** more expensive.
label: code-design
80241. FTS3 table handle
80242. We got the time
80243. Buffer to read position list from
80244. **comment:** previously SQLITE_CONFIG_CHUNKALLOC 12 which is now unused.
label: code-design
80245. 109
80246. Results of parsing argc/argv
80247. Check that the virtual-table is not already being initialized
80248. True for a prefix term
80249. ** Allocate a new hash table.
80250. ** Drop every cache entry whose page number is greater than "pgno". The ** caller must ensure that there are no outstanding references to any pages ** other than page 1 with a page number greater than pgno. ** ** If there is a reference to page 1 and the pgno parameter passed to this ** function is 0, then the data area associated with page 1 is zeroed, but ** the page object is not dropped.
80251. Pager state (OPEN, READER, WRITER_LOCKED..)
80252. If $>= 0$, filter for this column
80253. Record a pointer to the logger function and its first argument. ** The default is NULL. Logging is disabled if the function pointer is ** NULL.
80254. Absolute level of input segments
80255. True if arguments are desc
80256. 216
80257. ** json_type(JSON) ** json_type(JSON, PATH) ** ** Return the top-level "type" of a JSON string. Throw an error if ** either the JSON or PATH inputs are not well-formed.
80258. ***** The "pragmas.h" include file is an automatically generated file that ** that includes the PragType_XXXX macro definitions and the aPragmaName[] ** object. This ensures that the aPragmaName[] table is arranged in ** lexicographical order to facilitate a binary search of the pragma name. ** Do not edit pragma.h directly. Edit and rerun the script in at ** ../tool/mkpragmatab.tcl.
80259. At the pager level, a statement transaction is a savepoint with ** an index greater than all savepoints created explicitly using ** SQL statements. It is illegal to open, release or rollback any ** such savepoints while the statement transaction savepoint is active.
80260. **comment:** This case could be removed without changing the results of running ** this code. Including it causes gcc to generate a faster switch ** statement (since the range of switch targets now starts at zero and ** is contiguous) but does not cause any duplicate code to be generated ** (as gcc is clever enough to combine the two like cases). Other ** compilers might be similar.
label: code-design
80261. First absolute level for iLangid/iIndex
80262. ** 2014 May 31 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. **
May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **
***** Low level access to the FTS index stored in the database file. The ** routines in this file implement all read and write access to the ** %_data table. Other parts of the system access this functionality via ** the interface defined in fts5Int.h.
80263. ***** Continuing where we left off in sqlite3rbu.c *****
80264. Try to avoid a seek in BtreeInsert()
80265. Number of function parameters
80266. **comment:** If necessary, remove sample iMin to make room for the new sample.
label: code-design
80267. SQLITE_USE_FCNTL_TRACE
80268. GOTO label for next WHEN clause
80269. 470
80270. Fill in the abNotindexed array
80271. Total number of segments on level
80272. Ok to return a row more than once
80273. Clear any prior statistics
80274. Used to iterate through aAll
80275. #define SQLITE_ABORT 4 // Also an error code
80276. Page number to read
80277. If the PGHDR_NEED_SYNC flag is set for any of the nPage pages ** starting at pg1, then it needs to be set for all of them. Because ** writing to any of these nPage pages may damage the others, the ** journal file must contain sync(ed) copies of all of them ** before any of them can be written out to the database file.
80278. SQLITE_DEBUG - the sqlite3AssertMayAbort() function
80279. List of nodes removed during a CondenseTree operation. List is ** linked together via the pointer normally used for hash chains - ** RtreeNode.pNext.
RtreeNode.iNode stores the depth of the sub-tree ** headed by the node (leaf nodes have RtreeNode.iNode==0).
80280. Text of the token
80281. The destructor
80282. LCS value for this column
80283. Bytes of memory for in-memory hash
80284. OUT: User data for *pxFunc
80285. Index of the new VdbeCursor
80286. If true, LHS of IN operator is a rowid
80287. This test function is not currently used by the automated test-suite. ** Hence it is only available in debug builds.
80288. ** Add a new element to the pAggInfo->aCol[] array. Return the index of ** the new element. Return a negative number if malloc fails.
80289. One of the PAGER_JOURNALMODE_* values
80290. Recovery action to do in case of an error
80291. Opcode: AddImm P1 P2 * * * ** Synopsis: r[P1]=r[P1]+P2 ** ** Add the constant P2 to the value in register P1. ** The result is always an integer. ** ** To force any register to be an integer, just add 0.
80292. Agg finalizer
80293. 154
80294. refarg ::= ON UPDATE refact
80295. col
80296. First argument to xInput
80297. ** This function registered all of the above C functions as SQL ** functions. This should be the only routine in this file with ** external linkage.

80298. ** This function is an fts3ExprIterate() callback used by sqlite3Fts3Offsets().
80299. Number of u32 elements in match-info
80300. ** Information is passed from codeCursorHint() down to individual nodes of ** the expression tree (by sqlite3WalkExpr()) using an instance of this ** structure.
80301. This call is Ok even if this savepoint is actually a transaction ** savepoint (and therefore should not prompt xSavepoint() callbacks. ** If this is a transaction savepoint being opened, it is guaranteed ** that the db->aVTrans[] array is empty.
80302. Number of objects on pFresh
80303. Inserting a single row into a parent table cannot cause (or fix) ** an immediate foreign key violation. So do nothing in this case.
80304. INSTEAD => ID
80305. True if inside write transaction
80306. The progress callback
80307. ** Return true if, for the purposes of tokenizing with the tokenizer ** passed as the first argument, codepoint iCode is considered a token ** character (not a separator).
80308. Set iBlock to the index of the block pointed to by pOld in ** the array of mem5.szAtom byte blocks pointed to by mem5.zPool.
80309. ***** Begin file util.c *****
80310. Do not allow a transition to journal_mode=WAL for a database ** in temporary storage or if the VFS does not support shared memory
80311. ** Names of the various JSON types:
80312. Pointer to buffer
80313. Used when p4type is P4_FUNCCTX
80314. Integer 0
80315. Node pPhrase belongs to
80316. Do substitutes on p->pPrior too
80317. the prepared statement
80318. Name of system call to override
80319. ***** End of sqlite3session.c *****
80320. ** Lock the file with the lock specified by parameter eFileLock - one ** of the following: ** ** (1) SHARED_LOCK ** (2) RESERVED_LOCK ** (3) PENDING_LOCK ** (4) EXCLUSIVE_LOCK ** ** Sometimes when requesting one lock state, additional lock states ** are inserted in between. The locking might fail on one of the later ** transitions leaving the lock state different from what it started but ** still short of its goal. The following chart shows the allowed ** transitions and the inserted intermediate states: ** ** UNLOCKED -> SHARED ** SHARED -> RESERVED ** SHARED -> (PENDING) -> EXCLUSIVE ** RESERVED -> (PENDING) -> EXCLUSIVE ** PENDING -> EXCLUSIVE ** ** This routine will only increase a lock. Use the sqlite3OsUnlock() ** routine to lower a locking level. ** ** With dotfile locking, we really only support state (4): EXCLUSIVE. ** But we track the other locking levels internally.
80321. Pointer to buffer containing name
80322. The level of the FROM clause we are working on
80323. in case we need to try again for an :auto: named lock file
80324. "view", "trigger", or "index"
80325. Loop through table entries that match term pOrTerm.
80326. The pSpace buffer will be freed after the next call to ** balance_nonroot(), or just before this function returns, whichever ** comes first.
80327. Unix file with desc used in the key
80328. Update nBackfill
80329. tcomma ::= COMMA
80330. Initialize the PCache object
80331. Length of the apDef[] list
80332. ** xNext - Advance the cursor to the next row, if any.
80333. Index of column to read value from
80334. The specific database being pragmamed
80335. This condition is true when pItem is the FROM clause term on the ** right-hand-side of a LEFT or CROSS JOIN.
80336. Opcode: If P1 P2 P3 * * * Jump to P2 if the value in register P1 is true. The value ** is considered true if it is numeric and non-zero. If the value ** in P1 is NULL then take the jump if and only if P3 is non-zero.
80337. ** Append content to pParse that will complete zPath. Return a pointer ** to the inserted node, or return NULL if the append fails.
80338. ** Set or clear the access authorization function. ** ** The access authorization function is be called during the compilation ** phase to verify that the user has read and/or write access permission on ** various fields of the database. The first argument to the auth function ** is a copy of the 3rd argument to this routine. The second argument ** to the auth function is one of these constants: ** ** SQLITE_CREATE_INDEX ** SQLITE_CREATE_TABLE ** SQLITE_CREATE_TEMP_INDEX ** SQLITE_CREATE_TEMP_TABLE ** SQLITE_CREATE_TEMP_TRIGGER ** SQLITE_CREATE_TEMP_VIEW ** SQLITE_CREATE_TRIGGER ** SQLITE_CREATE_VIEW ** SQLITE_DELETE ** SQLITE_DROP_INDEX ** SQLITE_DROP_TABLE ** SQLITE_DROP_TEMP_INDEX ** SQLITE_DROP_TEMP_TABLE ** SQLITE_DROP_TEMP_TRIGGER ** SQLITE_DROP_TEMP_VIEW ** SQLITE_DROP_TRIGGER ** SQLITE_DROP_VIEW ** SQLITE_INSERT ** SQLITE_PRAGMA ** SQLITE_READ ** SQLITE_SELECT ** SQLITE_TRANSACTION ** SQLITE_UPDATE ** ** The third and fourth arguments to the auth function are the name of ** the table and the column that are being accessed. The auth function ** should return either SQLITE_OK, SQLITE_DENY, or SQLITE_IGNORE. If ** SQLITE_OK is returned, it means that access is allowed. SQLITE_DENY ** means that the SQL statement will never-run - the sqlite3_exec() call ** will return with an error. SQLITE_IGNORE means that the SQL statement ** should run but attempts to read the specified column will return NULL ** and attempts to write the column will be ignored. ** ** Setting the auth function to NULL disables this hook. The default ** setting of the auth function is NULL.
80339. ** Write mapping (iRowid->iNode) to the <rtree>_rowid table.
80340. Docid for entry to add
80341. Tables in outer loops of the join
80342. Number of entries in the hash table
80343. **comment:** ** Implementation of a scalar function that decodes r-tree nodes to ** human readable strings. This can be used for debugging and analysis. ** ** The scalar function takes two arguments: (1) the number of dimensions ** to the rtree (between 1 and 5, inclusive) and (2) a blob of data containing ** an r-tree node. For a two-dimensional r-tree structure called "rt", to ** deserialize all nodes, a statement like: ** ** SELECT rtreenode(2, data) FROM rt_node; ** ** The human readable string takes the form of a Tcl list with one ** entry for each cell in the r-tree node. Each entry is itself a ** list, containing the 8-byte rowid/pageno followed by the ** <num-dimension>*2 coordinates.
label: code-design
80344. Find new rows
80345. ** A different comparison function for SegReader structures. In this ** version, it is assumed that each SegReader points to an entry in ** a doclist for identical terms. Comparison is made as follows: ** ** 1) EOF (end of doclist in this case) is greater than not EOF. ** ** 2) By current docid. ** ** 3) By segment age. An older segment is considered larger.
80346. Check that a table or index named 'zName' does not already exist ** in database iDb. If so, this is an error.
80347. Leave any error code here
80348. Est. number of rows where key is less than this sample
80349. Effective size of the main journal
80350. ** Locate the in-memory structure that describes a particular database ** table given the name of that table and (optionally) the name of the ** database containing the table. Return NULL if not found. ** ** If zDatabase is 0, all databases are searched for the table and the ** first matching table is returned. (No checking for duplicate table ** names is done.) The search order is TEMP first, then MAIN, then any ** auxiliary databases added using the ATTACH command. ** ** See also sqlite3LocateTable().
80351. The VDBE cursor used to access the table
80352. seltablist ::= stl_prefix LP select RP as on_opt using_opt
80353. Reader object
80354. No encoding change
80355. ** Move the current position of the file handle passed as the first ** argument to offset iOffset within the file. If successful, return 0. ** Otherwise, set pFile->lastErno and return non-zero.

80356. Number of pages in the database
80357. expr ::= expr NOT expr
80358. ** Generate VDBE code for the statements inside the body of a single ** trigger.
80359. allocated even if there is no FROM clause
80360. **comment:** ** Helper function for sqlite3Error() - called rarely. Broken out into ** a separate routine to avoid unnecessary register saves on entry to **
sqlite3Error().
label: code-design
80361. ** json_set(JSON, PATH, VALUE, ...) ** ** Set the value at PATH to VALUE. Create the PATH if it does not already ** exist. Overwrite existing values that do exist. ** If JSON or PATH is malformed, throw an error. ** ** json_insert(JSON, PATH, VALUE, ...) ** ** Create PATH and initialize it to VALUE. If PATH already exists, this ** routine is a no-op. If JSON or PATH is malformed, throw an error.
80362. For a single-column UNIQUE index, once we have found a non-NULL ** row, we know that all the rest will be distinct, so skip ** subsequent distinctness tests.
80363. pPk is the PRIMARY KEY index for WITHOUT ROWID tables and NULL for ** normal rowid tables. nPkField is the number of key fields in the ** pPk index or 1 for a rowid table. In other words, nPkField is the ** number of fields in the true primary key of the table.
80364. ** Read a 64-bit variable-length integer from memory starting at p[0]. ** Return the number of bytes read. The value is stored in *v.
80365. If successful, populate the output variables. Otherwise, zero them and ** free any allocation made. An error code will be returned in this case.
80366. ** Routines to implement the count() aggregate function.
80367. Current table
80368. Allocate and initialize space for aTo, aFrom and aSortCost[]
80369. ** Return non-zero if the table pTab in database iDb or any of its indices ** have been opened at any point in the VDBE program. This is used to see if ** a statement of the form "INSERT INTO <iDb, pTab> SELECT ..." can ** run without using a temporary table for the results of the SELECT.
80370. Previous row data is stored here
80371. no, really unlock.
80372. nArg
80373. Source of new entry objects
80374. caller will handle out of memory
80375. ** If the RBU database contains the rbu_count table, use it to initialize ** the sqlite3rbu.nPhaseOneStep variable. The schema of the rbu_count table ** is assumed to contain the same columns as: ** ** CREATE TABLE rbu_count(tbl TEXT PRIMARY KEY, cnt INTEGER) WITHOUT ROWID; ** ** There should be one row in the table for each data_xxx table in the ** database. The 'tbl' column should contain the name of a data_xxx table, ** and the cnt column the number of rows it contains. ** ** sqlite3rbu.nPhaseOneStep is initialized to the sum of (1 + nIndex) * cnt ** for all rows in the rbu_count table, where nIndex is the number of ** indexes on the corresponding target database table.
80376. Used by: collation_list
80377. Largest possible freeblock offset
80378. The following state variables are used as part of the incremental ** checkpoint stage (eStage==RBU_STAGE_CKPT). See comments surrounding ** function rbuSetupCheckpoint() for details.
80379. Size of buffer pRec in bytes
80380. Array of overflow page numbers
80381. ***** End of alter.c *****
80382. Maximum NNN value for this transform
80383. OUT: error message
80384. The next page to be freed
80385. **comment:** ** 2003 September 6 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** This is the header file for information that is private to the ** VDBE. This information used to all be at the top of the single ** source code file "vdbe.c". When that file became too big (over ** 6000 lines long) it was split up into several smaller files and ** this header information was factored out.
label: code-design
80386. .pScopyFrom =
80387. Number of db changes made since last reset
80388. **comment:** ** Populate the low-level memory allocation function pointers in ** sqlite3GlobalConfig.m with pointers to the routines in this file. The ** arguments specify the block of memory to manage. ** ** This routine is only called by sqlite3_config(), and therefore ** is not required to be threadsafe (it is not).
label: code-design
80389. SQL statement for the OP_SqlExec opcode
80390. First 8 bytes will be zeroed
80391. ** Advance a stmt_cursor to its next row of output.
80392. ** An object of this type contains the state required to create or append ** to an appendable b-tree segment.
80393. Case 1: The table is a virtual-table. Use the VFilter and VNext ** to access the data.
80394. Non-zero if this is a v-tab with an xRename()
80395. ** Perform a read or write operation on a blob
80396. ** A bit in a Bitmask
80397. ** iMaster is the index of the master chunk. Most new allocations ** occur off of this chunk. szMaster is the size (in Mem3Blocks) ** of the current master. iMaster is 0 if there is not master chunk. ** The master chunk is not in either the aiHash[] or aiSmall[].
80398. Set the new 'max-root-page' value in the database header. This ** is the old value less one, less one more if that happens to ** be a root-page number, less one again if that is the ** PENDING_BYTE_PAGE.
80399. create a new path by replace the trailing '-conch' with '-break'
80400. If the cursor does not yet have a statement handle, obtain one now.
80401. ** This macro is used inside of assert() statements to indicate that ** the assert is only valid on a well-formed database. Instead of: ** ** assert(X); ** ** One writes: ** ** assert(X || CORRUPT_DB); ** ** CORRUPT_DB is true during normal operation. CORRUPT_DB does not indicate ** that the database is definitely corrupt, only that it might be corrupt. ** For most test cases, CORRUPT_DB is set to false using a special ** sqlite3_test_control(). This enables assert() statements to prove ** things that are always true for well-formed databases.
80402. 290
80403. ** Return true if the Mem object contains a TEXT or BLOB that is ** too large - whose size exceeds SQLITE_MAX_LENGTH.
80404. ** Copy the current sorter key into the memory cell pOut.
80405. Number of hash collisions
80406. Resolve the column names in all the expressions in the ** WHERE clause.
80407. ** current_date() ** ** This function returns the same value as date('now').
80408. ** Deprecated external interface. It used to set an alarm callback ** that was invoked when memory usage grew too large. Now it is a ** no-op.
80409. ** Advance iterator pIter to the next entry. ** ** This version of fts5SegIterNext() is only used if detail=none and the ** iterator is not a reverse direction iterator.
80410. ** Make sure we can call this stuff from C++.
80411. ***** Include sqlite3.h in the middle of sqliteInt.h *****
80412. COLUMN
80413. ***** Begin file mutex_w32.c *****
80414. Handle for sqlite3DbMallocZero() (may be null)
80415. same as TK_PLUS, synopsis: r[P3]=r[P1]+r[P2]
80416. The SHM file is readonly
80417. True if table b-trees. False for index b-trees
80418. Number of matching table names
80419. Columns in zFrom.zTbl
80420. Btree cursor used for pTab

80421. A buffer to hold the data for the new record
80422. d0..d7
80423. ** Set the collating sequence for expression pExpr to be the collating ** sequence named by pToken. Return a pointer to a new Expr node that ** implements the COLLATE operator. *** If a memory allocation error occurs, that fact is recorded in pParse->db ** and the pExpr parameter is returned unchanged.
80424. 31
80425. IMPLEMENTATION-OF: R-61436-13639 If the argument N is negative, then ** the number of cache pages is adjusted to use approximately abs(N*1024) ** bytes of memory.
80426. space allocated at zToken
80427. Race condition here: Another process might have been holding the ** the RESERVED lock and have a journal open at the sqlite3OsAccess() ** call above, but then delete the journal and drop the lock before ** we get to the following sqlite3OsCheckReservedLock() call. If that ** is the case, this routine might think there is a hot journal when ** in fact there is none. This results in a false-positive which will ** be dealt with by the playback routine. Ticket #3883.
80428. **comment:** The GCC_VERSION and MSVC_VERSION macros are used to ** conditionally include optimizations for each of these compilers. A ** value of 0 means that compiler is not being used. The ** SQLITE_DISABLE_INTRINSIC macro means do not use any compiler-specific ** optimizations, and hence set all compiler macros to 0 ** ** There was once also a CLANG_VERSION macro. However, we learn that the ** version numbers in clang are for "marketing" only and are inconsistent ** and unreliable. Fortunately, all versions of clang also recognize the ** gcc version numbers and have reasonable settings for gcc version numbers, ** so the GCC_VERSION macro will be set to a correct non-zero value even ** when compiling with clang.
label: code-design
80429. Zero or more UNIXFILE_* values
80430. Template WhereLoop
80431. Session object that owns pTab
80432. Mask of inner loops
80433. Opcode: OpenAutoindex P1 P2 * P4 * ** Synopsis: nColumn=P2 ** ** This opcode works the same as OP_OpenEphemeral. It has a ** different name to distinguish its use. Tables created using ** by this opcode will be used for automatically created transient ** indices in joins.
80434. Error message if rc!=SQLITE_OK
80435. Next database page to write
80436. Do not try to use skip-scan if true
80437. A connection to shared-memory
80438. synopsis: intkey=P3 data=r[P2]
80439. True if the table is not BTREE_UNORDERED
80440. "x IN (SELECT ...)": TUNING: the SELECT returns 25 rows
80441. Trying to read or write past the end of the data is an error. The ** conditional above is really: ** &aPayload[pCur->info.nLocal] > &pPage->aData[pBt->usableSize] ** but is recast into its current form to avoid integer overflow problems
80442. If the LIMIT is less than or equal to zero, loop forever. This ** is documented. But also, if the LIMIT+OFFSET exceeds 2^63 then ** also loop forever. This is undocumented. In fact, one could argue ** that the loop should terminate. But assuming 1 billion iterations ** per second (far exceeding the capabilities of any current hardware) ** it would take nearly 300 years to actually reach the limit. So ** looping forever is a reasonable approximation.
80443. ***** End of sqlite3rbu.h *****
80444. 1080
80445. Step 2.
80446. We failed long ago if this is not so
80447. sqlite3_step() has finished executing
80448. Opcode: IdxLT P1 P2 P3 P4 P5 ** Synopsis: key=r[P3@P4] ** ** The P4 register values beginning with P3 form an unpacked index ** key that omits the PRIMARY KEY or ROWID. Compare this key value against ** the index that P1 is currently pointing to, ignoring the PRIMARY KEY or ** ROWID on the P1 index. *** If the P1 index entry is less than the key value then jump to P2. ** Otherwise fall through to the next instruction.
80449. If an xFilter() callback was specified, invoke it now. If the ** xFilter callback returns zero, skip this table. If it returns ** non-zero, proceed.
80450. ***** Routines with file scope above. Interface to the rest of the where.c ** subsystem follows. *****
80451. The check expression
80452. One or more tables used to resolve names
80453. See ticket [98d973b8f5]
80454. setlist ::= setlist COMMA LP idlist RP EQ expr
80455. Temp register to free before returning
80456. ** Streaming versions of changegroup_output().
80457. ***** Begin file fkey.c *****
80458. ** Report the allocated size of a prior return from xMalloc() ** or xRealloc().
80459. Total number of bytes on a page
80460. WHERE
80461. ** Delete a "ascii" tokenizer.
80462. Value might be numeric
80463. **comment:** ** The %_segments table is declared as follows: ** ** CREATE TABLE %_segments(blockid INTEGER PRIMARY KEY, block BLOB) ** ** This function reads data from a single row of the %_segments table. The ** specific row is identified by the iBlockid parameter. If paBlob is not ** NULL, then a buffer is allocated using sqlite3_malloc() and populated ** with the contents of the blob stored in the "block" column of the ** identified table row is. Whether or not paBlob is NULL, *pnBlob is set ** to the size of the blob in bytes before returning. ** ** If an error occurs, or the table does not contain the specified row, ** an SQLite error code is returned. Otherwise, SQLITE_OK is returned. If ** paBlob is non-NULL, then it is the responsibility of the caller to ** eventually free the returned buffer. ** ** This function may leave an open sqlite3_blob* handle in the ** Fts3Table.pSegments variable. This handle is reused by subsequent calls ** to this function. The handle may be closed by calling the ** sqlite3Fts3SegmentsClose() function. Reusing a blob handle is a handy ** performance improvement, but the blob handle should always be closed ** before control is returned to the user (to prevent a lock being held ** on the database file for longer than necessary). Thus, any virtual table ** method (xFilter etc.) that may directly or indirectly call this function ** must call sqlite3Fts3SegmentsClose() before returning.
label: code-design
80464. Length of zInput in bytes
80465. xBestIndex
80466. True for DESC. False for ASC.
80467. Indicates a pending lock has been obtained
80468. Low accuracy coordinate
80469. Rowid argument passed to stat_push()
80470. Used to set error string on failure.
80471. Hash table representation
80472. Foreign key support may not be enabled or disabled while not ** in auto-commit mode.
80473. Max leading 0 in anEq[] for any a[] entry
80474. The "x IN (SELECT rowid FROM table)" case
80475. Size of the cell being deleted
80476. If there are outstanding locks, do not actually close the file just ** yet because that would clear those locks. Instead, add the file ** descriptor to pInode->aPending. It will be automatically closed when ** the last lock is cleared.
80477. **comment:** ** This object contains information needed to implement a single nested ** loop in WHERE clause. *** Contrast this object with WhereLoop. This object describes the ** implementation of the loop. WhereLoop describes the algorithm. ** This object contains a pointer to the WhereLoop algorithm as one of ** its elements. *** The WhereInfo object contains a single instance of this object for ** each term in the FROM clause (which is to say, for each of the ** nested loops as implemented). The order of WhereLevel objects determines ** the loop nested order, with WhereInfo.a[0] being the outer loop and ** WhereInfo.a[WhereInfo.nLevel-1] being the inner loop.

label: code-design

80478. WHERE clause to scan with

80479. ***** Begin %parse_accept code *****

80480. **comment:** Not needed. Silence a compiler warning.**label:** requirement

80481. aKWLlen[i] is the length (in bytes) of the i-th keyword

80482. Execute the recursive SELECT taking the single row in Current as ** the value for the recursive-table. Store the results in the Queue.

80483. ** Memory available for allocation

80484. The request was for a RESERVED or EXCLUSIVE lock. It is ** assumed that there is a SHARED or greater lock on the file ** already.

80485. Name of the function, zero-terminated

80486. ** If compiling for processor that lacks floating point support, ** substitute integer for floating-point.

80487. ** CAPI3REF: Test For Auto-Commit Mode ** KEYWORDS: {autocommit mode} ** METHOD: sqlite3_get_autocommit() interface returns non-zero or ** zero if the given database connection is or is not in autocommit mode, ** respectively. ^Autocommit mode is on by default. ** ^Autocommit mode is disabled by a [BEGIN] statement. ** ^Autocommit mode is re-enabled by a [COMMIT] or [ROLLBACK]. ** ** If certain kinds of errors occur on a statement within a multi-statement ** transaction (errors including [SQLITE_FULL], [SQLITE_IOERR], ** [SQLITE_NOMEM], [SQLITE_BUSY], and [SQLITE_INTERRUPT]) then the ** transaction might be rolled back automatically. The only way to ** find out whether SQLite automatically rolled back the transaction after ** an error is to use this function. ** ** If another thread changes the autocommit status of the database ** connection while this routine is running, then the return value ** is undefined.

80488. ** CAPI3REF: Retrieve the mutex for a database connection ** METHOD: sqlite3 ** ** ^This interface returns a pointer to the [sqlite3_mutex] object that ** serializes access to the [database connection] given in the argument ** when the [threading mode] is Serialized. ** ^If the [threading mode] is Single-thread or Multi-thread then this ** routine returns a NULL pointer.

80489. Object completely outside of query region

80490. **comment:** ** CAPI3REF: Setting The Result Of An SQL Function ** METHOD: sqlite3_context ** ** These routines are used by the xFunc or xFinal callbacks that ** implement SQL functions and aggregates. See ** [sqlite3_create_function()] and [sqlite3_create_function16()] ** for additional information. ** ** These functions work very much like the [parameter binding] family of ** functions used to bind values to host parameters in prepared statements. ** Refer to the [SQL parameter] documentation for additional information. ** ** ^The sqlite3_result_blob() interface sets the result from ** an application-defined function to be the BLOB whose content is pointed ** to by the second parameter and which is N bytes long where N is the ** third parameter. ** ** ^The sqlite3_result_zeroblob(C,N) and sqlite3_result_zeroblob64(C,N) ** interfaces set the result of the application-defined function to be ** a BLOB containing all zero bytes and N bytes in size. ** ** ^The sqlite3_result_double() interface sets the result from ** an application-defined function to be a floating point value specified ** by its 2nd argument. ** ** ^The sqlite3_result_error() and sqlite3_result_error16() functions ** cause the implemented SQL function to throw an exception. ** ^SQLite uses the string pointed to by the ** 2nd parameter of sqlite3_result_error() or sqlite3_result_error16() ** as the text of an error message. ^SQLite interprets the error ** message string from sqlite3_result_error() as UTF-8. ^SQLite ** interprets the string from sqlite3_result_error16() as UTF-16 in native ** byte order. ^If the third parameter to sqlite3_result_error() ** or sqlite3_result_error16() is negative then SQLite takes as the error ** message all text up through the first zero character. ** ^If the third parameter to sqlite3_result_error() or ** sqlite3_result_error16() is non-negative then SQLite takes that many ** bytes (not characters) from the 2nd parameter as the error message. ** ^The sqlite3_result_error() and sqlite3_result_error16() ** routines make a private copy of the error message text before ** they return. Hence, the calling function can deallocate or ** modify the text after they return without harm. ** ^The sqlite3_result_error_code() function changes the error code ** returned by SQLite as a result of an error in a function. ^By default, ** the error code is SQLITE_ERROR. ^A subsequent call to sqlite3_result_error() ** or sqlite3_result_error16() resets the error code to SQLITE_ERROR. ** ** ^The sqlite3_result_error_toobig() interface causes SQLite to throw an ** error indicating that a string or BLOB is too long to represent. ** ** ^The sqlite3_result_error_nomem() interface causes SQLite to throw an ** error indicating that a memory allocation failed. ** ** ^The sqlite3_result_int() interface sets the return value ** of the application-defined function to be the 32-bit signed integer ** value given in the 2nd argument. ** ^The sqlite3_result_int64() interface sets the return value ** of the application-defined function to be the 64-bit signed integer ** value given in the 2nd argument. ** ** ^The sqlite3_result_null() interface sets the return value ** of the application-defined function to be NULL. ** ** ^The sqlite3_result_text(), sqlite3_result_text16(), ** sqlite3_result_text16e(), and sqlite3_result_text16be() interfaces ** set the return value of the application-defined function to be ** a text string which is represented as UTF-8, UTF-16 native byte order, ** UTF-16 little endian, or UTF-16 big endian, respectively. ** ^The sqlite3_result_text64() interface sets the return value of an ** application-defined function to be a text string in an encoding ** specified by the fifth (and last) parameter, which must be one ** of [SQLITE_UTF8], [SQLITE_UTF16], [SQLITE_UTF16BE], or [SQLITE_UTF16LE]. ** ^SQLite takes the text result from the application from ** the 2nd parameter of the sqlite3_result_text* interfaces. ** ^If the 3rd parameter to the sqlite3_result_text* interfaces ** is negative, then SQLite takes result text from the 2nd parameter ** through the first zero character. ** ^If the 3rd parameter to the sqlite3_result_text* interfaces ** is non-negative, then as many bytes (not characters) of the text ** pointed to by the 2nd parameter are taken as the application-defined ** function result. If the 3rd parameter is non-negative, then it ** must be the byte offset into the string where the NUL terminator would ** appear if the string where NUL terminated. If any NUL characters occur ** in the string at a byte offset that is less than the value of the 3rd ** parameter, then the resulting string will contain embedded NULs and the ** result of expressions operating on strings with embedded NULs is undefined. ** ^If the 4th parameter to the sqlite3_result_text* interfaces ** or sqlite3_result_blob is a non-NULL pointer, then SQLite calls that ** function as the destructor on the text or BLOB result when it has ** finished using that result. ** ^If the 4th parameter to the sqlite3_result_text* interfaces or to ** sqlite3_result_blob is the special constant SQLITE_STATIC, then SQLite ** assumes that the text or BLOB result is in constant space and does not ** copy the content of the parameter nor call a destructor on the content ** when it has finished using that result. ** ^If the 4th parameter to the sqlite3_result_text* interfaces ** or sqlite3_result_blob is the special constant SQLITE_TRANSIENT ** then SQLite makes a copy of the result into space obtained ** from [sqlite3_malloc()] before it returns. ** ** ^The sqlite3_result_value() interface sets the result of ** the application-defined function to be a copy of the [sqlite3_value] object specified by the 2nd parameter. ^The ** sqlite3_result_value() interface makes a copy of the [sqlite3_value] ** so that the [sqlite3_value] specified in the parameter may change or ** be deallocated after sqlite3_result_value() returns without harm. ** ^A [protected] sqlite3_value] object may always be used where an ** [unprotected sqlite3_value] object is required, so either ** kind of [sqlite3_value] object can be used with this interface. ** ** ^The sqlite3_result_pointer(C,P,T,D) interface sets the result to an ** SQL NULL value, just like [sqlite3_result_null(C)], except that it ** also associates the host-language pointer P or type T with that ** NULL value such that the pointer can be retrieved within an ** [application-defined SQL function] using [sqlite3_value_pointer()]. ** ^If the D parameter is not NULL, then it is a pointer to a destructor ** for the P parameter. ^SQLite invokes D with P as its only argument ** when SQLite is finished with P. The T parameter should be a static ** string and preferably a string literal. The sqlite3_result_pointer() ** routine is part of the [pointer passing interface] added for SQLite 3.20.0. ** ** If these routines are called from within the different thread ** than the one containing the application-defined function that received ** the [sqlite3_context] pointer, the results are undefined.**label:** code-design

80491. ** CAPI3REF: OS Interface Open File Handle ** ** An [sqlite3_file] object represents an open file in the ** [sqlite3_vfs | OS interface layer]. Individual OS interface ** implementations will ** want to subclass this object by appending additional fields ** for their own use. The pMethods entry is a pointer to an ** [sqlite3_io_methods] object that defines methods for performing ** I/O operations on the open file.

80492. **comment:** ** EXTENSION API FUNCTIONS ** ** xUserData(pFts): ** Return a copy of the context pointer the extension function was ** registered with. ** ** xColumnTotalSize(pFts, iCol, pnToken): ** If parameter iCol is less than zero, set output variable *pnToken ** to the total number of tokens in the FTS5 table. Or, if iCol is ** non-negative but less than the number of columns in the table, return ** the total number of tokens in column iCol, considering all rows in ** the FTS5 table. ** ** If parameter iCol is greater than or equal to the number of columns ** in the table, SQLITE_RANGE is returned. Or, if an error occurs (e.g. ** an OOM condition or IO error), an appropriate SQLite error code is ** returned. ** ** xColumnCount(pFts): ** Return the number of columns in the table. ** ** xColumnSize(pFts, iCol, pnToken): ** If parameter iCol is less than zero, set output variable *pnToken ** to the total number of tokens in the current row. Or, if iCol is ** non-negative but less than the number of columns in the table, set ** *pnToken to the number of tokens in column iCol of the current row. ** ** If parameter iCol is greater than or equal to the number of columns ** in the table, SQLITE_RANGE is returned. Or, if an error occurs (e.g. ** an OOM condition or IO error), an appropriate SQLite error code is ** returned. ** ** This function may be quite inefficient if used with an FTS5 table ** created with the "columnsize=0" option. ** ** xColumnText: ** This function attempts to retrieve the text of column iCol of the ** current document. If successful, (*pz) is set to point to a buffer ** containing the text in utf-8 encoding, (*pn) is set to the size in bytes ** (not characters) of the buffer and SQLITE_OK is returned. Otherwise, ** if an error occurs, an SQLite error code is returned and the final values ** of (*pz) and (*pn) are undefined. ** ** xPhraseCount: ** Returns the number of phrases in the current query expression. ** ** xPhraseSize: ** Returns the number of tokens in phrase iPhrase of the query. Phrases ** are numbered starting from zero. ** ** xInstCount: ** Set *pnInst to the total number of occurrences of all phrases within ** the query within the current row. Return SQLITE_OK if successful, or ** an error code (i.e. SQLITE_NOMEM) if an error occurs. ** ** This API can be quite slow if used with an FTS5 table created with the ** "detail=None" or "detail=column" option. If the FTS5 table is created ** with either "detail=None" or "detail=column" and "content=" option ** (i.e. if it is a contentless table), then this API always returns 0. ** ** xInst: ** Query for the details of phrase match iIdx within the current row. ** Phrase matches are

numbered starting from zero, so the iIdx argument ** should be greater than or equal to zero and smaller than the value ** output by xInstCount(). *** Usually, output parameter *piPhrase is set to the phrase number, *piCol ** to the column in which it occurs and *piOff the token offset of the ** first token of the phrase. The exception is if the table was created ** with the offsets=0 option specified. In this case *piOff is always ** set to -1. *** Returns SQLITE_OK if successful, or an error code (i.e. SQLITE_NOMEM) ** if an error occurs. *** This API can be quite slow if used with an FTS5 table created with the ** "detail=None" or "detail=column" option. *** xRowid: ** Returns the rowid of the current row. *** xTokenize: ** Tokenize text using the tokenizer belonging to the FTS5 table. *** xQueryPhrase(pFts5, iPhrase, pUserData, xCallback): ** This API function is used to query the FTS table for phrase iPhrase ** of the current query. Specifically, a query equivalent to: *** ... FROM ftstable WHERE ftstable MATCH \$p ORDER BY rowid ** with \$p set to a phrase equivalent to the phrase iPhrase of the ** current query is executed. Any column filter that applies to ** phrase iPhrase of the current query is included in \$p. For each ** row visited, the callback function passed as the fourth argument ** is invoked. The context and API objects passed to the callback ** function may be used to access the properties of each matched row. ** Invoking Api.xUserData() returns a copy of the pointer passed as ** the third argument to pUserData. *** If the callback function returns any value other than SQLITE_OK, the ** query is abandoned and the xQueryPhrase function returns immediately. ** If the returned value is SQLITE_DONE, xQueryPhrase returns SQLITE_OK. ** Otherwise, the error code is propagated upwards. *** If the query runs to completion without incident, SQLITE_OK is returned. ** Or, if some error occurs before the query completes or is aborted by ** the callback, an SQLite error code is returned. *** *** xSetAuxdata(pFts5, pAux, xDelete) ** Save the pointer passed as the second argument as the extension functions ** "auxiliary data". The pointer may then be retrieved by the current or any ** future invocation of the same fts5 extension function made as part of ** of the same MATCH query using the xGetAuxdata() API. *** Each extension function is allocated a single auxiliary data slot for ** each FTS query (MATCH expression). If the extension function is invoked ** more than once for a single FTS query, then all invocations share a ** single auxiliary data context. *** If there is already an auxiliary data pointer when this function is ** invoked, then it is replaced by the new pointer. If an xDelete callback ** was specified along with the original pointer, it is invoked at this ** point. *** The xDelete callback, if one is specified, is also invoked on the ** auxiliary data pointer after the FTS5 query has finished. *** If an error (e.g. an OOM condition) occurs within this function, an ** the auxiliary data is set to NULL and an error code returned. If the ** xDelete parameter was not NULL, it is invoked on the auxiliary data ** pointer before returning. *** xGetAuxdata(pFts5, bClear) ** Returns the current auxiliary data pointer for the fts5 extension ** function. See the xSetAuxdata() method for details. *** If the bClear argument is non-zero, then the auxiliary data is cleared ** (set to NULL) before this function returns. In this case the xDelete, ** if any, is not invoked. *** xRowCount(pFts5, pnRow) ** This function is used to retrieve the total number of rows in the table. ** In other words, the same value that would be returned by: *** SELECT count(*) FROM ftstable; *** xPhraseFirst() ** This function is used, along with type Fts5PhraseIter and the xPhraseNext() ** method, to iterate through all instances of a single query phrase within ** the current row. This is the same information as is accessible via the ** xInstCount/xInst APIs. While the xInstCount/xInst APIs are more convenient ** to use, this API may be faster under some circumstances. To iterate ** through instances of phrase iPhrase, use the following code: *** Fts5PhraseIter iter; ** int iCol, iOff; ** for(pApi->xPhraseFirst(pFts, iPhrase, &iter, &iCol, &iOff); ** iCol>=0; ** pApi->xPhraseNext(pFts, &iter, &iCol, &iOff) **) { ** // An instance of phrase iPhrase at offset iOff of column iCol ** } *** The Fts5PhraseIter structure is defined above. Applications should not ** modify this structure directly - it should only be used as shown above ** with the xPhraseFirst() and xPhraseNext() API methods (and by ** xPhraseFirstColumn() and xPhraseNextColumn() as illustrated below). *** This API can be quite slow if used with an FTS5 table created with the ** "detail=None" or "detail=column" option. If the FTS5 table is created ** with either "detail=None" or "detail=column" and "content=" option ** (i.e. if it is a contentless table), then this API always iterates ** through an empty set (all calls to xPhraseFirst() set iCol to -1). *** xPhraseNext() ** See xPhraseFirst above. *** xPhraseFirstColumn() ** This function and xPhraseNextColumn() are similar to the xPhraseFirst() ** and xPhraseNext() APIs described above. The difference is that instead ** of iterating through all instances of a phrase in the current row, these ** APIs are used to iterate through the set of columns in the current row ** that contain one or more instances of a specified phrase. For example: *** Fts5PhraseIter iter; ** int iCol; ** for(pApi->xPhraseFirstColumn(pFts, iPhrase, &iter, &iCol); ** iCol>=0; ** pApi->xPhraseNextColumn(pFts, &iter, &iCol) **) { ** // Column iCol contains at least one instance of phrase iPhrase ** } *** This API can be quite slow if used with an FTS5 table created with the ** "detail=None" option. If the FTS5 table is created with either ** "detail=None" "content=" option (i.e. if it is a contentless table), ** then this API always iterates through an empty set (all calls to ** xPhraseFirstColumn() set iCol to -1). *** The information accessed using this API and its companion ** xPhraseFirstColumn() may also be obtained using xPhraseFirst/xPhraseNext ** (or xInst/xInstCount). The chief advantage of this API is that it is ** significantly more efficient than those alternatives when used with ** "detail=column" tables. *** xPhraseNextColumn() ** See xPhraseFirstColumn above.

label: code-design

80493. Methods above are valid for version 2

80494. int

80495. Source database name

80496. Pointer to application-specific data

80497. ** CAPI3REF: Obtaining SQL Values ** METHOD: sqlite3_value ** ** Summary: ** <blockquote><table border=0 cellpadding=0 cellspacing=0> ** <tr><td>sqlite3_value_blob</td>→;<td>BLOB value **
<td>sqlite3_value_double</td>→;<td>REAL value **
<td>sqlite3_value_int</td>→;<td>32-bit INTEGER value **
<td>sqlite3_value_int64</td>→;<td>64-bit INTEGER value **
<td>sqlite3_value_pointer</td>→;<td>Pointer value **
<td>sqlite3_value_text</td>→;<td>UTF-8 TEXT value **
<td>sqlite3_value_text16</td>→;<td>UTF-16 TEXT value in ** the native byteorder **
<td>sqlite3_value_text16be</td>→;<td>UTF-16be TEXT value **
<td>sqlite3_value_text16le</td>→;<td>UTF-16le TEXT value **
<td> <td> <td> <td>
<td>sqlite3_value_bytes</td>→;<td>Size of a BLOB ** or a UTF-8 TEXT in bytes **
<td>sqlite3_value_type</td>→;<td>Default ** datatype of the value **
<td>sqlite3_value_numeric_type</td>→;<td>Best numeric datatype of the value ** </table></blockquote> ** ** Details: ** ** These routines extract type, size, and content information from ** [protected sqlite3_value] objects. Protected sqlite3_value objects ** are used to pass parameter information into implementation of ** [application-defined SQL functions] and [virtual tables]. ** ** These routines work only with [protected sqlite3_value] objects. ** Any attempt to use these routines on an [unprotected sqlite3_value] ** is not threadsafe. ** ** ^These routines work just like the corresponding [column access functions] ** except that these routines take a single [protected sqlite3_value] object ** pointer instead of a [sqlite3_stmt] pointer and an integer column number. ** ** ^The sqlite3_value_text16() interface extracts a UTF-16 string ** in the native byte-order of the host machine. ^The ** sqlite3_value_text16be() and sqlite3_value_text16le() interfaces ** extract UTF-16 strings as big-endian and little-endian respectively. ** ** ^If [sqlite3_value] object V was initialized ** using [sqlite3_bind_pointer(S,I,P,X,D)] or [sqlite3_result_pointer(C,P,X,D)] ** and if X and Y are strings that compare equal according to strcmp(X,Y), ** then sqlite3_value_pointer(V,Y) will return the pointer P. ^Otherwise, ** sqlite3_value_pointer(V,Y) returns a NULL. The sqlite3_bind_pointer() ** routine is part of the [pointer passing interface] added for SQLite 3.20.0. ** ** ^^(The sqlite3_value_type(V) interface returns the ** [SQLITE_INTEGER | datatype code] for the initial datatype of the ** [sqlite3_value] object V. The returned value is one of [SQLITE_INTEGER], ** [SQLITE_FLOAT], [SQLITE_TEXT], [SQLITE_BLOB], or [SQLITE_NULL].)^ ** Other interfaces might change the datatype for an sqlite3_value object. ** For example, if the datatype is initially SQLITE_INTEGER and ** sqlite3_value_text(V) is called to extract a text value for that ** integer, then subsequent calls to sqlite3_value_type(V) might return ** SQLITE_TEXT. Whether or not a persistent internal datatype conversion ** occurs is undefined and may change from one release of SQLite to the next. ** ** ^^(The sqlite3_value_numeric_type() interface attempts to apply ** numeric affinity to the value. This means that an attempt is ** made to convert the value to an integer or floating point. If ** such a conversion is possible without loss of information (in other ** words, if the value is a string that looks like a number) ** then the conversion is performed. Otherwise no conversion occurs. ** The [SQLITE_INTEGER | datatype] after conversion is returned.)^ ** Please pay particular attention to the fact that the pointer returned ** from [sqlite3_value_blob()], [sqlite3_value_text()], or ** [sqlite3_value_text16()] can be invalidated by a subsequent call to ** [sqlite3_value_bytes()], [sqlite3_value_bytes16()], [sqlite3_value_text16()], ** or [sqlite3_value_text16()]. ** ** These routines must be called from the same thread as ** the SQL function that supplied the [sqlite3_value*] parameters.

80498. ** CAPI3REF: Extract Metadata About A Column Of A Table ** METHOD: sqlite3 *** ** ^^(The sqlite3_table_column_metadata(X,D,T,C,...) routine returns ** information about column C of table T in database D ** on [database connection] X.)^ ** The sqlite3_table_column_metadata() ** interface returns SQLITE_OK and fills in the non-NULL pointers in ** the final five arguments with appropriate values if the specified ** column exists. ^The sqlite3_table_column_metadata() interface returns ** SQLITE_ERROR and if the specified column does not exist. ** ^If the column-name parameter to sqlite3_table_column_metadata() is a ** NULL pointer, then this routine simply checks for the existence of the ** table and returns SQLITE_OK if the table exists and SQLITE_ERROR if it ** does not. If the table name parameter T in a call to ** sqlite3_table_column_metadata(X,D,T,C,...) is NULL then the result is ** undefined behavior. *** ^The column is identified by the second, third and fourth parameters to ** this function. ^^(The second parameter is either the name of the database ** (i.e. "main", "temp", or an attached database) containing the specified ** table or NULL.)^ ** If it is NULL, then all attached databases are searched ** for the table using the same algorithm used by the database engine to ** resolve unqualified table references. ** ** ^^(The third and fourth parameters to this function are the table and column ** name of the desired column, respectively. ** ** ^Metadata is returned by writing to the memory locations passed as the 5th ** and subsequent parameters to this function. ^Any of these arguments may be ** NULL, in which case the corresponding element of metadata is omitted. ** ** ^<blockquote> ** <table border="1"> **
<th> Parameter <th> Output
<th> Type <th> Description ** **
<td> 5th <td> const char* <td> Data type **
<td> 6th <td> const char* <td> Name of default collation sequence **
<td> 7th <td> int <td> True if column has a NOT NULL constraint **
<td> 8th <td> int <td> True if column is part of the PRIMARY KEY **
<td> 9th <td> int <td> True if column is [AUTOINCREMENT] ** </table> ** </blockquote>)*** ^^(The memory pointed to by the

character pointers returned for the ** declaration type and collation sequence is valid until the next ** call to any SQLite API function. *** ^If the specified table is actually a view, an [error code] is returned. *** ^If the specified column is "rowid", "oid" or "_rowid_" and the table ** is not a [WITHOUT ROWID] table and an ** [INTEGER PRIMARY KEY] column has been explicitly declared, then the output ** parameters are set for the explicitly declared column. ^If there is no ** [INTEGER PRIMARY KEY] column, then the outputs ** for the [rowid] are set as follows: *** <pre> ** data type: "INTEGER" ** collation sequence: "BINARY" ** not null: 0 ** primary key: 1 ** auto increment: 0 ** </pre> *** ^This function causes all database schemas to be read from disk and ** parsed, if that has not already been done, and returns an error if ** any errors are encountered while loading the schema.

80499. ** CAPI3REF: Test if a changeset has recorded any changes. *** Return non-zero if no changes to attached tables have been recorded by ** the session object passed as the first argument. Otherwise, if one or ** more changes have been recorded, return zero. *** Even if this function returns zero, it is possible that calling ** [sqlite3session_changeset()] on the session handle may still return a ** changeset that contains no changes. This can happen when a row in ** an attached table is modified and then later on the original values ** are restored. However, if this function returns non-zero, then it is ** guaranteed that a call to sqlite3session_changeset() will return a ** changeset containing zero changes.

80500. int *psz

80501. ** CAPI3REF: Text Encodings *** These constant define integer codes that represent the various ** text encodings supported by SQLite.

80502. ** CAPI3REF: Evaluate An SQL Statement ** METHOD: sqlite3_stmt ** After a [prepared statement] has been prepared using any of ** [sqlite3_prepare_v2()], [sqlite3_prepare_v3()], [sqlite3_prepare16_v2()] or ** or [sqlite3_prepare16_v3()] or one of the legacy ** interfaces [sqlite3_prepare()] or [sqlite3_prepare16()], this function ** must be called one or more times to evaluate the statement. *** The details of the behavior of the sqlite3_step() interface depend ** on whether the statement was prepared using the newer "vX" interfaces ** [sqlite3_prepare_v3()], [sqlite3_prepare_v2()], [sqlite3_prepare16_v3()] or ** [sqlite3_prepare16_v2()] or the older legacy ** interfaces [sqlite3_prepare()] and [sqlite3_prepare16()]. The use of the ** new "vX" interface is recommended for new applications but the legacy ** interface will continue to be supported. *** ^In the legacy interface, the return value will be either [SQLITE_BUSY], ** [SQLITE_DONE], [SQLITE_ROW], [SQLITE_ERROR], or [SQLITE_MISUSE]. ** ^With the "v2" interface, any of the other [result codes] or ** [extended result codes] might be returned as well. *** ^[SQLITE_BUSY] means that the database engine was unable to acquire the ** database locks it needs to do its job. ^If the statement is a [COMMIT] ** or occurs outside of an explicit transaction, then you can retry the ** statement. If the statement is not a [COMMIT] and occurs within an ** explicit transaction then you should rollback the transaction before ** continuing. *** ^[SQLITE_DONE] means that the statement has finished executing ** successfully. sqlite3_step() should not be called again on this virtual ** machine without first calling [sqlite3_reset()] to reset the virtual ** machine back to its initial state. *** ^If the SQL statement being executed returns any data, then [SQLITE_ROW] ** is returned each time a new row of data is ready for processing by the ** caller. The values may be accessed using the [column access functions]. ** sqlite3_step() is called again to retrieve the next row of data. *** ^[SQLITE_ERROR] means that a run-time error (such as a constraint ** violation) has occurred. sqlite3_step() should not be called again on ** the VM. More information may be found by calling [sqlite3_errmsg()]. ** ^With the legacy interface, a more specific error code (for example, ** [SQLITE_INTERRUPT], [SQLITE_SCHEMA], [SQLITE_CORRUPT], and so forth) ** can be obtained by calling [sqlite3_reset()] on the ** [prepared statement]. ^In the "v2" interface, ** the more specific error code is returned directly by sqlite3_step(). ** ^[SQLITE_MISUSE] means that this routine was called inappropriately. ** Perhaps it was called on a [prepared statement] that has ** already been [sqlite3_finalize | finalized] or on one that had ** previously returned [SQLITE_ERROR] or [SQLITE_DONE]. Or it could ** be the case that the same database connection is being used by two or ** more threads at the same moment in time. *** ^For all versions of SQLite up to and including 3.6.23.1, a call to ** [sqlite3_reset()] was required after sqlite3_step() returned anything ** other than [SQLITE_ROW] before any subsequent invocation of ** sqlite3_step(). Failure to reset the prepared statement using ** [sqlite3_reset()] would result in an [SQLITE_MISUSE] return from ** sqlite3_step(). But after [version 3.6.23.1] ([dateof:3.6.23.1]), ** sqlite3_step() began ** calling [sqlite3_reset()] automatically in this circumstance rather ** than returning [SQLITE_MISUSE]. This is not considered a compatibility ** break because any application that ever receives an [SQLITE_MISUSE] error ** is broken by definition. The [SQLITE_OMIT_AUTORESET] compile-time option ** can be used to restore the legacy behavior. *** Goofy Interface Alert: In the legacy interface, the sqlite3_step() ** API always returns a generic error code, [SQLITE_ERROR], following any ** error other than [SQLITE_BUSY] and [SQLITE_MISUSE]. You must call ** [sqlite3_reset()] or [sqlite3_finalize()] in order to find one of the ** specific [error codes] that better describes the error. ** We admit that this is a goofy design. The problem has been fixed ** with the "v2" interface. If you prepare all of your SQL statements ** using [sqlite3_prepare_v3()] or [sqlite3_prepare_v2()] ** or [sqlite3_prepare16_v2()] or [sqlite3_prepare16_v3()] instead ** of the legacy [sqlite3_prepare()] and [sqlite3_prepare16()] interfaces, ** then the more specific [error codes] are returned directly ** by sqlite3_step(). The use of the "vX" interfaces is recommended.

80503. ** CAPI3REF: String LIKE Matching * ** ^The [sqlite3_strlike(P,X,E)] interface returns zero if and only if ** string X matches the [LIKE] pattern P with escape character E. ** ^The definition of [LIKE] pattern matching used in ** [sqlite3_strlike(P,X,E)] is the same as for the "X LIKE P ESCAPE E" ** operator in the SQL dialect understood by SQLite. ^For "X LIKE P" without ** the ESCAPE clause, set the E parameter of [sqlite3_strlike(P,X,E)] to 0. ** ^As with the LIKE operator, the [sqlite3_strlike(P,X,E)] function is case ** insensitive - equivalent upper and lower case ASCII characters match ** one another. *** ^The [sqlite3_strlike(P,X,E)] function matches Unicode characters, though ** only ASCII characters are case folded. *** ^Note that this routine returns zero on a match and non-zero if the strings ** do not match, the same as [sqlite3_strcmp()] and [sqlite3_strncmp()]. *** See also: [sqlite3_strglob()].

80504. ** Specify the key for an encrypted database. This routine should be ** called right after sqlite3_open(). *** The code to implement this API is not available in the public release ** of SQLite.

80505. Database lock protocol error

80506. ** CAPI3REF: Checkpoint a database ** METHOD: sqlite3 *** ^The sqlite3_wal_checkpoint_v2(D,X,M,L,C) interface runs a checkpoint ** operation on database X of [database connection] D in mode M. Status ** information is written back into integers pointed to by L and C. ^ ** ^The M parameter must be a valid [checkpoint mode]:*** <dl> ** <dt>SQLITE_CHECKPOINT_PASSIVE<dd> ** ^Checkpoint as many frames as possible without waiting for any database ** readers or writers to finish, then sync the database file if all frames ** in the log were checkpointed. ^The [busy-handler callback] ** is never invoked in the SQLITE_CHECKPOINT_PASSIVE mode. ** ^On the other hand, passive mode might leave the checkpoint unfinished ** if there are concurrent readers or writers. *** <dt>SQLITE_CHECKPOINT_FULL<dd> ** ^This mode blocks (it invokes the ** [sqlite3_busy_handler|busy-handler callback]) until there is no ** database writer and all readers are reading from the most recent database ** snapshot. ^It then checkpoints all frames in the log file and syncs the ** database file. ^This mode blocks new database writers while it is pending, ** but new database readers are allowed to continue unimpeded. *** <dt>SQLITE_CHECKPOINT_RESTART<dd> ** ^This mode works the same way as SQLITE_CHECKPOINT_FULL with the addition ** that after checkpointing the log file it blocks (calls the ** [busy-handler callback]) ** until all readers are reading from the database file only. ^This ensures ** that the next writer will restart the log file from the beginning. ** ^Like SQLITE_CHECKPOINT_FULL, this mode blocks new ** database writer attempts while it is pending, but does not impede readers. *** <dt>SQLITE_CHECKPOINT_TRUNCATE<dd> ** ^This mode works the same way as SQLITE_CHECKPOINT_RESTART with the ** addition that it also truncates the log file to zero bytes just prior ** to a successful return. ** </dt> *** ^If pnLog is not NULL, then *pnLog is set to the total number of frames in ** the log file or to -1 if the checkpoint could not run because ** of an error or because the database is not in [WAL mode]. ^If pnCkpt is not ** NULL, then *pnCkpt is set to the total number of checkpointed frames in the ** log file (including any that were already checkpointed before the function ** was called) or to -1 if the checkpoint could not run due to an error or ** because the database is not in WAL mode. ^Note that upon successful ** completion of an SQLITE_CHECKPOINT_TRUNCATE, the log file will have been ** truncated to zero bytes and so both *pnLog and *pnCkpt will be set to zero. *** ^All calls obtain an exclusive "checkpoint" lock on the database file. ^If ** any other process is running a checkpoint operation at the same time, the ** lock cannot be obtained and SQLITE_BUSY is returned. ^Even if there is a ** busy-handler configured, it will not be invoked in this case. *** ^The SQLITE_CHECKPOINT_FULL, RESTART and TRUNCATE modes also obtain the ** exclusive "writer" lock on the database file. ^If the writer lock cannot be ** obtained immediately, and a busy-handler is configured, it is invoked and ** the writer lock retried until either the busy-handler returns 0 or the lock ** is successfully obtained. ^The busy-handler is also invoked while waiting for ** database readers as described above. ^If the busy-handler returns 0 before ** the writer lock is obtained or while waiting for database readers, the ** checkpoint operation proceeds from that point in the same way as ** SQLITE_CHECKPOINT_PASSIVE - checkpointing as many frames as possible ** without blocking any further. ^SQLITE_BUSY is returned in this case. *** ^If parameter zDb is NULL or points to a zero length string, then the ** specified operation is attempted on all WAL databases [attached] to ** [database connection] db. In this case the ** values written to output parameters *pnLog and *pnCkpt are undefined. ^If ** an SQLITE_BUSY error is encountered when processing one or more of the ** attached WAL databases, the operation is still attempted on any remaining ** attached databases and SQLITE_BUSY is returned at the end. ^If any other ** error occurs while processing an attached database, processing is abandoned ** and the error code is returned to the caller immediately. ^If no error ** (SQLITE_BUSY or otherwise) is encountered while processing the attached ** databases, SQLITE_OK is returned. *** ^If database zDb is the name of an attached database that is not in WAL ** mode, SQLITE_OK is returned and both *pnLog and *pnCkpt set to -1. ^If ** zDb is not NULL (or a zero length string) and is not the name of any ** attached database, SQLITE_ERROR is returned to the caller. *** ^Unless it returns SQLITE_MISUSE, ** the sqlite3_wal_checkpoint_v2() interface ** sets the error information that is queried by ** [sqlite3_errcode()] and [sqlite3_errmsg()]. *** ^The [PRAGMA wal_checkpoint] command can be used to invoke this interface ** from SQL.

80507. ** CAPI3REF: Virtual Table Constraint Operator Codes *** These macros defined the allowed values for the ** [sqlite3_index_info].aConstraint[].op field. Each value represents ** an operator that is part of a constraint term in the wHERE clause of ** a query that uses a [virtual table].

80508. Unknown opcode in sqlite3_file_control()

80509. sqlite3_pcache_methods2*

80510. ** CAPI3REF: Finding The Subtype Of SQL Values ** METHOD: sqlite3_value ** ** ^The sqlite3_value_subtype(V) function returns the subtype for ** an [application-defined SQL function] argument V. The subtype ** information can be used to pass a limited amount of context from ** one SQL function to another. Use the [sqlite3_result_subtype()] ** routine to set the subtype for the return value of an SQL function.

80511. xFunc, void*

80512. ** CAPI3REF: Enable Or Disable Shared Pager Cache ** ** ^This routine enables or disables the sharing of the database cache ** and schema data structures between [database connection | connections] ** to the same database. Sharing is enabled if the argument is true ** and disabled if the argument is false.)^** ** ^Cache sharing is enabled and disabled for an entire process. ** This is a change as of SQLite [version 3.5.0] ([dateof:3.5.0]). ** In prior versions of SQLite, ** sharing was enabled or disabled for each thread separately. ** ** ^The cache sharing mode set by this interface effects all subsequent ** calls to [sqlite3_open0], [sqlite3_open_v20], and [sqlite3_open160]. ** Existing database connections continue use the sharing mode ** that was in effect at the time they were opened.)^** ** ^This routine returns [SQLITE_OK] if shared cache was enabled or disabled ** successfully. An [error code] is returned otherwise.)^** ** ^Shared cache is disabled by default. But this might change in ** future releases of SQLite. Applications that care about shared ** cache setting should set it explicitly. ** ** Note: This method is disabled on MacOS X 10.7 and iOS version 5.0 ** and will always return SQLITE_MISUSE. On those systems, ** shared cache mode should be enabled per-database connection via ** [sqlite3_open_v20] with [SQLITE_OPEN_SHAREDCACHE]. ** ** This interface is threadsafe on processors where writing a ** 32-bit integer is atomic. ** ** See Also: [SQLite Shared-Cache Mode]

80513. **comment:** ** CAPI3REF: Finalize A Changeset Iterator ** ** This function is used to finalize an iterator allocated with ** [sqlite3changeset_start()]. ** ** This function should only be called on iterators created using the ** [sqlite3changeset_start()] function. If an application calls this ** function with an iterator passed to a conflict-handler by ** [sqlite3changeset_apply0], [SQLITE_MISUSE] is immediately returned and the ** call has no effect. ** ** If an error was encountered within a call to an sqlite3changeset_xxx() ** function (for example an [SQLITE_CORRUPT] in [sqlite3changeset_next0] or an ** [SQLITE_NOMEM] in [sqlite3changeset_new0]) then an error code corresponding ** to that error is returned by this function. Otherwise, SQLITE_OK is ** returned. This is to allow the following pattern (pseudo-code): ** ** sqlite3changeset_start(); ** while(SQLITE_ROW==sqlite3changeset_next0()) { ** // Do something with change. ** } ** rc = sqlite3changeset_finalize(); ** if(rc!=SQLITE_OK) { ** // An error has occurred ** }

label: code-design

80514. End of the 'extern "C"' block

80515. ** CAPI3REF: Virtual Table Interface Configuration ** ** This function may be called by either the [xConnect] or [xCreate] method ** of a [virtual table] implementation to configure ** various facets of the virtual table interface. ** ** If this interface is invoked outside the context of an xConnect or ** xCreate virtual table method then the behavior is undefined. ** ** At present, there is only one option that may be configured using ** this function. (See [SQLITE_VTAB_CONSTRAINT_SUPPORT].) Further options ** may be added in the future.

80516. end of the 'extern "C"' block

80517. ** CAPI3REF: Cancel Automatic Extension Loading ** ** ^The [sqlite3_cancel_auto_extension(X)] interface unregisters the ** initialization routine X that was registered using a prior call to ** [sqlite3_auto_extension(X)]. ^The [sqlite3_cancel_auto_extension(X)] ** routine returns 1 if initialization routine X was successfully ** unregistered and it returns 0 if X was not on the list of initialization ** routines.

80518. ***** End of sqlite3tree.h *****

80519. ** CAPI3REF: Initialize The SQLite Library ** ** ^The sqlite3_initialize() routine initializes the ** SQLite library. ^The sqlite3_shutdown() routine ** deallocates any resources that were allocated by sqlite3_initialize(). ** These routines are designed to aid in process initialization and ** shutdown on embedded systems. Workstation applications using ** SQLite normally do not need to invoke either of these routines. ** ** A call to sqlite3_initialize() is an "effective" call if it is ** the first time sqlite3_initialize() is invoked during the lifetime of ** the process, or if it is the first time sqlite3_initialize() is invoked ** following a call to sqlite3_shutdown(). ^Only an effective call ** of sqlite3_initialize() does any initialization. All other calls ** are harmless no-ops.)^** ** A call to sqlite3_shutdown() is an "effective" call if it is the first ** call to sqlite3_shutdown() since the last sqlite3_initialize(). ^Only ** an effective call to sqlite3_shutdown() does any deinitialization. ** All other valid calls to sqlite3_shutdown() are harmless no-ops.)^** ** ^The sqlite3_initialize() interface is threadsafe, but sqlite3_shutdown() ** is not. The sqlite3_shutdown() interface must only be called from a ** single thread. All open [database connections] must be closed and all ** other SQLite resources must be deallocated prior to invoking ** sqlite3_shutdown0). ** ** Among other things, ^sqlite3_initialize() will invoke ** sqlite3_os_init(). Similarly, ^sqlite3_shutdown() ** will invoke sqlite3_os_end0). ** ** ^The sqlite3_initialize() routine returns [SQLITE_OK] on success. ** ^If for some reason, sqlite3_initialize() is unable to initialize ** the library (perhaps it is unable to allocate a needed resource such ** as a mutex) it returns an [error code] other than [SQLITE_OK]. ** ** ^The sqlite3_initialize() routine is called internally by many other ** SQLite interfaces so that an application usually does not need to ** invoke sqlite3_initialize() directly. For example, [sqlite3_open0] ** calls sqlite3_initialize() so the SQLite library will be automatically ** initialized when [sqlite3_open0] is called if it has not been initialized ** already. ^However, if SQLite is compiled with the [SQLITE_OMIT_AUTOINIT] ** compile-time option, then the automatic calls to sqlite3_initialize() ** are omitted and the application must call sqlite3_initialize() directly ** prior to using any other SQLite interface. For maximum portability, ** it is recommended that applications always invoke sqlite3_initialize() ** directly prior to using any other SQLite interface. Future releases ** of SQLite may require this. In other words, the behavior exhibited ** when SQLite is compiled with [SQLITE_OMIT_AUTOINIT] might become the ** default behavior in some future release of SQLite. ** ** The sqlite3_os_init() routine does operating-system specific ** initialization of the SQLite library. The sqlite3_os_end0) ** routine undoes the effect of sqlite3_os_init(). Typical tasks ** performed by these routines include allocation or deallocation ** of static resources, initialization of global variables, ** setting up a default [sqlite3_vfs] module, or setting up ** a default configuration using [sqlite3_config0]. ** ** The application should never invoke either sqlite3_os_init() ** or sqlite3_os_end0 directly. The application should only invoke ** sqlite3_initialize() and sqlite3_shutdown0). The sqlite3_os_init() ** interface is called automatically by sqlite3_initialize() and ** sqlite3_os_end() is called by sqlite3_shutdown0). Appropriate ** implementations for sqlite3_os_init() and sqlite3_os_end() ** are built into SQLite when it is compiled for Unix, Windows, or OS/2. ** When [custom builds | built for other platforms] ** (using the [SQLITE_OS_OTHER=1] compile-time ** option) the application must supply a suitable implementation for ** sqlite3_os_init() and sqlite3_os_end0). An application-supplied ** implementation of sqlite3_os_init() or sqlite3_os_end0) ** must return [SQLITE_OK] on success and some other [error code] upon ** failure.

80520. Copy of third arg to preupdate_hook()

80521. Called by SQLite to clean up pUser

80522. ** CAPI3REF: Virtual Table Configuration Options ** ** These macros define the various options to the ** [sqlite3_vtab_config0] interface that [virtual table] implementations ** can use to customize and optimize their behavior. ** ** <dl> ** <dt>SQLITE_VTAB_CONSTRAINT_SUPPORT ** <dd>Calls of the form ** [sqlite3_vtab_config](db,SQLITE_VTAB_CONSTRAINT_SUPPORT,X) are supported, ** where X is an integer. If X is zero, then the [virtual table] whose ** [xCreate] or [xConnect] method invoked [sqlite3_vtab_config0] does not ** support constraints. In this configuration (which is the default) if ** a call to the [xUpdate] method returns [SQLITE_CONSTRAINT], then the entire ** statement is rolled back as if [ON CONFLICT | OR ABORT] had been ** specified as part of the users SQL statement, regardless of the actual ** ON CONFLICT mode specified. ** ** If X is non-zero, then the virtual table implementation guarantees ** that if [xUpdate] returns [SQLITE_CONSTRAINT], it will do so before ** any modifications to internal or persistent data structures have been made. ** If the [ON CONFLICT] mode is ABORT, FAIL, IGNORE or ROLLBACK, SQLite ** is able to roll back a statement or database transaction, and abandon ** or continue processing the current SQL statement as appropriate. ** If the ON CONFLICT mode is REPLACE and the [xUpdate] method returns ** [SQLITE_CONSTRAINT], SQLite handles this as if the ON CONFLICT mode ** had been ABORT. ** ** Virtual table implementations that are required to handle OR REPLACE ** must do so within the [xUpdate] method. If a call to the ** [sqlite3_vtab_on_conflict0] function indicates that the current ON ** CONFLICT policy is REPLACE, the virtual table implementation should ** silently replace the appropriate rows within the xUpdate callback and ** return SQLITE_OK. Or, if this is not possible, it may return ** SQLITE_CONSTRAINT, in which case SQLite falls back to OR ABORT ** constraint handling. ** </dl>

80523. **comment:** ** CAPI3REF: Low-Level Control Of Database Files ** METHOD: sqlite3 *** ** ^The [sqlite3_file_control0] interface makes a direct call to the ** xFileControl method for the [sqlite3_io_methods] object associated ** with a particular database identified by the second argument. ^The ** name of the database is "main" for the main database or "temp" for the ** TEMP database, or the name that appears after the AS keyword for ** databases that are added using the [ATTACH] SQL command. ** ^A NULL pointer can be used in place of "main" to refer to the ** main database file. ** ^The third and fourth parameters to this routine ** are passed directly through to the second and third parameters of ** the xFileControl method. ^The return value of the xFileControl ** method becomes the return value of this routine. ** ** ^The SQLITE_FCNTL_FILE_POINTER value for the op parameter causes ** a pointer to the underlying [sqlite3_file] object to be written into ** the space pointed to by the 4th parameter. ^The SQLITE_FCNTL_FILE_POINTER ** case is a short-circuit path which does not actually invoke the ** underlying sqlite3_io_methods.xFileControl method. ** ** ^If the second parameter (zDbName) does not match the name of any ** open database file, then SQLITE_ERROR is returned. ^This error ** code is not remembered and will not be recalled by [sqlite3_errcode0] ** or [sqlite3_errmsg0]. The underlying xFileControl method might ** also return SQLITE_ERROR. There is no way to distinguish between ** an incorrect zDbName and an SQLITE_ERROR return from the underlying ** xFileControl method. ** ** See also: [SQLITE_FCNTL_LOCKSTATE]

label: code-design

80524. **comment:** **** * CUSTOM TOKENIZERS *** Applications may also register custom tokenizer types. A tokenizer ** is registered by providing fts5 with a populated instance of the ** following structure. All structure methods must be defined, setting ** any member of the fts5_tokenizer struct to NULL leads to undefined ** behaviour. The structure methods are expected to function as follows: ** ** xCreate: ** This function is used to allocate and initialize a tokenizer instance. ** A tokenizer instance is required to actually tokenize text. *** The first argument passed to this function is a copy of the (void*) ** pointer provided by the application when the fts5_tokenizer object ** was registered with FTS5 (the third argument to xCreateTokenizer()). ** The second and third arguments are an array of nul-terminated strings ** containing the tokenizer arguments, if any, specified following the ** tokenizer name as part of the CREATE VIRTUAL TABLE statement used ** to create the FTS5 table. *** The final argument is an output variable. If successful, (*ppOut) ** should be set to point to the new tokenizer handle and SQLITE_OK ** returned. If an error occurs, some value other than SQLITE_OK should ** be returned. In this case, fts5 assumes that the final value of *ppOut ** is undefined. *** ** xDelete: ** This function is invoked to delete a tokenizer handle previously ** allocated using xCreate(). Fts5 guarantees that this function will ** be invoked exactly once for each successful call to xCreate(). *** ** xTokenize: ** This function is expected to tokenize the nText byte string indicated ** by argument pText. pText may or may not be null-terminated. The first ** argument passed to this function is a pointer to an Fts5Tokenizer object ** returned by an earlier call to xCreate(). *** ** The second argument indicates the reason that FTS5 is requesting ** tokenization of the supplied text. This is always one of the following ** four values: ** ** <i> FTS5_TOKENIZE_DOCUMENT - A document is being inserted into ** or removed from the FTS table. The tokenizer is being invoked to ** determine the set of tokens to add to (or delete from) the ** FTS index. ** ** <i> FTS5_TOKENIZE_QUERY - A MATCH query is being executed ** against the FTS index. The tokenizer is being called to tokenize ** a bareword or quoted string specified as part of the query. ** ** <i> (FTS5_TOKENIZE_QUERY | FTS5_TOKENIZE_PREFIX) - Same as ** FTS5_TOKENIZE_QUERY, except that the bareword or quoted string is ** followed by a "" character, indicating that the last token ** returned by the tokenizer will be treated as a token prefix. ** ** <i> FTS5_TOKENIZE_AUX - The tokenizer is being invoked to ** satisfy an fts5_api.xTokenize() request made by an auxiliary ** function. Or an fts5_api.xColumnSize() request made by the same ** on a columnsize=0 database. ** *** ** For each token in the input string, the supplied callback xToken() must ** be invoked. The first argument to it should be a copy of the pointer ** passed as the second argument to xTokenize(). The third and fourth ** arguments are a pointer to a buffer containing the token text, and the ** size of the token in bytes. The 4th and 5th arguments are the byte offsets ** of the first byte of and first byte immediately following the text from ** which the token is derived within the input. *** ** The second argument passed to the xToken() callback ("tflags") should ** normally be set to 0. The exception is if the tokenizer supports ** synonyms. In this case see the discussion below for details. *** ** FTS5 assumes the xToken() callback is invoked for each token in the ** order that they occur within the input text. *** ** If an xToken() callback returns any value other than SQLITE_OK, then ** the tokenization should be abandoned and the xTokenize() method should ** immediately return a copy of the xToken() return value. Or, if the ** input buffer is exhausted, xTokenize() should return SQLITE_OK. Finally, ** if an error occurs with the xTokenize() implementation itself, it ** may abandon the tokenization and return any error code other than ** SQLITE_OK or SQLITE_DONE. *** ** SYNONYM SUPPORT *** ** Custom tokenizers may also support synonyms. Consider a case in which a ** user wishes to query for a phrase such as "first place". Using the ** built-in tokenizers, the FTS5 query 'first + place' will match instances ** of "first place" within the document set, but not alternative forms ** such as "1st place". In some applications, it would be better to match ** all instances of "first place" or "1st place" regardless of which form ** the user specified in the MATCH query text. *** ** There are several ways to approach this in FTS5: ** ** By mapping all synonyms to a single token. In this case, the ** In the above example, this means that the tokenizer returns the ** same token for inputs "first" and "1st". Say that token is in ** fact "first", so that when the user inserts the document "I won ** 1st place" entries are added to the index for tokens "1", "won", ** "first" and "place". If the user then queries for '1st + place', ** the tokenizer substitutes "first" for "1st" and the query works ** as expected. ** ** By adding multiple synonyms for a single term to the FTS index. ** In this case, when tokenizing query text, the tokenizer may ** provide multiple synonyms for a single term within the document. ** FTS5 then queries the index for each synonym individually. For ** example, faced with the query: ** ** <codeblock> ** ... MATCH 'first place'</codeblock> ** ** the tokenizer offers both "1st" and "first" as synonyms for the ** first token in the MATCH query and FTS5 effectively runs a query ** similar to: ** ** <codeblock> ** ... MATCH '(first OR 1st) place'</codeblock> ** ** except that, for the purposes of auxiliary functions, the query ** still appears to contain just two phrases - "(first OR 1st)" ** being treated as a single phrase. ** ** By adding multiple synonyms for a single term to the FTS index. ** Using this method, when tokenizing document text, the tokenizer ** provides multiple synonyms for each token. So that when a ** document such as "I won first place" is tokenized, entries are ** added to the FTS index for "i", "won", "first", "1st" and ** "place". ** ** This way, even if the tokenizer does not provide synonyms ** when tokenizing query text (it should not - to do would be ** inefficient), it doesn't matter if the user queries for ** 'first + place' or '1st + place', as there are entries in the ** FTS index corresponding to both forms of the first token. ** *** ** Whether it is parsing document or query text, any call to xToken that ** specifies a <i>tflags</i> argument with the FTS5_TOKEN_COLOCATED bit ** is considered to supply a synonym for the previous token. For example, ** when parsing the document "I won first place", a tokenizer that supports ** synonyms would call xToken() 5 times, as follows: ** ** <codeblock> ** xToken(pCtx, 0, "i", 1, 0, 1); ** xToken(pCtx, 0, "won", 3, 2, 5); ** xToken(pCtx, 0, "first", 5, 6, 11); ** xToken(pCtx, FTS5_TOKEN_COLOCATED, "1st", 3, 6, 11); ** xToken(pCtx, 0, "place", 5, 12, 17); ** </codeblock> ** ** It is an error to specify the FTS5_TOKEN_COLOCATED flag the first time ** xToken() is called. Multiple synonyms may be specified for a single token ** by making multiple calls to xToken(FTS5_TOKEN_COLOCATED) in sequence. ** There is no limit to the number of synonyms that may be provided for a ** single token. ** ** In many cases, method (1) above is the best approach. It does not add ** extra data to the FTS index or require FTS5 to query for multiple terms, ** so it is efficient in terms of disk space and query speed. However, it ** does not support prefix queries very well. If, as suggested above, the ** token "first" is substituted for "1st" by the tokenizer, then the query: ** ** <codeblock> ** ... MATCH '1s'*</codeblock> ** ** will not match documents that contain the token "1st" (as the tokenizer ** will probably not map "1s" to any prefix of "first"). ** ** For full prefix support, method (3) may be preferred. In this case, ** because the index contains entries for both "first" and "1st", prefix ** queries such as 'fi*' or '1s*' will match correctly. However, because ** extra entries are added to the FTS index, this method uses more space ** within the database. ** ** Method (2) offers a midpoint between (1) and (3). Using this method, ** a query such as '1s*' will match documents that contain the literal ** token "1st", but not "first" (assuming the tokenizer is not able to ** provide synonyms for prefixes). However, a non-prefix query like '1st' ** will match against "1st" and "first". This method does not require ** extra disk space, as no extra entries are added to the FTS index. ** On the other hand, it may require more CPU cycles to run MATCH queries, ** as separate queries of the FTS index are required for each synonym. ** ** When using methods (2) or (3), it is important that the tokenizer only ** provide synonyms when tokenizing document text (method (2)) or query ** text (method (3)), not both. Doing so will not cause any errors, but is ** inefficient.

label: code-design

80525. Information desired. SQLITE_SCANSTAT_*

80526. int nByte

80527. OUTPUT: True if column is auto-increment

80528. Ok for sqlite3_open_v2()

80529. OUT: Total number of frames checkpointed

80530. sqlite3_int64, sqlite3_int64

80531. Flags

80532. IMP: R-37514-35566

80533. Name of the database

80534. boolean

80535. Number used to identify the index

80536. Input changeset

80537. ***** Begin file sqlite3rtree.h *****

80538. Byte offset of token within input text

80539. Used internally - xBestIndex should ignore

80540. Data type mismatch

80541. ** CAPI3REF: Delete A Changegroup Object

80542. ** CAPI3REF: Virtual File System Objects *** A virtual filesystem (VFS) is an [sqlite3_vfs] object ** that SQLite uses to interact ** with the underlying operating system. Most SQLite builds come with a ** single default VFS that is appropriate for the host computer. ** New VFSes can be registered and existing VFSes can be unregistered. ** The following interfaces are provided. ** ** ^The sqlite3_vfs_find() interface returns a pointer to a VFS given its name. ** ^Names are case sensitive. ** ^Names are zero-terminated UTF-8 strings. ** ^If there is no match, a NULL pointer is returned. ** ^If zVfsName is NULL then the default VFS is returned. ** ** ^New VFSes are registered with sqlite3_vfs_register(). ** ^Each new VFS becomes the default VFS if the makeDflt flag is set. ** ^The same VFS can be registered multiple times without injury. ** ^To make an existing VFS into the default VFS, register it again ** with the makeDflt flag set. If two different VFSes with the ** same name are registered, the behavior is undefined. If a ** VFS is registered with a name that is NULL or an empty string, ** then the behavior is undefined. ** ** ^Unregister a VFS with the sqlite3_vfs_unregister() interface. ** ^If the default VFS is unregistered, another VFS is chosen as ** the default. The choice for the new VFS is arbitrary.)^

80543. **comment:** ** CAPI3REF: Invert A Changeset *** This function is used to "invert" a changeset object. Applying an inverted ** changeset to a database reverses the effects of applying the uninverted ** changeset. Specifically: ** ** ** Each DELETE change is changed to an INSERT, and ** Each INSERT

change is changed to a DELETE, and ** For each UPDATE change, the old.* and new.* values are exchanged. ** *** This function does not change the order in which changes appear within ** the changeset. It merely reverses the sense of each individual change. *** If successful, a pointer to a buffer containing the inverted changeset ** is stored in *ppOut, the size of the same buffer is stored in *pnOut, and ** SQLITE_OK is returned. If an error occurs, both *pnOut and *ppOut are ** zeroed and an SQLite error code returned. *** It is the responsibility of the caller to eventually call sqlite3_free() ** on the *ppOut pointer to free the buffer allocation following a successful ** call to this function. *** WARNING/TODO: This function currently assumes that the input is a valid ** changeset. If it is not, the results are undefined.

label: code-design

80544. ***** ** FTS5 EXTENSION REGISTRATION API
80545. **comment:** ** CAPI3REF: Result Values From A Query ** KEYWORDS: {column access functions} ** METHOD: sqlite3_stmt ** ** Summary: ** <blockquote><table border=0 cellpadding=0 cellspacing=0> ** <tr><td>sqlite3_column_blob</td>→<td>BLOB result ** <tr><td> sqlite3_column_double</td>→<td>REAL result ** <tr><td>sqlite3_column_int</td>→<td>32-bit INTEGER result ** <tr><td> sqlite3_column_int64</td>→<td>64-bit INTEGER result ** <tr><td>sqlite3_column_text</td>→<td>UTF-8 TEXT result ** <tr><td> sqlite3_column_text16</td>→<td>UTF-16 TEXT result ** <tr><td>sqlite3_column_value</td>→<td>The result as an ** [sqlite3_value|unprotected sqlite3_value] object. ** <tr><td> <td> <td> ** <tr><td>sqlite3_column_bytes</td>→<td>Size of a BLOB ** or a UTF-8 TEXT result in bytes ** <tr><td>sqlite3_column_bytes16</td> <td> ** <td>→<td> <td>Size of UTF-16 ** TEXT in bytes ** <tr><td>sqlite3_column_type</td>→<td>Default ** datatype of the result ** </table></blockquote> *** Details: *** ^These routines return information about a single column of the current ** result row of a query. ^In every case the first argument is a pointer ** to the [prepared statement] that is being evaluated (the [sqlite3_stmt*] ** that was returned from [sqlite3_prepare_v2()] or one of its variants) ** and the second argument is the index of the column for which information ** should be returned. ^The leftmost column of the result set has the index 0. *** ^The number of columns in the result can be determined using ** [sqlite3_column_count()]. *** If the SQL statement does not currently point to a valid row, or if the ** column index is out of range, the result is undefined. ** These routines may only be called when the most recent call to ** [sqlite3_step()] has returned [SQLITE_ROW] and neither ** [sqlite3_reset()] nor [sqlite3_finalize()] have been called subsequently. ** If any of these routines are called after [sqlite3_reset()] or ** [sqlite3_finalize()] or after [sqlite3_step()] has returned ** something other than [SQLITE_ROW], the results are undefined. ** If [sqlite3_step()] or [sqlite3_reset()] or [sqlite3_finalize()] ** are called from a different thread while any of these routines ** are pending, then the results are undefined. *** The first six interfaces (_blob, _double, _int, _int64, _text, and _text16) ** each return the value of a result column in a specific data format. If ** the result column is not initially in the requested format (for example, ** if the query returns an integer but the sqlite3_column_text() interface ** is used to extract the value) then an automatic type conversion is performed. *** ^The sqlite3_column_type() routine returns the ** [SQLITE_INTEGER | datatype code] for the initial data type ** of the result column. ^The returned value is one of [SQLITE_INTEGER], ** [SQLITE_FLOAT], [SQLITE_TEXT], [SQLITE_BLOB], or [SQLITE_NULL]. ** The return value of sqlite3_column_type() can be used to decide which ** of the first six interface should be used to extract the column value. ** The value returned by sqlite3_column_type() is only meaningful if no ** automatic type conversions have occurred for the value in question. ** After a type conversion, the result of calling sqlite3_column_type() ** is undefined, though harmless. Future ** versions of SQLite may change the behavior of sqlite3_column_type() ** following a type conversion. *** If the result is a BLOB or a TEXT string, then the sqlite3_column_bytes() ** or sqlite3_column_bytes16() interfaces can be used to determine the size ** of that BLOB or string. *** ^If the result is a BLOB or UTF-8 string then the sqlite3_column_bytes() ** routine returns the number of bytes in that BLOB or string. ** ^If the result is a UTF-16 string, then sqlite3_column_bytes() converts ** the string to UTF-8 and then returns the number of bytes. ** ^If the result is a numeric value then sqlite3_column_bytes() uses ** [sqlite3_snprintf()] to convert that value to a UTF-8 string and returns ** the number of bytes in that string. ** ^If the result is NULL, then sqlite3_column_bytes() returns zero. *** ^If the result is a BLOB or UTF-16 string then the sqlite3_column_bytes16() ** routine returns the number of bytes in that BLOB or string. ** ^If the result is a UTF-8 string, then sqlite3_column_bytes16() converts ** the string to UTF-16 and then returns the number of bytes. ** ^If the result is a numeric value then sqlite3_column_bytes16() uses ** [sqlite3_snprintf()] to convert that value to a UTF-16 string and returns ** the number of bytes in that string. ** ^If the result is NULL, then sqlite3_column_bytes16() returns zero. *** ^The values returned by [sqlite3_column_bytes()] and ** [sqlite3_column_bytes16()] do not include the zero terminators at the end ** of the string. ^For clarity: the values returned by ** [sqlite3_column_bytes()] and [sqlite3_column_bytes16()] are the number of ** bytes in the string, not the number of characters. *** ^Strings returned by sqlite3_column_text() and sqlite3_column_text16(), ** even empty strings, are always zero-terminated. ^The return ** value from sqlite3_column_blob() for a zero-length BLOB is a NULL pointer. *** Warning: ^The object returned by [sqlite3_column_value()] is an ** [unprotected sqlite3_value] object. In a multithreaded environment, ** an unprotected sqlite3_value object may only be used safely with ** [sqlite3_bind_value()] and [sqlite3_result_value()]. ** If the [unprotected sqlite3_value] object returned by ** [sqlite3_column_value()] is used in any other way, including calls ** to routines like [sqlite3_value_int()], [sqlite3_value_text()], ** or [sqlite3_value_bytes()], the behavior is not threadsafe. ** Hence, the sqlite3_column_value() interface ** is normally only useful within the implementation of ** [application-defined SQL functions] or [virtual tables], not within ** top-level application code. *** ^The these routines may attempt to convert the datatype of the result. ** ^For example, if the internal representation is FLOAT and a text result ** is requested, [sqlite3_snprintf()] is used internally to perform the ** conversion automatically. ^The following table details the conversions ** that are applied: *** <blockquote> ** <table border="1"> ** <tr><th> Internal
Type <th> Requested
Type <th> Conversion ** <tr><td> NULL <td> INTEGER <td> Result is 0 ** <tr><td> NULL <td> FLOAT <td> Result is 0.0 ** <tr><td> NULL <td> TEXT <td> Result is a NULL pointer ** <tr><td> NULL <td> BLOB <td> Result is a NULL pointer ** <tr><td> INTEGER <td> FLOAT <td> Convert from integer to float ** <tr><td> INTEGER <td> TEXT <td> ASCII rendering of the integer ** <tr><td> INTEGER <td> BLOB <td> Same as INTEGER->TEXT ** <tr><td> FLOAT <td> INTEGER <td> [CAST] to INTEGER ** <tr><td> FLOAT <td> TEXT <td> ASCII rendering of the float ** <tr><td> FLOAT <td> BLOB <td> [CAST] to BLOB ** <tr><td> TEXT <td> INTEGER <td> [CAST] to INTEGER ** <tr><td> TEXT <td> FLOAT <td> [CAST] to REAL ** <tr><td> TEXT <td> BLOB <td> No change ** <tr><td> BLOB <td> INTEGER <td> [CAST] to INTEGER ** <tr><td> BLOB <td> FLOAT <td> [CAST] to REAL ** <tr><td> BLOB <td> TEXT <td> Add a zero terminator if needed ** </table> ** </blockquote>)^ *** Note that when type conversions occur, pointers returned by prior ** calls to sqlite3_column_blob(), sqlite3_column_text(), and/or ** sqlite3_column_text16() may be invalidated. ** Type conversions and pointer invalidations might occur ** in the following cases: *** ** The initial content is a BLOB and sqlite3_column_text() or ** sqlite3_column_text16() is called. A zero-terminator might ** need to be added to the string. ** The initial content is UTF-8 text and sqlite3_column_bytes16() or ** sqlite3_column_text16() is called. The content must be converted ** to UTF-16. ** The initial content is UTF-16 text and sqlite3_column_bytes() or ** sqlite3_column_text() is called. The content must be converted ** to UTF-8. ** *** ^Conversions between UTF-16be and UTF-16le are always done in place and do ** not invalidate a prior pointer, though of course the content of the buffer ** that the prior pointer references will have been modified. Other kinds ** of conversion are done in place when it is possible, but sometimes they ** are not possible and in those cases prior pointers are invalidated. *** ^The safest policy is to invoke these routines ** in one of the following ways: *** ** sqlite3_column_text() followed by sqlite3_column_bytes() ** sqlite3_column_blob() followed by sqlite3_column_bytes() ** sqlite3_column_text16() followed by sqlite3_column_bytes16() ** *** ^In other words, you should call sqlite3_column_text(), ** sqlite3_column_blob(), or sqlite3_column_text16() first to force the result ** into the desired format, then invoke sqlite3_column_bytes() or ** sqlite3_column_bytes16() to find the size of the result. Do not mix calls ** to sqlite3_column_text() or sqlite3_column_blob() with calls to ** sqlite3_column_bytes16(), and do not mix calls to sqlite3_column_text16() ** with calls to sqlite3_column_bytes(). ** ^The pointers returned are valid until a type conversion occurs as ** described above, or until [sqlite3_step()] or [sqlite3_reset()] or ** [sqlite3_finalize()] is called. ^The memory space used to hold strings ** and BLOBS is freed automatically. Do not pass the pointers returned ** from [sqlite3_column_blob()], [sqlite3_column_text()], etc. into ** [sqlite3_free()]. *** ^If a memory allocation error occurs during the evaluation of any ** of these routines, a default value is returned. The default value ** is either the integer 0, the floating point number 0.0, or a NULL ** pointer. Subsequent calls to [sqlite3_errcode()] will return ** [SQLITE_NOMEM].)^
label: code-design

80546. ** CAPI3REF: Determine If An SQL Statement Writes The Database ** METHOD: sqlite3_stmt ** ** ^The sqlite3_stmt_READONLY(X) interface returns true (non-zero) if ** and only if the [prepared statement] X makes no direct changes to ** the content of the database file. *** ^Note that [application-defined SQL functions] or ** [virtual tables] might change the database indirectly as a side effect. ** ^For example, if an application defines a function "eval()" that ** calls [sqlite3_exec()], then the following SQL statement would ** change the database file through side-effects: *** <blockquote><pre> ** SELECT eval('DELETE FROM t1') FROM t2; ** </pre></blockquote> ** ^But because the [SELECT] statement does not change the database file ** directly, sqlite3_stmt_READONLY() would still return true.)^ *** ^Transaction control statements such as [BEGIN], [COMMIT], [ROLLBACK], ** [SAVEPOINT], and [RELEASE] cause sqlite3_stmt_READONLY() to return true, ** since the statements themselves do not actually modify the database but ** rather they control the timing of when other statements modify the ** database. ^The [ATTACH] and [DETACH] statements also cause ** sqlite3_stmt_READONLY() to return true since, while those statements ** change the configuration of a database connection, they do not make ** changes to the content of the database files on disk. ** ^The sqlite3_stmt_READONLY() interface returns true for [BEGIN] since ** [BEGIN] merely sets internal flags, but the [BEGIN|BEGIN IMMEDIATE] and ** [BEGIN|BEGIN EXCLUSIVE] commands do touch the database and so ** sqlite3_stmt_READONLY() returns false for those commands.

80547. Return the size of an allocation

80548. sqlite3_mem_methods*

80549. if >0, constraint is part of argv to xFilter

80550. Database filename (UTF-16)
80551. Attempt to write a readonly database
80552. Input function
80553. ** CAPI3REF: Error Logging Interface *** ** ^The [sqlite3_log()] interface writes a message into the [error log] ** established by the [SQLITE_CONFIG_LOG] option to [sqlite3_config()]. ** ^If logging is enabled, the zFormat string and subsequent arguments are ** used with [sqlite3_snprintf()] to generate the final output string. *** ** The sqlite3_log() interface is intended for use by extensions such as ** virtual tables, collating functions, and SQL functions. While there is ** nothing to prevent an application from calling sqlite3_log(), doing so ** is considered bad form. *** ** The zFormat string must not be NULL. *** ** To avoid deadlocks and other threading problems, the sqlite3_log() routine ** will not use dynamically allocated memory. The log message is stored in ** a fixed-length buffer on the stack. If the log message is longer than ** a few hundred characters, it will be truncated to the length of the ** buffer.
80554. ** CAPI3REF: Copy And Free SQL Values *** METHOD: sqlite3_value *** ** ^The sqlite3_value_dup(V) interface makes a copy of the [sqlite3_value] ** object D and returns a pointer to that copy. ^The [sqlite3_value] returned ** is a [protected sqlite3_value] object even if the input is not. ** ^The sqlite3_value_dup(V) interface returns NULL if V is NULL or if a ** memory allocation fails. *** ** ^The sqlite3_value_free(V) interface frees an [sqlite3_value] object ** previously obtained from [sqlite3_value_dup()]. ^If V is a NULL pointer ** then sqlite3_value_free(V) is a harmless no-op.
80555. ** CAPI3REF: Automatically Load Statically Linked Extensions *** ** ^This interface causes the xEntryPoint() function to be invoked for ** each new [database connection] that is created. The idea here is that ** xEntryPoint() is the entry point for a statically linked [SQLite extension] ** that is to be automatically loaded into all new database connections. *** ** ^Even though the function prototype shows that xEntryPoint() takes ** no arguments and returns void, SQLite invokes xEntryPoint() with three ** arguments and expects an integer result as if the signature of the ** entry point where as follows: ** ** <blockquote><pre> ** &nbs; int xEntryPoint(** &nbs; sqlite3 *db, ** &nbs; const char **pzErrMsg, ** &nbs; const struct sqlite3_api_routines *pThunk ** &nbs;); ** </pre> </blockquote>** ** ^If the xEntryPoint routine encounters an error, it should make *pzErrMsg ** point to an appropriate error message (obtained from [sqlite3_mprintf()]) ** and return an appropriate [error code]. ^SQLite ensures that *pzErrMsg ** is NULL before calling the xEntryPoint(). ^SQLite will invoke ** [sqlite3_free()] on *pzErrMsg after xEntryPoint() returns. ^If any ** xEntryPoint() returns an error, the [sqlite3_open()], [sqlite3_open16()], ** or [sqlite3_open_v2()] call that provoked the xEntryPoint() will fail. *** ** ^Calling sqlite3_auto_extension(X) with an entry point X that is already ** on the list of automatic extensions is a harmless no-op. ^No entry point ** will be called more than once for each database connection that is opened. *** ** See also: [sqlite3_reset_auto_extension()] ** and [sqlite3_cancel_auto_extension()]
80556. Use native byte order
80557. !defined(_SQLITESESSION_H_) && defined(SQLITE_ENABLE_SESSION)
80558. **comment:** ** CAPI3REF: Name Of The Folder Holding Temporary Files *** ** ^If this global variable is made to point to a string which is ** the name of a folder (a.k.a. directory), then all temporary files ** created by SQLite when using a built-in [sqlite3_vfs | VFS] ** will be placed in that directory.)^ If this variable ** is a NULL pointer, then SQLite performs a search for an appropriate ** temporary file directory. *** ** Applications are strongly discouraged from using this global variable. ** It is required to set a temporary folder on Windows Runtime (WinRT). ** But for all other platforms, it is highly recommended that applications ** neither read nor write this variable. This global variable is a relic ** that exists for backwards compatibility of legacy applications and should ** be avoided in new projects. *** ** It is not safe to read or modify this variable in more than one ** thread at a time. It is not safe to read or modify this variable ** if a [database connection] is being used at the same time in a separate ** thread. ** It is intended that this variable be set once ** as part of process initialization and before any SQLite interface ** routines have been called and that this variable remain unchanged ** thereafter. *** ** ^The [temp_store_directory pragma] may modify this variable and cause ** it to point to memory obtained from [sqlite3_malloc]. ^Furthermore, ** the [temp_store_directory pragma] always assumes that any string ** that this variable points to is held in memory obtained from ** [sqlite3_malloc] and the pragma may attempt to free that memory ** using [sqlite3_free]. ** Hence, if this variable is modified directly, either it should be ** made NULL or made to point to memory obtained from [sqlite3_malloc] ** or else the use of the [temp_store_directory pragma] should be avoided. ** Except when requested by the [temp_store_directory pragma], SQLite ** does not free the memory that sqlite3_temp_directory points to. If ** the application wants that memory to be freed, it must do ** so itself, taking care to only do so after all [database connection] ** objects have been destroyed. *** ** Note to Windows Runtime users: The temporary directory must be set ** prior to calling [sqlite3_open] or [sqlite3_open_v2]. Otherwise, various ** features that require the use of temporary files may fail. Here is an ** example of how to do this using C++ with the Windows Runtime: *** ** <blockquote><pre> ** LPCWSTR zPath = Windows::Storage::ApplicationData::Current-> ** &nbs; TemporaryFolder->Path->Data(); ** char zPathBuf#91;MAX_PATH + 1#93;; ** memset(zPathBuf, 0, sizeof(zPathBuf)); ** WideCharToMultiByte(CP_UTF8, 0, zPath, -1, zPathBuf, sizeof(zPathBuf), ** &nbs; NULL, NULL); ** sqlite3_temp_directory = sqlite3_mprintf("%s", zPathBuf); ** </pre></blockquote>
label: code-design
80559. **comment:** ** CAPI3REF: Declare The Schema Of A Virtual Table *** ** ^The [xCreate] and [xConnect] methods of a ** [virtual table module] call this interface ** to declare the format (the names and datatypes of the columns) of ** the virtual tables they implement.
label: code-design
80560. Callback implementation user data
80561. Number of coordinates
80562. Database name
80563. ** CAPI3REF: The pre-update hook. *** ** ^These interfaces are only available if SQLite is compiled using the ** [SQLITE_ENABLE_PREUPDATE_HOOK] compile-time option. *** ** ^The [sqlite3_preupdate_hook()] interface registers a callback function ** that is invoked prior to each [INSERT], [UPDATE], and [DELETE] operation ** on a database table. ** ^At most one preupdate hook may be registered at a time on a single ** [database connection]; each call to [sqlite3_preupdate_hook()] overrides ** the previous setting. ** ^The preupdate hook is disabled by invoking [sqlite3_preupdate_hook()] ** with a NULL pointer as the second parameter. ** ^The third parameter to [sqlite3_preupdate_hook()] is passed through as ** the first parameter to callbacks. *** ** ^The preupdate hook only fires for changes to real database tables; the ** preupdate hook is not invoked for changes to [virtual tables] or to ** system tables like sqlite_master or sqlite_stat1. *** ** ^The second parameter to the preupdate callback is a pointer to ** the [database connection] that registered the preupdate hook. ** ^The third parameter to the preupdate callback is one of the constants ** [SQLITE_INSERT], [SQLITE_DELETE], or [SQLITE_UPDATE] to identify the ** kind of update operation that is about to occur. ** ^The fourth parameter to the preupdate callback is the name of the ** database within the database connection that is being modified. This ** will be "main" for the main database or "temp" for TEMP tables or ** the name given after the AS keyword in the [ATTACH] statement for attached ** databases.)^ ** ^The fifth parameter to the preupdate callback is the name of the ** table that is being modified. *** ** For an UPDATE or DELETE operation on a [rowid table], the sixth ** parameter passed to the preupdate callback is the initial [rowid] of the ** row being modified or deleted. For an INSERT operation on a rowid table, ** or any operation on a WITHOUT ROWID table, the value of the sixth ** parameter is undefined. For an INSERT or UPDATE on a rowid table the ** seventh parameter is the final rowid value of the row being inserted ** or updated. The value of the seventh parameter passed to the callback ** function is not defined for operations on WITHOUT ROWID tables, or for ** INSERT operations on rowid tables. *** ** The [sqlite3_preupdate_old()], [sqlite3_preupdate_new()], ** [sqlite3_preupdate_count()], and [sqlite3_preupdate_depth()] interfaces ** provide additional information about a preupdate event. These routines ** may only be called from within a preupdate callback. Invoking any of ** these routines from outside of a preupdate callback or with a ** [database connection] pointer that is different from the one supplied ** to the preupdate callback results in undefined and probably undesirable ** behavior. *** ** ^The [sqlite3_preupdate_count(D)] interface returns the number of columns ** in the row that is being inserted, updated, or deleted. *** ** ^The [sqlite3_preupdate_old(D,N,P)] interface writes into P a pointer to ** a [protected sqlite3_value] that contains the value of the Nth column of ** the table row before it is updated. The N parameter must be between 0 ** and one less than the number of columns or the behavior will be ** undefined. This must only be used within SQLITE_UPDATE and SQLITE_DELETE ** preupdate callbacks; if it is used by an SQLITE_INSERT callback then the ** behavior is undefined. The [sqlite3_value] that P points to ** will be destroyed when the preupdate callback returns. *** ** ^The [sqlite3_preupdate_new(D,N,P)] interface writes into P a pointer to ** a [protected sqlite3_value] that contains the value of the Nth column of ** the table row after it is updated. The N parameter must be between 0 ** and one less than the number of columns or the behavior will be ** undefined. This must only be used within SQLITE_INSERT and SQLITE_UPDATE ** preupdate callbacks; if it is used by an SQLITE_DELETE callback then the ** behavior is undefined. The [sqlite3_value] that P points to ** will be destroyed when the preupdate callback returns. *** ** ^The [sqlite3_preupdate_depth(D)] interface returns 0 if the preupdate ** callback was invoked as a result of a direct insert, update, or delete ** operation; or 1 for inserts, updates, or deletes invoked by top-level ** triggers; or 2 for changes resulting from triggers called by top-level ** triggers; and so forth. *** ** See also: [sqlite3_update_hook()]
80564. Load the extension into this database connection
80565. Estimated cost of using this index
80566. **comment:** ** CAPI3REF: Add A Changeset To A Changegroup *** ** Add all changes within the changeset (or patchset) in buffer pData (size ** nData bytes) to the changegroup. *** ** If the buffer contains a patchset, then all prior calls to this function ** on the same changegroup object must also have specified patchsets. Or, if ** the buffer contains a changeset, so must have the earlier calls to this ** function. Otherwise, SQLITE_ERROR is returned and no changes are added ** to the changegroup. *** ** Rows within the changeset and changegroup are identified by the values in ** their PRIMARY KEY columns. A change in the changeset is considered to ** apply to the same row as a change already present in the changegroup if ** the two rows have the same primary key. *** ** Changes to rows that do not already appear in the changegroup are ** simply copied into it. Or, if both the new changeset and the changegroup ** contain changes that apply to a

single row, the final contents of the ** changegroup depends on the type of each change, as follows:

```
*** <tr><th style="white-space:pre">Existing Change </th> *** <th style="white-space:pre">New Change </th> *** <th>Output Change *** <tr><td>INSERT <td>INSERT <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td>INSERT <td>UPDATE <td> ** The INSERT change remains in the changegroup. The values in the ** INSERT change are modified as if the row was inserted by the ** existing change and then updated according to the new change. ** <tr><td>INSERT <td>DELETE <td> ** The existing INSERT is removed from the changegroup. The DELETE is ** not added. ** <tr><td>UPDATE <td>INSERT <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td>UPDATE <td>UPDATE <td> ** The existing UPDATE remains within the changegroup. It is amended ** so that the accompanying values are as if the row was updated once ** by the existing change and then again by the new change. ** <tr><td>UPDATE <td>DELETE <td> ** The existing UPDATE is replaced by the new DELETE within the ** changegroup. ** <tr><td>DELETE <td>INSERT <td> ** If one or more of the column values in the row inserted by the ** new change differ from those in the row deleted by the existing ** change, the existing DELETE is replaced by an UPDATE within the ** changegroup. Otherwise, if the inserted row is exactly the same ** as the deleted row, the existing DELETE is simply discarded. ** <tr><td>DELETE <td>UPDATE <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** <tr><td>DELETE <td>DELETE <td> ** The new change is ignored. This case does not occur if the new ** changeset was recorded immediately after the changesets already ** added to the changegroup. ** </table> ***
```

If the new changeset contains changes to a table that is already present ** in the changegroup, then the number of columns and the position of the ** primary key columns for the table must be consistent. If this is not the ** case, this function fails with SQLITE_SCHEMA. If the input changeset ** appears to be corrupt and the corruption is detected, SQLITE_CORRUPT is ** returned. Or, if an out-of-memory condition occurs during processing, this ** function returns SQLITE_NOMEM. In all cases, if an error occurs the ** final contents of the changegroup is undefined. ** * If no error occurs, SQLITE_OK is returned.

label: code-design

80567. Outputs

80568. ** CAPI3REF: Device Characteristics *** ** The xDeviceCharacteristics method of the [sqlite3_io_methods] ** object returns an integer which is a vector of these ** bit values expressing I/O characteristics of the mass storage ** device that holds the file that the [sqlite3_io_methods] ** refers to. *** ** The SQLITE_IOCAP_ATOMIC property means that all writes of ** any size are atomic. The SQLITE_IOCAP_ATOMICnnn values ** mean that writes of blocks that are nnn bytes in size and ** are aligned to an address which is an integer multiple of ** nnn are atomic. The SQLITE_IOCAP_SAFE_APPEND value means ** that when data is appended to a file, the data is appended ** first then the size of the file is extended, never the other ** way around. The SQLITE_IOCAP_SEQUENTIAL property means that ** information is written to disk in the same order as calls ** to xWrite(). The SQLITE_IOCAP_POWERSAFE_OVERWRITE property means that ** after reboot following a crash or power loss, the only bytes in a ** file that were written at the application level might have changed ** and that adjacent bytes, even bytes within the same sector are ** guaranteed to be unchanged. The SQLITE_IOCAP_UNDELETABLE_WHEN_OPEN ** flag indicates that a file cannot be deleted when open. The ** SQLITE_IOCAP_IMMUTABLE flag indicates that the file is on ** read-only media and cannot be changed even by processes with ** elevated privileges.

80569. function to free pUser

80570. Number of result rows written here

80571. Name of attached database (or NULL)

80572. Number of pending entries in the queue

80573. **comment:** ** These no-op macros are used in front of interfaces to mark those ** interfaces as either deprecated or experimental. New applications ** should not use deprecated interfaces - they are supported for backwards ** compatibility only. Application writers should be aware that ** experimental interfaces are subject to change in point releases. *** ** These macros used to resolve to various kinds of compiler magic that ** would generate warning messages when they were used. But that ** compiler magic ended up generating such a flurry of bug reports ** that we have taken it all out and gone back to using simple ** noop macros.

label: code-design

80574. ** CAPI3REF: Session Object Handle

80575. ** CAPI3REF: Obtain A Composite Changeset From A Changegroup *** ** Obtain a buffer containing a changeset (or patchset) representing the ** current contents of the changegroup. If the inputs to the changegroup ** were themselves changesets, the output is a changeset. Or, if the ** inputs were patchsets, the output is also a patchset. ** ** As with the output of the sqlite3session_changeset() and ** sqlite3session_patchset() functions, all changes related to a single ** table are grouped together in the output of this function. Tables appear ** in the same order as for the very first changeset added to the changegroup. ** If the second or subsequent changesets added to the changegroup contain ** changes for tables that do not appear in the first changeset, they are ** appended onto the end of the output changeset, again in the order in ** which they are first encountered. *** ** If an error occurs, an SQLite error code is returned and the output ** variables (*pnData) and (*ppData) are set to 0. Otherwise, SQLITE_OK ** is returned and the output variables are set to the size of and a ** pointer to the output buffer, respectively. In this case it is the ** responsibility of the caller to eventually free the buffer using a ** call to sqlite3_free().

80576. int int

80577. Database to be rekeyed

80578. Connection handle

80579. Pointer to buffer containing changeset A

80580. ** CAPI3REF: Return The Filename For A Database Connection ** METHOD: sqlite3 *** ** ^The sqlite3_db_filename(D,N) interface returns a pointer to a filename ** associated with database N of connection D. ^The main database file ** has the name "main". If there is no attached database N on the database ** connection D, or if database N is a temporary or in-memory database, then ** a NULL pointer is returned. *** ** ^The filename returned by this function is the output of the ** xFullPathname method of the [VFS]. ^In other words, the filename ** will be an absolute pathname, even if the filename used ** to open the database originally was a URI or relative pathname.

80581. The content of the page

80582. ** CAPI3REF: Compare the ages of two snapshot handles. ** EXPERIMENTAL ** ** The sqlite3_snapshot_cmp(P1, P2) interface is used to compare the ages ** of two valid snapshot handles. *** ** If the two snapshot handles are not associated with the same database ** file, the result of the comparison is undefined. ** Additionally, the result of the comparison is only valid if both of the ** snapshot handles were obtained by calling sqlite3_snapshot_get() since the ** last time the wal file was deleted. The wal file is deleted when the ** database is changed back to rollback mode or when the number of database ** clients drops to zero. If either snapshot handle was obtained before the ** wal file was last deleted, the value returned by this function ** is undefined. *** ** Otherwise, this API returns a negative value if P1 refers to an older ** snapshot than P2, zero if the two handles refer to the same database ** snapshot, and a positive value if P1 is a newer snapshot than P2.

80583. Zero or more SQLITE_PREPARE_flags

80584. Error message from sqlite3_mprintf()

80585. Pointer to buffer containing token

80586. Table Name Module Name

80587. ** CAPI3REF: Number Of SQL Parameters ** METHOD: sqlite3_stmt *** ** ^This routine can be used to find the number of [SQL parameters] ** in a [prepared statement]. SQL parameters are tokens of the ** form "?", "?NNN", ":AAA", "\$AAA", or "@AAA" that serve as ** placeholders for values that are [sqlite3_bind_blob | bound] ** to the parameters at a later time. *** ** ^This routine actually returns the index of the largest (rightmost) ** parameter. For all forms except ?NNN, this will correspond to the ** number of unique parameters. If parameters of the ?NNN form are used, ** there may be gaps in the list.^ ** See also: [sqlite3_bind_blob|sqlite3_bind()], ** [sqlite3_bind_parameter_name|sqlite3_bind_parameter_name0], and ** [sqlite3_bind_parameter_index|sqlite3_bind_parameter_index0].

80588. **comment:** ** CAPI3REF: Configuring The SQLite Library *** ** The sqlite3_config() interface is used to make global configuration ** changes to SQLite in order to tune SQLite to the specific needs of ** the application. The default configuration is recommended for most ** applications and so this routine is usually not necessary. It is ** provided to support rare applications with unusual needs. *** ** The sqlite3_config() interface is not threadsafe. The application ** must ensure that no other SQLite interfaces are invoked by other ** threads while sqlite3_config() is running. ** ** The sqlite3_config() interface ** may only be invoked prior to library initialization using ** [sqlite3_initialize()] or after shutdown by [sqlite3_shutdown()]. ** ^If sqlite3_config() is called after [sqlite3_initialize()] and before ** [sqlite3_shutdown()] then it will return SQLITE_MISUSE. ** Note, however, that ^sqlite3_config() can be called as part of the ** implementation of an application-defined [sqlite3_os_init()]. *** ** The first argument to sqlite3_config() is an integer ** [configuration option] that determines ** what property of SQLite is to be configured. Subsequent arguments ** vary depending on the [configuration option] ** in the first argument. *** ** ^When a configuration option is set, sqlite3_config() returns [SQLITE_OK]. ** ^If the option is unknown or SQLite is unable to set the option ** then this routine returns a non-zero [error code].

label: code-design

80589. sqlite3_create_collation only

80590. Byte offset of end of token within input text

80591. ** CAPI3REF: SQL Trace Hook ** METHOD: sqlite3 *** ^The sqlite3_trace_v2(D,M,X,P) interface registers a trace callback ** function X against [database connection] D, using property mask M ** and context pointer P. ^If the X callback is ** NULL or if the M mask is zero, then tracing is disabled. The ** M argument should be the bitwise OR-ed combination of ** zero or more [SQLITE_TRACE] constants. *** ^Each call to either sqlite3_trace() or sqlite3_trace_v2() overrides ** (cancels) any prior calls to sqlite3_trace() or sqlite3_trace_v2(). *** ^The X callback is invoked whenever any of the events identified by ** mask M occur. ^The integer return value from the callback is currently ** ignored, though this may change in future releases. Callback ** implementations should return zero to ensure future compatibility. *** ^A trace callback is invoked with four arguments: callback(T,C,P,X). *** ^The T argument is one of the [SQLITE_TRACE] ** constants to indicate why the callback was invoked. *** ^The C argument is a copy of the context pointer. ** The P and X arguments are pointers whose meanings depend on T. *** The sqlite3_trace_v2() interface is intended to replace the legacy ** interfaces [sqlite3_trace()] and [sqlite3_profile()], both of which ** are deprecated.

80592. ** CAPI3REF: File Locking Levels *** SQLite uses one of these integer values as the second ** argument to calls it makes to the xLock() and xUnlock() methods ** of an [sqlite3_io_methods] object.

80593. Context for returning result/error

80594. **comment:** ** CAPI3REF: Synchronization Type Flags *** When SQLite invokes the xSync() method of an ** [sqlite3_io_methods] object it uses a combination of ** these integer values as the second argument. *** When the SQLITE_SYNC_DATAONLY flag is used, it means that the ** sync operation only needs to flush data to mass storage. Inode ** information need not be flushed. If the lower four bits of the flag ** equal SQLITE_SYNC_NORMAL, that means to use normal fsync() semantics. ** If the lower four bits equal SQLITE_SYNC_FULL, that means ** to use Mac OS X style fullsync instead of fsync(). *** ** Do not confuse the SQLITE_SYNC_NORMAL and SQLITE_SYNC_FULL flags ** with the [PRAGMA synchronous]=NORMAL and [PRAGMA synchronous]=FULL ** settings. The [synchronous pragma] determines when calls to the ** xSync VFS method occur and applies uniformly across all platforms. ** The SQLITE_SYNC_NORMAL and SQLITE_SYNC_FULL flags determine how ** energetic or rigorous or forceful the sync operations are and ** only make a difference on Mac OSX for the default SQLite code. ** (Third-party VFS implementations might also make the distinction ** between SQLITE_SYNC_NORMAL and SQLITE_SYNC_FULL, but among the ** operating systems natively supported by SQLite, only Mac OSX ** cares about the difference.)

label: code-design

80595. OUT: Write the score here

80596. **comment:** sqlite3_malloc()

label: code-design

80597. ** Structures used by the virtual table interface

80598. **comment:** ** CAPI3REF: Memory Allocation Routines *** An instance of this object defines the interface between SQLite ** and low-level memory allocation routines. *** This object is used in only one place in the SQLite interface. ** A pointer to an instance of this object is the argument to ** [sqlite3_config()] when the configuration option is ** [SQLITE_CONFIG_MALLOC] or [SQLITE_CONFIG_GETMALLOC]. ** By creating an instance of this object ** and passing it to [sqlite3_config()][SQLITE_CONFIG_MALLOC] ** during configuration, an application can specify an alternative ** memory allocation subsystem for SQLite to use for all of its ** dynamic memory needs. *** Note that SQLite comes with several [built-in memory allocators] ** that are perfectly adequate for the overwhelming majority of applications ** and that this object is only useful to a tiny minority of applications ** with specialized memory allocation requirements. This object is ** also used during testing of SQLite in order to specify an alternative ** memory allocator that simulates memory out-of-memory conditions in ** order to verify that SQLite recovers gracefully from such ** conditions. *** The xMalloc, xRealloc, and xFree methods must work like the ** malloc(), realloc() and free() functions from the standard C library. ** ^SQLite guarantees that the second argument to ** xRealloc is always a value returned by a prior call to xRoundup. ** ^xSize should return the allocated size of a memory allocation ** previously obtained from xMalloc or xRealloc. The allocated size ** is always at least as big as the requested size but may be larger. ** ^The xRoundup method returns what would be the allocated size of ** a memory allocation given a particular requested size. Most memory ** allocators round up memory allocations at least to the next multiple ** of 8. Some allocators round up to a larger multiple or to a power of 2. ** Every memory allocation request coming in through [sqlite3_malloc()] ** or [sqlite3_realloc()] first calls xRoundup. If xRoundup returns 0, ** that causes the corresponding memory allocation to fail. ** ^The xInit method initializes the memory allocator. For example, ** it might allocate any required mutexes or initialize internal data ** structures. The xShutdown method is invoked (indirectly) by ** [sqlite3_shutdown()] and should deallocate any resources acquired ** by xInit. The pAppData pointer is used as the only parameter to ** xInit and xShutdown. ** ^SQLite holds the [SQLITE_MUTEX_STATIC_MASTER] mutex when it invokes ** the xInit method, so the xInit method need not be threadsafe. The ** xShutdown method is only called from [sqlite3_shutdown()] so it does ** not need to be threadsafe either. For all other methods, SQLite ** holds the [SQLITE_MUTEX_STATIC_MEM] mutex as long as the ** [SQLITE_CONFIG_MEMSTATUS] configuration option is turned on (which ** it is by default) and so the methods are automatically serialized. ** However, if [SQLITE_CONFIG_MEMSTATUS] is disabled, then the other ** methods must be threadsafe or else make their own arrangements for ** serialization. ** ^SQLite will never invoke xInit() more than once without an intervening ** call to xShutdown().

label: code-design

80599. Operation terminated by sqlite3_interrupt()

80600. **comment:** ** CAPI3REF: Create Or Redefine SQL Functions ** KEYWORDS: {function creation routines} ** KEYWORDS: {application-defined SQL function} ** KEYWORDS: {application-defined SQL functions} ** METHOD: sqlite3 *** ^These functions (collectively known as "function creation routines") ** are used to add SQL functions or aggregates or to redefine the behavior ** of existing SQL functions or aggregates. The only differences between ** these routines are the text encoding expected for ** the second parameter (the name of the function being created) ** and the presence or absence of a destructor callback for ** the application data pointer. *** ^The first parameter is the [database connection] to which the SQL ** function is to be added. ^If an application uses more than one database ** connection then application-defined SQL functions must be added ** to each database connection separately. *** ^The second parameter is the name of the SQL function to be created or ** redefined. ^The length of the name is limited to 255 bytes in a UTF-8 ** representation, exclusive of the zero-terminator. ^Note that the name ** length limit is in UTF-8 bytes, not characters nor UTF-16 bytes. ** ^Any attempt to create a function with a longer name ** will result in [SQLITE_MISUSE] being returned. *** ^The third parameter (nArg) ** is the number of arguments that the SQL function or ** aggregate takes. ^If this parameter is -1, then the SQL function or ** aggregate may take any number of arguments between 0 and the limit ** set by [sqlite3_limit](SQLITE_LIMIT_FUNCTION_ARG). If the third ** parameter is less than -1 or greater than 127 then the behavior is ** undefined. *** ^The fourth parameter, eTextRep, specifies what ** [SQLITE_UTF8 | text encoding] this SQL function prefers for ** its parameters. The application should set this parameter to ** [SQLITE_UTF16LE] if the function implementation invokes ** [sqlite3_value_text16le()] on an input, or [SQLITE_UTF16BE] if the ** implementation invokes [sqlite3_value_text16be()] on an input, or ** [SQLITE_UTF16] if [sqlite3_value_text16()] is used, or [SQLITE_UTF8] ** otherwise. ^The same SQL function may be registered multiple times using ** different preferred text encodings, with different implementations for ** each encoding. ** ^When multiple implementations of the same function are available, SQLite ** will pick the one that involves the least amount of data conversion. *** ^The fourth parameter may optionally be ORed with [SQLITE_DETERMINISTIC] ** to signal that the function will always return the same result given ** the same inputs within a single SQL statement. Most SQL functions are ** deterministic. The built-in [random()] SQL function is an example of a ** function that is not deterministic. The SQLite query planner is able to ** perform additional optimizations on deterministic functions, so use ** of the [SQLITE_DETERMINISTIC] flag is recommended where possible. *** ^The fifth parameter is an arbitrary pointer. The implementation of the ** function can gain access to this pointer using [sqlite3_user_data()].** ^The sixth, seventh and eighth parameters, xFunc, xStep and xFinal, are ** pointers to C-language functions that implement the SQL function or ** aggregate. ^A scalar SQL function requires an implementation of the xFunc ** callback only; NULL pointers must be passed as the xStep and xFinal ** parameters. ^An aggregate SQL function requires an implementation of xStep ** and xFinal and NULL pointer must be passed for xFunc. ^To delete an existing ** SQL function or aggregate, pass NULL pointers for all three function ** callbacks. ** ^If the ninth parameter to sqlite3_create_function_v2() is not NULL, ** then it is destructor for the application data pointer. ** The destructor is invoked when the function is deleted, either by being ** overloaded or when the database connection closes. ** ^The destructor is also invoked if the call to ** sqlite3_create_function_v2() fails. ** ^When the destructor callback of the tenth parameter is invoked, it ** is passed a single argument which is a copy of the application data ** pointer which was the fifth parameter to sqlite3_create_function_v2(). ** ^It is permitted to register multiple implementations of the same ** functions with the same name but with either differing numbers of ** arguments or differing preferred text encodings. ^SQLite will use ** the implementation that most closely matches the way in which the ** SQL function is used. ^A function implementation with a non-negative ** nArg parameter is a better match than a function implementation with ** a negative nArg. ^A function where the preferred text encoding ** matches the database encoding is a better ** match than a function where the encoding is different. ** ^A function where the encoding difference is between UTF16le and UTF16be ** is a closer match than a function where the encoding difference is ** between UTF8 and UTF16. ** ^Built-in functions may be overloaded by new application-defined functions. ** ^An application-defined function is permitted to call other ** SQLite interfaces. However, such calls must not ** close the database connection nor finalize or reset the prepared ** statement in which the function is running.

label: code-design

80601. ** CAPI3REF: A Handle To An Open BLOB ** KEYWORDS: {BLOB handle} {BLOB handles} *** An instance of this object represents an open BLOB on which ** [sqlite3_blob_open | incremental BLOB I/O] can be performed. ** ^Objects of this type are created by [sqlite3_blob_open()] ** and destroyed by

[sqlite3_blob_close()]. ** ^The [sqlite3_blob_read()] and [sqlite3_blob_write()] interfaces ** can be used to read or write small subsections of the BLOB. ** ^The [sqlite3_blob_bytes()] interface returns the size of the BLOB in bytes.

80602. ** CAPI3REF: Retrieving Statement SQL ** METHOD: sqlite3_stmt ** ** ^The sqlite3_sql(P) interface returns a pointer to a copy of the UTF-8 ** SQL text used to create [prepared statement] P if P was ** created by [sqlite3_prepare_v2()], [sqlite3_prepare_v3()], ** [sqlite3_prepare16_v2()], or [sqlite3_prepare16_v3()]. ** ^The sqlite3_expanded_sql(P) interface returns a pointer to a UTF-8 ** string containing the SQL text of prepared statement P with ** [bound parameters] expanded. ** ** ^For example, if a prepared statement is created using the SQL ** text "SELECT \$abc,:xyz" and if parameter \$abc is bound to integer 2345 ** and parameter :xyz is unbound, then sqlite3_sql() will return ** the original string, "SELECT \$abc,:xyz" but sqlite3_expanded_sql() ** will return "SELECT 2345,NULL".) ** ** ^The sqlite3_expanded_sql() interface returns NULL if insufficient memory ** is available to hold the result, or if the result would exceed the ** the maximum string length determined by the [SQLITE_LIMIT_LENGTH]. ** ** ^The [SQLITE_TRACE_SIZE_LIMIT] compile-time option limits the size of ** bound parameter expansions. ^The [SQLITE_OMIT_TRACE] compile-time ** option causes sqlite3_expanded_sql() to always return NULL. ** ** ^The string returned by sqlite3_sql(P) is managed by SQLite and is ** automatically freed when the prepared statement is finalized. ** ^The string returned by sqlite3_expanded_sql(P), on the other hand, ** is obtained from [sqlite3_malloc()] and must be free by the application ** by passing it to [sqlite3_free()].

80603. ** Specify the activation key for a CEROD database. Unless ** activated, none of the CEROD routines will work.

80604. ** CAPI3REF: Run-time Limits ** METHOD: sqlite3 ** ** ^This interface allows the size of various constructs to be limited ** on a connection by connection basis. The first parameter is the ** [database connection] whose limit is to be set or queried. The ** second parameter is one of the [limit categories] that define a ** class of constructs to be size limited. The third parameter is the ** new limit for that construct. ^If the new limit is a negative number, the limit is unchanged. ** ^For each limit category SQLITE_LIMIT_<i>NAME</i> there is a ** [limits | hard upper bound] ** set at compile-time by a C preprocessor macro called ** [limits | SQLITE_MAX_<i>NAME</i>]. ** ^The "_LIMIT_" in the name is changed to "_MAX_").) ** ** ^Attempts to increase a limit above its hard upper bound are ** silently truncated to the hard upper bound. ** ** ^Regardless of whether or not the limit was changed, the ** [sqlite3_limit()] interface returns the prior value of the limit. ** ^Hence, to find the current value of a limit without changing it, ** simply invoke this interface with the third parameter set to -1. ** ** Run-time limits are intended for use in applications that manage ** both their own internal database and also databases that are controlled ** by untrusted external sources. An example application might be a ** web browser that has its own databases for storing history and ** separate databases controlled by JavaScript applications downloaded ** off the Internet. The internal databases can be given the ** large, default limits. Databases managed by external sources can ** be given much smaller limits designed to prevent a denial of service ** attack. Developers might also want to use the [sqlite3_set_authorizer()] ** interface to further control untrusted SQL. The size of the database ** created by an untrusted script can be contained using the ** [max_page_count] [PRAGMA]. ** ** New run-time limit categories may be added in future releases.

80605. ** CAPI3REF: Load The Difference Between Tables Into A Session ** ** If it is not already attached to the session object passed as the first ** argument, this function attaches table zTbl in the same manner as the ** [sqlite3session_attach()] function. If zTbl does not exist, or if it ** does not have a primary key, this function is a no-op (but does not return ** an error). ** ** Argument zFromDb must be the name of a database ("main", "temp" etc.) ** attached to the same database handle as the session object that contains ** a table compatible with the table attached to the session by this function. ** A table is considered compatible if it: *** ** Has the same set of columns declared in the same order, and ** Has the same PRIMARY KEY definition. ** ** ** If the tables are not compatible, SQLITE_SCHEMA is returned. If the tables ** are compatible but do not have any PRIMARY KEY columns, it is not an error ** but no changes are added to the session object. As with other session ** APIs, tables without PRIMARY KEYS are simply ignored. ** ** This function adds a set of changes to the session object that could be ** used to update the table in database zFrom (call this the "from-table") ** so that its content is the same as the table attached to the session ** object (call this the "to-table"). Specifically: *** ** For each row (primary key) that exists in the to-table but not in ** the from-table, an INSERT record is added to the session object. ** ** For each row (primary key) that exists in the to-table but not in ** the from-table, a DELETE record is added to the session object. ** ** For each row (primary key) that exists in both tables, but features ** different non-PK values in each, an UPDATE record is added to the ** session. ** ** ** To clarify, if this function is called and then a changeset constructed ** using [sqlite3session_changeset()], then after applying that changeset to ** database zFrom the contents of the two compatible tables would be ** identical. ** ** It an error if database zFrom does not exist or does not contain the ** required compatible table. ** ** If the operation successful, SQLITE_OK is returned. Otherwise, an SQLite ** error code. In this case, if argument pzErrMsg is not NULL, *pzErrMsg ** may be set to point to a buffer containing an English language error ** message. It is the responsibility of the caller to free this buffer using ** sqlite3_free().

80606. Trigger Name Table Name

80607. comment: previously SQLITE_CONFIG_CHUNKALLOC 12 which is now unused.

label: code-design

80608. The database file is locked

80609. OUT: New session object

80610. The methods above are in version 1 of the sqlite_module object. Those ** below are for version 2 and greater.

80611. ** CAPI3REF: Create An Iterator To Traverse A Changeset ** ** Create an iterator used to iterate through the contents of a changeset. ** If successful, *pp is set to point to the iterator handle and SQLITE_OK ** is returned. Otherwise, if an error occurs, *pp is set to zero and an ** SQLite error code is returned. ** ** The following functions can be used to advance and query a changeset ** iterator created by this function: *** ** [sqlite3changeset_next()] ** [sqlite3changeset_op()] ** [sqlite3changeset_new()] ** [sqlite3changeset_old()] ** ** ** It is the responsibility of the caller to eventually destroy the iterator ** by passing it to [sqlite3changeset_finalize()]. The buffer containing the ** changeset (pChangeset) must remain valid until after the iterator is ** destroyed. ** ** Assuming the changeset blob was created by one of the ** [sqlite3session_changeset()], [sqlite3changeset_concat()] or ** [sqlite3changeset_invert()] functions, all changes within the changeset ** that apply to a single table are grouped together. This means that when ** an application iterates through a changeset using an iterator created by ** this function, all changes that relate to a single table are visited ** consecutively. There is no chance that the iterator will visit a change ** the applies to table X, then one for table Y, and then later on visit ** another change for table X.

80612. Copy of third arg to _filter_table()

80613. ** CAPI3REF: Error Codes And Messages ** METHOD: sqlite3 ** ** ^If the most recent sqlite3_* API call associated with ** [database connection] D failed, then the sqlite3_errcode(D) interface ** returns the numeric [result code] or [extended result code] for that ** API call. ** If the most recent API call was successful, ** then the return value from sqlite3_errcode() is undefined. ** ^The sqlite3_extended_errcode() ** interface is the same except that it always returns the ** [extended result code] even when extended result codes are ** disabled. ** ** ^The sqlite3_errmsg() and sqlite3_errmsg16() return English-language ** text that describes the error, as either UTF-8 or UTF-16 respectively. ** ^Memory to hold the error message string is managed internally. ** The application does not need to worry about freeing the result. ** However, the error string might be overwritten or deallocated by ** subsequent calls to other SQLite interface functions. ** ** ^The sqlite3_errstr() interface returns the English-language text ** that describes the [result code], as UTF-8. ** ^Memory to hold the error message string is managed internally ** and must not be freed by the application). ** ** When the serialized [threading mode] is in use, it might be the ** case that a second error occurs on a separate thread in between ** the time of the first error and the call to these interfaces. ** When that happens, the second error will be reported since these ** interfaces always report the most recent result. To avoid ** this, each thread can obtain exclusive use of the [database connection] D ** by invoking [sqlite3_mutex_enter]([sqlite3_db_mutex](D)) before beginning ** to use D and invoking [sqlite3_mutex_leave]([sqlite3_db_mutex](D)) after ** all calls to the interfaces listed here are completed. ** ** If an interface fails with SQLITE_MISUSE, that means the interface ** was invoked incorrectly by the application. In that case, the ** error code and message may or may not be set.

80614. String or BLOB exceeds size limit

80615. ** CAPI3REF: Data Change Notification Callbacks ** METHOD: sqlite3 ** ** ^The sqlite3_update_hook() interface registers a callback function ** with the [database connection] identified by the first argument ** to be invoked whenever a row is updated, inserted or deleted in ** a [rowid table]. ** ^Any callback set by a previous call to this function ** for the same database connection is overridden. ** ** ^The second argument is a pointer to the function to invoke when a ** row is updated, inserted or deleted in a rowid table. ** ^The first argument to the callback is a copy of the third argument ** to sqlite3_update_hook(). ** ^The second callback argument is one of [SQLITE_INSERT], [SQLITE_DELETE], ** or [SQLITE_UPDATE], depending on the operation that caused the callback ** to be invoked. ** ^The third and fourth arguments to the callback contain pointers to the ** database and table name containing the affected row. ** ^The final callback parameter is the [rowid] of the row. ** ^In the case of an update, this is the [rowid] after the update takes place. ** ** ^The update hook is not invoked when internal system tables are ** modified (i.e. sqlite_master and sqlite_sequence).) ** ** ^The update hook is not invoked when [WITHOUT ROWID] tables are modified. ** ** ^In the current implementation, the update hook ** is not invoked when conflicting rows are deleted because of an ** [ON CONFLICT | ON CONFLICT REPLACE] clause. ^Nor is the update hook ** invoked when rows are deleted using the [truncate optimization]. ** The exceptions defined in this paragraph might change in a future ** release of SQLite. ** ** The update hook implementation must not do anything that will modify ** the database connection that invoked the update hook. Any actions ** to modify the database connection must be deferred until after the ** completion of the [sqlite3_step()] call that triggered the update hook. ** Note that [sqlite3_prepare_v2()] and [sqlite3_step()] both modify their ** database connections for the meaning of "modify" in this paragraph. ** ** ^The sqlite3_update_hook(D,C,P) function ** returns the P argument from the previous call ** on the same [database connection] D, or NULL for ** the first call on D. ** ** See also the [sqlite3_commit_hook()], [sqlite3_rollback_hook()], ** and [sqlite3_prepupdate_hook()] interfaces.

80616. Changeset blob

80617. Inputs

80618. **comment:** ** The interface to the virtual-table mechanism defined above (back up ** to a comment remarkably similar to this one) is currently considered ** to be experimental. The interface might change in incompatible ways. ** If this is a problem for you, do not use the interface at this time. *** ** When the virtual-table mechanism stabilizes, we will declare the ** interface fixed, support it indefinitely, and remove this comment.

label: code-design

80619. ** CAPI3REF: 64-Bit Integer Types ** KEYWORDS: sqlite_int64 sqlite_uint64 *** ** Because there is no cross-platform way to specify 64-bit integer types ** SQLite includes typedefs for 64-bit signed and unsigned integers. ** ** The sqlite3_int64 and sqlite3_uint64 are the preferred type definitions. ** The sqlite_int64 and sqlite_uint64 types are supported for backwards ** compatibility only. ** ** ^The sqlite3_int64 and sqlite_int64 types can store integer values ** between -9223372036854775808 and +9223372036854775807 inclusive. ^The ** sqlite3_uint64 and sqlite_uint64 types can store integer values ** between 0 and +18446744073709551615 inclusive.

80620. ** CAPI3REF: Prepare Flags ** ** These constants define various flags that can be passed into ** "prepFlags" parameter of the [sqlite3_prepare_v3()] and ** [sqlite3_prepare16_v3()] interfaces. ** ** New flags may be added in future releases of SQLite. ** ** <dl> ** [[SQLITE_PREPARE_PERSISTENT]]
^(<dt>SQLITE_PREPARE_PERSISTENT</dt> ** <dd>The SQLITE_PREPARE_PERSISTENT flag is a hint to the query planner ** that the prepared statement will be retained for a long time and ** probably reused many times.)^ ^Without this flag, [sqlite3_prepare_v3()] ** and [sqlite3_prepare16_v3()] assume that the prepared statement will ** be used just once or at most a few times and then destroyed using ** [sqlite3_finalize()] relatively soon. The current implementation acts ** on this hint by avoiding the use of [lookaside memory] so as not to ** deplete the limited store of lookaside memory. Future versions of ** SQLite may act on this hint differently. ** </dl>

80621. Wait for writers, then checkpoint

80622. ** CAPI3REF: Opening A New Database Connection ** CONSTRUCTOR: sqlite3 *** ** ^These routines open an SQLite database file as specified by the ** filename argument. ^The filename argument is interpreted as UTF-8 for ** sqlite3_open() and sqlite3_open_v2() and as UTF-16 in the native byte ** order for sqlite3_open16(). ^A [database connection] handle is usually ** returned in *ppDb, even if an error occurs. The only exception is that ** if SQLite is unable to allocate memory to hold the [sqlite3] object, ** a NULL will be written into *ppDb instead of a pointer to the [sqlite3] ** object.^ ^If the database is opened (and/or created) successfully, then ** [SQLITE_OK] is returned. Otherwise an [error code] is returned.)^ ^The ** [sqlite3_errmsg0()] or [sqlite3_errmsg160()] routines can be used to obtain ** an English language description of the error following a failure of any ** of the sqlite3_open() routines. *** ** ^The default encoding will be UTF-8 for databases created using ** sqlite3_open() or sqlite3_open_v2(). ^The default encoding for databases ** created using sqlite3_open16() will be UTF-16 in the native byte order. *** ** Whether or not an error occurs when it is opened, resources ** associated with the [database connection] handle should be released by ** passing it to [sqlite3_close()] when it is no longer required. *** ** The sqlite3_open_v2() interface works like sqlite3_open() ** except that it accepts two additional parameters for additional control ** over the new database connection. ^The flags parameter to ** sqlite3_open_v2() can take one of ** the following three values, optionally combined with the ** [SQLITE_OPEN_NOMUTEX], [SQLITE_OPEN_FULLMUTEX], [SQLITE_OPEN_SHARED_CACHE], ** [SQLITE_OPEN_PRIVATECACHE], and/or [SQLITE_OPEN_URI] flags:)*** ** <dl> ** (<dt>[SQLITE_OPEN_READONLY]</dt> ** <dd>The database is opened in read-only mode. If the database does not ** already exist, an error is returned.</dd>)*** ** ^(<dt>[SQLITE_OPEN_READWRITE]</dt> ** <dd>The database is opened for reading and writing if possible, or reading ** only if the file is write protected by the operating system. In either ** case the database must already exist, otherwise an error is returned.</dd>)*** ** ^(<dt>[SQLITE_OPEN_READWRITE] | [SQLITE_OPEN_CREATE]</dt> ** <dd>The database is opened for reading and writing, and is created if ** it does not already exist. This is the behavior that is always used for ** sqlite3_open() and sqlite3_open16().</dd>)*** ** <dd> If the 3rd parameter to sqlite3_open_v2() is not one of the ** combinations shown above optionally combined with other ** [SQLITE_OPEN_READONLY | SQLITE_OPEN_* bits] ** then the behavior is undefined. *** ** ^If the [SQLITE_OPEN_NOMUTEX] flag is set, then the database connection ** opens in the multi-thread [threading mode] as long as the single-thread ** mode has not been set at compile-time or start-time. ^If the ** [SQLITE_OPEN_FULLMUTEX] flag is set then the database connection opens ** in the serialized [threading mode] unless single-thread was ** previously selected at compile-time or start-time. ** ^The [SQLITE_OPEN_SHARED_CACHE] flag causes the database connection to be ** eligible to use [shared cache mode], regardless of whether or not shared ** cache is enabled using [sqlite3_enable_shared_cache()]. ^The ** [SQLITE_OPEN_PRIVATECACHE] flag causes the database connection to not ** participate in [shared cache mode] even if it is enabled. *** ** ^The fourth parameter to sqlite3_open_v2() is the name of the ** [sqlite3_vfs] object that defines the operating system interface that ** the new database connection should use. ^If the fourth parameter is ** a NULL pointer then the default [sqlite3_vfs] object is used. *** ** ^If the filename is ":memory:", then a private, temporary in-memory database ** is created for the connection. ^This in-memory database will vanish when ** the database connection is closed. Future versions of SQLite might ** make use of additional special filenames that begin with the ":" character. ** It is recommended that when a database filename actually does begin with ** a ":" character you should prefix the filename with a pathname such as ** "/" to avoid ambiguity. *** ** ^If the filename is an empty string, then a private, temporary ** on-disk database will be created. ^This private database will be ** automatically deleted as soon as the database connection is closed. *** ** [[URI filenames in sqlite3_open()]] <h3>URI Filenames</h3> *** ** ^If [URI filename] interpretation is enabled, and the filename argument ** begins with "file:", then the filename is interpreted as a URI. ^URI ** filename interpretation is enabled if the [SQLITE_OPEN_URI] flag is ** set in the fourth argument to sqlite3_open_v2(), or if it has ** been enabled globally using the [SQLITE_CONFIG_URI] option with the ** [sqlite3_config0()] method or by the [SQLITE_USE_URI] compile-time option. ** As of SQLite version 3.7.7, URI filename interpretation is turned off ** by default, but future releases of SQLite might enable URI filename ** interpretation by default. See "[URI filenames]" for additional ** information. *** ** URI filenames are parsed according to RFC 3986. ^If the URI contains an ** authority, then it must be either an empty string or the string ** "localhost". ^If the authority is not an empty string or "localhost", an ** error is returned to the caller. ^The fragment component of a URI, if ** present, is ignored. *** ** ^SQLite uses the path component of the URI as the name of the disk file ** which contains the database. ^If the path begins with a '/' character, ** then it is interpreted as an absolute path. ^If the path does not begin ** with a '/' (meaning that the authority section is omitted from the URI) ** then the path is interpreted as a relative path. ** ^On windows, the first component of an absolute path ** is a drive specification (e.g. "C:").*** ** [[core URI query parameters]] ** The query component of a URI may contain parameters that are interpreted ** either by SQLite itself, or by a [VFS | custom VFS implementation]. ** SQLite and its built-in [VFSes] interpret the ** following query parameters: *** ** ** vfs: ^The "vfs" parameter may be used to specify the name of ** a VFS object that provides the operating system interface that should ** be used to access the database file on disk. ^If this option is set to ** an empty string the default VFS object is used. ^Specifying an unknown ** VFS is an error. ^If sqlite3_open_v2() is used and the vfs option is ** present, then the VFS specified by the option takes precedence over ** the value passed as the fourth parameter to sqlite3_open_v2(). *** ** mode: ^The mode parameter may be set to either "ro", "rw", ** "rwc", or "memory". Attempting to set it to any other value is ** an error). ** ^If "ro" is specified, then the database is opened for read-only ** access, just as if the [SQLITE_OPEN_READONLY] flag had been set in the ** third argument to sqlite3_open_v2(). ^If the mode option is set to ** "rw", then the database is opened for read-write (but not create) ** access, as if SQLITE_OPEN_READWRITE (but not SQLITE_OPEN_CREATE) had ** been set. ^Value "rwc" is equivalent to setting both ** SQLITE_OPEN_READWRITE and SQLITE_OPEN_CREATE. ^If the mode option is ** set to "memory" then a pure [in-memory database] that never reads ** or writes from disk is used. ^It is an error to specify a value for ** the mode parameter that is less restrictive than that specified by ** the flags passed in the third parameter to sqlite3_open_v2(). *** ** cache: ^The cache parameter may be set to either "shared" or ** "private". ^Setting it to "shared" is equivalent to setting the ** SQLITE_OPEN_SHARED_CACHE bit in the flags argument passed to ** sqlite3_open_v2(). ^Setting the cache parameter to "private" is ** equivalent to setting the SQLITE_OPEN_PRIVATECACHE bit. ** ^If sqlite3_open_v2() is used and the "cache" parameter is present in ** a URI filename, its value overrides any behavior requested by setting ** SQLITE_OPEN_PRIVATECACHE or SQLITE_OPEN_SHARED_CACHE flag. *** ** psow: ^The psow parameter indicates whether or not the ** [powersafe overwrite] property does or does not apply to the ** storage media on which the database file resides. *** ** nolock: ^The nolock parameter is a boolean query parameter ** which if set disables file locking in rollback journal modes. This ** is useful for accessing a database on a filesystem that does not ** support locking. Caution: Database corruption might result if two ** or more processes write to the same database and any one of those ** processes uses nolock=1. *** ** immutable: ^The immutable parameter is a boolean query ** parameter that indicates that the database file is stored on ** read-only media. ^When immutable is set, SQLite assumes that the ** database file cannot be changed, even by a process with higher ** privilege, and so the database is opened read-only and all locking ** and change detection is disabled. Caution: Setting the immutable ** property on a database file that does in fact change can result ** in incorrect query results and/or [SQLITE_CORRUPT] errors. ** See also: [SQLITE_IOCAP_IMMUTABLE]. *** ** ** ^Specifying an unknown parameter in the query component of a URI is not an ** error. Future versions of SQLite might understand additional query ** parameters. See "[query parameters with special meaning to SQLite]" for ** additional information. *** ** [[URI filename examples]] <h3>URI filename examples</h3> *** ** <table border="1" align="center" cellpadding="5"> ** <tr><th> URI filenames </th> Results </tr><td> file:data.db <td> ** Open the file "data.db" in the current directory. ** <tr><td> file:/home/fred/data.db
 ** file:///home/fred/data.db
 ** file://localhost/home/fred/data.db
 <td> ** Open the database file "/home/fred/data.db". ** <tr><td> file://darkstar/home/fred/data.db <td> ** An error. "darkstar" is not a recognized authority. ** <tr><td style="white-space:nowrap"> ** file:///C:/Documents%20and%20Settings/fred/Desktop/data.db ** <td> Windows only: Open the file "data.db" on fred's desktop on drive ** C:. Note that the %20 escaping in this example is not strictly ** necessary - space characters can be used literally ** in URI filenames. ** <tr><td> file:data.db?mode=ro&cache=private <td> ** Open file "data.db" in the current directory for read-only access. ** Regardless of whether or not shared-cache mode is enabled by ** default, use a private cache. ** <tr><td> file:/home/fred/data.db?vfs=unix-dotfile

<td> ** Open file "/home/fred/data.db". Use the special VFS "unix-dotfile" ** that uses dot-files in place of posix advisory locking. ** <tr><td> file:data.db? mode=readonly <td> ** An error. "readonly" is not a valid option for the "mode" parameter. ** </table> ** ** ^URI hexadecimal escape sequences (%HH) are supported within the path and ** query components of a URI. A hexadecimal escape sequence consists of a ** percent sign - "%" - followed by exactly two hexadecimal digits ** specifying an octet value. ^Before the path or query components of a ** URI filename are interpreted, they are encoded using UTF-8 and all ** hexadecimal escape sequences replaced by a single byte containing the ** corresponding octet. If this process generates an invalid UTF-8 encoding, ** the results are undefined. ** ** Note to Windows users: The encoding used for the filename argument ** of sqlite3_open() and sqlite3_open_v2() must be UTF-8, not whatever ** codepage is currently defined. Filenames containing international ** characters must be converted to UTF-8 prior to passing them into ** sqlite3_open() or sqlite3_open_v2(). ** ** Note to Windows Runtime users: The temporary directory must be set ** prior to calling sqlite3_open() or sqlite3_open_v2(). Otherwise, various ** features that require the use of temporary files may fail. ** ** See also: [sqlite3_temp_directory]

80623. ** CAPI3REF: Run-Time Limit Categories ** KEYWORDS: {limit category} {*limit categories} ** ** These constants define various performance limits ** that can be lowered at run-time using [sqlite3_limit()]. ** The synopsis of the meanings of the various limits is shown below. ** Additional information is available at [limits | Limits in SQLite]. ** ** <dl> ** [[SQLITE_LIMIT_LENGTH]] ^<dt>SQLITE_LIMIT_LENGTH</dt> ** <dd>The maximum size of any string or BLOB or table row, in bytes.<dd>^ ** [[SQLITE_LIMIT_SQL_LENGTH]] ^<dt>SQLITE_LIMIT_SQL_LENGTH</dt> ** <dd>The maximum length of an SQL statement, in bytes.<dd>^ ** [[SQLITE_LIMIT_COLUMN]] ^<dt>SQLITE_LIMIT_COLUMN</dt> ** <dd>The maximum number of columns in a table definition or in the ** result set of a [SELECT] or the maximum number of columns in an index ** or in an ORDER BY or GROUP BY clause.<dd>^ ** [[SQLITE_LIMIT_EXPR_DEPTH]] ^<dt>SQLITE_LIMIT_EXPR_DEPTH</dt> ** <dd>The maximum depth of the parse tree on any expression.<dd>^ ** [[SQLITE_LIMIT_COMPOUND_SELECT]] ^<dt>SQLITE_LIMIT_COMPOUND_SELECT</dt> ** <dd>The maximum number of terms in a compound SELECT statement.<dd>^ ** [[SQLITE_LIMIT_VDBE_OP]] ^<dt>SQLITE_LIMIT_VDBE_OP</dt> ** <dd>The maximum number of instructions in a virtual machine program ** used to implement an SQL statement. If [sqlite3_prepare_v2()] or ** the equivalent tries to allocate space for more than this many opcodes ** in a single prepared statement, an SQLITE_NOMEM error is returned.<dd>^ ** [[SQLITE_LIMIT_FUNCTION_ARG]] ^<dt>SQLITE_LIMIT_FUNCTION_ARG</dt> ** <dd>The maximum number of arguments on a function.<dd>^ ** [[SQLITE_LIMIT_ATTACHED]] ^<dt>SQLITE_LIMIT_ATTACHED</dt> ** <dd>The maximum number of [ATTACH | attached databases].<dd> ** [[SQLITE_LIMIT_LIKE_PATTERN_LENGTH]] ** ^<dt>SQLITE_LIMIT_LIKE_PATTERN_LENGTH</dt> ** <dd>The maximum length of the pattern argument to the [LIKE] or ** [GLOB] operators.<dd>^ ** [[SQLITE_LIMIT_VARIABLE_NUMBER]] ** ^<dt>SQLITE_LIMIT_VARIABLE_NUMBER</dt> ** <dd>The maximum index number of any [parameter] in an SQL statement.^ ** [[SQLITE_LIMIT_TRIGGER_DEPTH]] ^<dt>SQLITE_LIMIT_TRIGGER_DEPTH</dt> ** <dd>The maximum depth of recursion for triggers.<dd>^ ** [[SQLITE_LIMIT_WORKER_THREADS]] ^<dt>SQLITE_LIMIT_WORKER_THREADS</dt> ** <dd>The maximum number of auxiliary worker threads that a single ** [prepared statement] may start.<dd>^ ** </dl>

80624. Do as much as possible w/o blocking

80625. Number of terms in the ORDER BY clause

80626. Flags that may be passed by the tokenizer implementation back to FTS5 ** as the third argument to the supplied xToken callback.

80627. ** CAPI3REF: Database Connection Handle ** KEYWORDS: {database connection} {database connections} ** ** Each open SQLite database is represented by a pointer to an instance of ** the opaque structure named "sqlite3". It is useful to think of an sqlite3 ** pointer as an object. The [sqlite3_open()], [sqlite3_open16()], and ** [sqlite3_open_v2()] interfaces are its constructors, and [sqlite3_close()] ** and [sqlite3_close_v2()] are its destructors. There are many other ** interfaces (such as ** [sqlite3_prepare_v2()], [sqlite3_create_function()], and ** [sqlite3_busy_timeout()] to name but three) that are methods on an ** sqlite3 object.

80628. ** CAPI3REF: Database Connection Status ** METHOD: sqlite3 ** ** ^This interface is used to retrieve runtime status information ** about a single [database connection]. ^The first argument is the ** database connection object to be interrogated. ^The second argument ** is an integer constant, taken from the set of ** [SQLITE_DBSTATUS options], that ** determines the parameter to interrogate. The set of ** [SQLITE_DBSTATUS options] is likely ** to grow in future releases of SQLite. ** ** ^The current value of the requested parameter is written into *pCur ** and the highest instantaneous value is written into *pHiwr. ^If ** the resetFlg is true, then the highest instantaneous value is ** reset back down to the current value. ** ** ^The sqlite3_db_status() routine returns SQLITE_OK on success and a ** non-zero [error code] on failure. ** ** See also: [sqlite3_status()] and [sqlite3_stmt_status()].

80629. ** CAPI3REF: Status Parameters for prepared statements ** KEYWORDS: {SQLITE_STMTSTATUS counter} {SQLITE_STMTSTATUS counters} ** ** These preprocessor macros define integer codes that name counter ** values associated with the [sqlite3_stmt_status()] interface. ** The meanings of the various counters are as follows: ** ** <dl> ** [[SQLITE_STMTSTATUS_FULLSCAN_STEP]] <dt>SQLITE_STMTSTATUS_FULLSCAN_STEP</dt> ** <dd>^This is the number of times that SQLite has stepped forward in ** a table as part of a full table scan. Large numbers for this counter ** may indicate opportunities for performance improvement through ** careful use of indices.<dd>^ ** [[SQLITE_STMTSTATUS_SORT]] <dt>SQLITE_STMTSTATUS_SORT</dt> ** <dd>^This is the number of sort operations that have occurred. ** A non-zero value in this counter may indicate an opportunity to ** improvement performance through careful use of indices.<dd>^ ** [[SQLITE_STMTSTATUS_AUTOINDEX]] <dt>SQLITE_STMTSTATUS_AUTOINDEX</dt> ** <dd>^This is the number of rows inserted into transient indices that ** were created automatically in order to help joins run faster. ** A non-zero value in this counter may indicate an opportunity to ** improvement performance by adding permanent indices that do not ** need to be reinitialized each time the statement is run.<dd>^ ** [[SQLITE_STMTSTATUS_VM_STEP]] <dt>SQLITE_STMTSTATUS_VM_STEP</dt> ** <dd>^This is the number of virtual machine operations executed ** by the prepared statement if that number is less than or equal ** to 2147483647. The number of virtual machine operations can be ** used as a proxy for the total work done by the prepared statement. ** If the number of virtual machine operations exceeds 2147483647 ** then the value returned by this statement status code is undefined. ** [[SQLITE_STMTSTATUS_REPREPARE]] <dt>SQLITE_STMTSTATUS_REPREPARE</dt> ** <dd>^This is the number of times that the prepare statement has been ** automatically regenerated due to schema changes or change to ** [bound parameters] that might affect the query plan. ** [[SQLITE_STMTSTATUS_RUN]] <dt>SQLITE_STMTSTATUS_RUN</dt> ** <dd>^This is the number of times that the prepared statement has ** been run. A single "run" for the purposes of this counter is one ** or more calls to [sqlite3_step()] followed by a call to [sqlite3_reset()]. ** The counter is incremented on the first [sqlite3_step()] call of each ** cycle. ** [[SQLITE_STMTSTATUS_MEMUSED]] <dt>SQLITE_STMTSTATUS_MEMUSED</dt> ** <dd>^This is the approximate number of bytes of heap memory ** used to store the prepared statement. ^This value is not actually ** a counter, and so the resetFlg parameter to sqlite3_stmt_status() ** is ignored when the opcode is SQLITE_STMTSTATUS_MEMUSED. ** </dd> ** </dl>

80630. ** CAPI3REF: Number Of Columns In A Result Set ** METHOD: sqlite3_stmt ** ** ^Return the number of columns in the result set returned by the ** [prepared statement]. ^If this routine returns 0, that means the ** [prepared statement] returns no data (for example an [UPDATE]). ** ^However, just because this routine returns a positive number does not ** mean that one or more rows of data will be returned. ^A SELECT statement ** will always have a positive sqlite3_column_count() but depending on the ** WHERE clause constraints and the table content, it might return no rows. ** ** See also: [sqlite3_data_count()]

80631. Number of result columns written here

80632. SQL statement, UTF-8 encoded

80633. OUT: Size of WAL log in frames

80634. The database disk image is malformed

80635. Results of the query

80636. Argument to xInit() and xShutdown()

80637. Size of changeset in bytes

80638. OUTPUT: True if column part of PK

80639. ** CAPI3REF: Collation Needed Callbacks ** METHOD: sqlite3 ** ** ^To avoid having to register all collation sequences before a database ** can be used, a single callback function may be registered with the ** [database connection] to be invoked whenever an undefined collation ** sequence is required. ** ** ^If the function is registered using the sqlite3_collation_needed() API, ** then it is passed the names of undefined collation sequences as strings ** encoded in UTF-8. ^If sqlite3_collation_needed16() is used, ** the names are passed as UTF-16 in machine native byte order. ** ^A call to either function replaces the existing collation-needed callback. ** ** ^When the callback is invoked, the first argument passed is a copy ** of the second argument to sqlite3_collation_needed() or ** sqlite3_collation_needed16(). The second argument is the database ** connection. The third argument is one of [SQLITE_UTF8], [SQLITE_UTF16BE], ** or [SQLITE_UTF16LE], indicating the most desirable form of the collation ** sequence function required. The fourth parameter is the name of the ** required collation sequence.)^ ** ** The callback function should register the desired collation using ** [sqlite3_create_collation()], [sqlite3_create_collation16()], or ** [sqlite3_create_collation_v2()].

80640. The double-precision datatype used by RTree depends on the ** SQLITE_RTREE_INT_ONLY compile-time option.

80641. ** 2014 May 31 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: ** ** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. ** ***** Interfaces to extend FTS5. Using the interfaces defined in this file, ** FTS5 may be extended with: ** ** * custom tokenizers, and ** * custom auxiliary functions.

80642. sqlite3_mutex_methods*

80643. ** Change the key on an open database. If the current database is not ** encrypted, this routine will encrypt it. If pNew==0 or nNew==0, the ** database is decrypted. *** The code to implement this API is not available in the public release ** of SQLite.

80644. Number of function parameters

80645. OUT: Number of columns in table

80646. ** CAPI3REF: Online Backup Object ** ** The sqlite3_backup object records state information about an ongoing ** online backup operation. ^The sqlite3_backup object is created by ** a call to [sqlite3_backup_init()] and is destroyed by a call to ** [sqlite3_backup_finish()]. *** See Also: [Using the SQLite Online Backup API]

80647. Number of bytes in buffer pB

80648. ** CAPI3REF: Authorizer Return Codes ** ** The [sqlite3_set_authorizer | authorizer callback function] must ** return either [SQLITE_OK] or one of these two constants in order ** to signal SQLite whether or not the action is permitted. See the ** [sqlite3_set_authorizer | authorizer documentation] for additional ** information. *** Note that SQLITE_IGNORE is also used as a [conflict resolution mode] ** returned from the [sqlite3_vtab_on_conflict()] interface.

80649. Same position as prev. token

80650. The database schema changed

80651. **comment:** ** CAPI3REF: Tracing And Profiling Functions ** METHOD: sqlite3 ** ** These routines are deprecated. Use the [sqlite3_trace_v2()] interface ** instead of the routines described here. *** These routines register callback functions that can be used for ** tracing and profiling the execution of SQL statements. *** ^The callback function registered by sqlite3_trace() is invoked at ** various times when an SQL statement is being run by [sqlite3_step()]. ** ^The sqlite3_trace() callback is invoked with a UTF-8 rendering of the ** SQL statement text as the statement first begins executing. ** ^Additional sqlite3_trace() callbacks might occur ** as each triggered subprogram is entered. The callbacks for triggers ** contain a UTF-8 SQL comment that identifies the trigger. ** ^The [SQLITE_TRACE_SIZE_LIMIT] compile-time option can be used to limit ** the length of [bound parameter] expansion in the output of sqlite3_trace(). ** ^The callback function registered by sqlite3_profile() is invoked ** as each SQL statement finishes. ^The profile callback contains ** the original statement text and an estimate of wall-clock time ** of how long that statement took to run. ^The profile callback ** time is in units of nanoseconds, however the current implementation ** is only capable of millisecond resolution so the six least significant ** digits in the time are meaningless. Future versions of SQLite ** might provide greater resolution on the profiler callback. The ** sqlite3_profile() function is considered experimental and is ** subject to change in future versions of SQLite.

label: code-design

80652. Name of this virtual file system

80653. Constraint operator

80654. ** CAPI3REF: User Data For Functions ** METHOD: sqlite3_context ** ** ^The sqlite3_user_data() interface returns a copy of ** the pointer that was the pUserData parameter (the 5th parameter) ** of the [sqlite3_create_function()] ** and [sqlite3_create_function16()] routines that originally ** registered the application defined function. *** This routine must be called from the same thread in which ** the application-defined function is running.

80655. ** CAPI3REF: Low-level system error code ** ** ^Attempt to return the underlying operating system error code or error ** number that caused the most recent I/O error or failure to open a file. ** The return value is OS-dependent. For example, on unix systems, after ** [sqlite3_open_v2()] returns [SQLITE_CANTOPEN], this interface could be ** called to get back the underlying "errno" that caused the problem, such ** as ENOSPC, EAUTH, EISDIR, and so forth.

80656. ** CAPI3REF: Delete A Session Object ** ** Delete a session object previously allocated using ** [sqlite3session_create()]. Once a session object has been deleted, the ** results of attempting to use pSession with any other session module ** function are undefined. *** Session objects must be deleted before the database handle to which they ** are attached is closed. Refer to the documentation for ** [sqlite3session_create()] for details.

80657. ** 2001-09-15 ** ** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil. ** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

***** This header file defines the interface that the SQLite library ** presents to client programs. If a C-function, structure, datatype, ** or constant definition does not appear in this file, then it is ** not a published API of SQLite, is subject to change without ** notice, and should not be referenced by programs that use SQLite. *** Some of the definitions that are in this file are marked as ** "experimental". Experimental interfaces are normally new ** features recently added to SQLite. We do not anticipate changes ** to experimental interfaces but reserve the right to make minor changes ** if experience from use "in the wild" suggest such changes are prudent. *** The official C-language API documentation for SQLite is derived ** from comments in this file. This file is the authoritative source ** on how SQLite interfaces are supposed to operate. *** The name of this file under configuration management is "sqlite.h.in". ** The makefile makes some minor changes to this file (such as inserting ** the version number) and changes its name to "sqlite3.h" as ** part of the build process.

80658. Create a new auxiliary function

80659. ** CAPI3REF: Query Progress Callbacks ** METHOD: sqlite3 ** ** ^The sqlite3_progress_handler(D,N,X,P) interface causes the callback ** function X to be invoked periodically during long running calls to ** [sqlite3_exec()], [sqlite3_step()] and [sqlite3_get_table()] for ** database connection D. An example use for this ** interface is to keep a GUI updated during a large query. *** ^The parameter P is passed through as the only parameter to the ** callback function X. ^The parameter N is the approximate number of ** [virtual machine instructions] that are evaluated between successive ** invocations of the callback X. ^If N is less than one then the progress ** handler is disabled. ** ^Only a single progress handler may be defined at one time per ** [database connection]; setting a new progress handler cancels the ** old one. ^Setting parameter X to NULL disables the progress handler. ** ^The progress handler is also disabled by setting N to a value less ** than 1. *** ^If the progress callback returns non-zero, the operation is ** interrupted. This feature can be used to implement a ** "Cancel" button on a GUI progress dialog box. ** ** The progress handler callback must not do anything that will modify ** the database connection that invoked the progress handler. ** Note that [sqlite3_prepare_v2()] and [sqlite3_step()] both modify their ** database connections for the meaning of "modify" in this paragraph. **

80660. #define SQLITE_ABORT 4 // Also an error code

80661. lru page list

80662. ** CAPI3REF: Register A Callback To Handle SQLITE_BUSY Errors ** KEYWORDS: {busy-handler callback} {busy handler} ** METHOD: sqlite3 ** ** ^The sqlite3_busy_handler(D,X,P) routine sets a callback function X ** that might be invoked with argument P whenever ** an attempt is made to access a database table associated with ** [database connection] D when another thread ** or process has the table locked. ** The sqlite3_busy_handler() interface is used to implement ** [sqlite3_busy_timeout()] and [PRAGMA busy_timeout]. *** ^If the busy callback is NULL, then [SQLITE_BUSY] ** is returned immediately upon encountering the lock. ^If the busy callback ** is not NULL, then the callback might be invoked with two arguments. *** ^The first argument to the busy handler is a copy of the void* pointer which ** is the third argument to sqlite3_busy_handler(). ^The second argument to the busy handler callback is the number of times that the busy handler has ** been invoked previously for the same locking event. ^If the ** busy callback returns 0, then no additional attempts are made to ** access the database and [SQLITE_BUSY] is returned ** to the application. ** ^If the callback returns non-zero, then another attempt ** is made to access the database and the cycle repeats. ** ** The presence of a busy handler does not guarantee that it will be invoked ** when there is lock contention. ^If SQLite determines that invoking the busy ** handler could result in a deadlock, it will go ahead and return [SQLITE_BUSY] ** to the application instead of invoking the ** busy handler. ** Consider a scenario where one process is holding a read lock that ** it is trying to promote to a reserved lock and ** a second process is holding a reserved lock that it is trying ** to promote to an exclusive lock. The first process cannot proceed ** because it is blocked by the second and the second process cannot ** proceed because it is blocked by the first. If both processes ** invoke the busy handlers, neither will make any progress. Therefore, ** SQLite returns [SQLITE_BUSY] for the first process, hoping that this ** will induce the first process to release its read lock and allow ** the second process to proceed. *** ^The default busy callback is NULL. *** ^There can only be a single busy handler defined for each ** [database connection]. Setting a new busy handler clears any ** previously set handler. ^Note that calling [sqlite3_busy_timeout()] ** or evaluating [PRAGMA busy_timeout=N] will change the ** busy handler and thus clear any previously set busy handler. ** ** The busy callback should not take any actions which modify the ** database connection that invoked the busy handler. In other words, ** the busy handler is not reentrant. Any such actions ** result in undefined behavior. ** ** A busy handler must not close the database connection ** or [prepared statement] that invoked the busy handler.

80663. OUT: Inverse of input

80664. **comment:** ** CAPI3REF: Deprecated Functions ** DEPRECATED ** ** These functions are [deprecated]. In order to maintain ** backwards compatibility with older code, these functions continue ** to be supported. However, new applications should avoid ** the use of these functions. To encourage programmers to avoid ** these functions, we will not explain what they do.

label: code-design

80665. An open database

80666. ** CAPI3REF: Read Data From A BLOB Incrementally ** METHOD: sqlite3_blob ** ** ^This function is used to read data from an open [BLOB handle] into a ** caller-supplied buffer. N bytes of data are copied into buffer Z ** from the open BLOB, starting at offset iOffset.)** ** ^If offset iOffset is less than N bytes from the end of the BLOB, ** [SQLITE_ERROR] is returned and no data is read. ^If N or iOffset is ** less than zero, [SQLITE_ERROR] is returned and no data is read. ** ^The size of the blob (and hence the maximum value of N+iOffset) ** can be determined using the [sqlite3_blob_bytes()] interface. *** ^An attempt

to read from an expired [BLOB handle] fails with an ** error code of [SQLITE_ABORT]. *** ^On success, sqlite3_blob_read() returns SQLITE_OK. ** Otherwise, an [error code] or an [extended error code] is returned.)^ *** This routine only works on a [BLOB handle] which has been created ** by a prior successful call to [sqlite3_blob_open()] and which has not ** been closed by [sqlite3_blob_close()]. Passing any other pointer in ** to this routine results in undefined and probably undesirable behavior. *** See also: [sqlite3_blob_write()].

80667. **comment:** *** CAPI3REF: Mutex Types *** The [sqlite3_mutex_alloc()] interface takes a single argument ** which is one of these integer constants. *** The set of static mutexes may change from one SQLite release to the ** next. Applications that override the built-in mutex logic must be ** prepared to accommodate additional static mutexes.
- label:** code-design
80668. Visibility of parent node
80669. **comment:** Not used
- label:** code-design
80670. Column number
80671. Index Name Table Name
80672. Text to tokenize
80673. ** CAPI3REF: Virtual Table Instance Object ** KEYWORDS: sqlite3_vtab *** Every [virtual table module] implementation uses a subclass ** of this object to describe a particular instance ** of the [virtual table]. Each subclass will ** be tailored to the specific needs of the module implementation. ** The purpose of this superclass is to define certain fields that are ** common to all module implementations. *** ^Virtual tables methods can set an error message by assigning a ** string obtained from [sqlite3_mprintf()] to zErrMsg. The method should ** take care that any prior string is freed by a call to [sqlite3_free()] ** prior to assigning a new string to zErrMsg. ^After the error message ** is delivered up to the client application, the string will be automatically ** freed by sqlite3_free() and the zErrMsg field will be zeroed.
80674. ** CAPI3REF: Conflict resolution modes ** KEYWORDS: {conflict resolution mode} *** These constants are returned by [sqlite3_vtab_on_conflict()] to ** inform a [virtual table] implementation what the [ON CONFLICT] mode ** is for the SQL statement being evaluated. *** Note that the [SQLITE_IGNORE] constant is also used as a potential ** return value from the [sqlite3_set_authorizer()] callback and that ** [SQLITE_ABORT] is also a [result code].
80675. **comment:** ** CAPI3REF: Constants Defining Special Destructor Behavior ** These are special values for the destructor that is passed in as the ** final argument to routines like [sqlite3_result_blob()]. ^If the destructor ** argument is SQLITE_STATIC, it means that the content pointer is constant ** and will never change. It does not need to be destroyed. ^The ** SQLITE_TRANSIENT value means that the content will likely change in ** the near future and that SQLite should make its own private copy of ** the content before returning. *** The typedef is necessary to work around problems in certain ** C++ compilers.
- label:** code-design
80676. **comment:** ** CAPI3REF: Memory Allocator Statistics ** SQLite provides these two interfaces for reporting on the status ** of the [sqlite3_malloc()], [sqlite3_free()], and [sqlite3_realloc()] ** routines, which form the built-in memory allocation subsystem. *** ^The [sqlite3_memory_used()] routine returns the number of bytes ** of memory currently outstanding (malloced but not freed). ** ^The [sqlite3_memory_highwater()] routine returns the maximum ** value of [sqlite3_memory_used()] since the high-water mark ** was last reset. ^The values returned by [sqlite3_memory_used()] and ** [sqlite3_memory_highwater()] include any overhead ** added by SQLite in its implementation of [sqlite3_malloc()], ** but not overhead added by the any underlying system library ** routines that [sqlite3_malloc()] may call. *** ^The memory high-water mark is reset to the current value of ** [sqlite3_memory_used()] if and only if the parameter to ** [sqlite3_memory_highwater()] is true. ^The value returned ** by [sqlite3_memory_highwater(1)] is the high-water mark ** prior to the reset.
- label:** code-design
80677. Memory allocation function
80678. Maximum file pathname length
80679. sqlite3_randomness()
80680. OUT: True for an 'indirect' change
80681. **comment:** ** CAPI3REF: Obtain old.* Values From A Changeset Iterator ** The pIter argument passed to this function may either be an iterator ** passed to a conflict-handler by [sqlite3changeset_apply()], or an iterator ** created by [sqlite3changeset_start()]. In the latter case, the most recent ** call to [sqlite3changeset_next()] must have returned SQLITE_ROW. ** Furthermore, it may only be called if the type of change that the iterator ** currently points to is either [SQLITE_DELETE] or [SQLITE_UPDATE]. Otherwise, ** this function returns [SQLITE_MISUSE] and sets *ppValue to NULL. *** Argument iVal must be greater than or equal to 0, and less than the number ** of columns in the table affected by the current change. Otherwise, ** [SQLITE_RANGE] is returned and *ppValue is set to NULL. *** If successful, this function sets *ppValue to point to a protected ** sqlite3_value object containing the iVal'th value from the vector of ** original row values stored as part of the UPDATE or DELETE change and ** returns SQLITE_OK. The name of the function comes from the fact that this ** is similar to the "old." columns available to update or delete triggers. *** If some other error occurs (e.g. an OOM condition), an SQLite error code ** is returned and *ppValue is set to NULL.
- label:** code-design
80682. Virtual table of this cursor
80683. Operation NULL
80684. **comment:** ** CAPI3REF: Obtain The Current Operation From A Changeset Iterator ** The pIter argument passed to this function may either be an iterator ** passed to a conflict-handler by [sqlite3changeset_apply()], or an iterator ** created by [sqlite3changeset_start()]. In the latter case, the most recent ** call to [sqlite3changeset_next()] must have returned [SQLITE_ROW]. If this ** is not the case, this function returns [SQLITE_MISUSE]. *** If argument pzTab is not NULL, then *pzTab is set to point to a ** nul-terminated utf-8 encoded string containing the name of the table ** affected by the current change. The buffer remains valid until either ** sqlite3changeset_next() is called on the iterator or until the ** conflict-handler function returns. If pnCol is not NULL, then *pnCol is ** set to the number of columns in the table affected by the change. If ** pbIncorrect is not NULL, then *pbIndirect is set to true (1) if the change ** is an indirect change, or false (0) otherwise. See the documentation for ** [sqlite3session_indirect()] for a description of direct and indirect ** changes. Finally, if pOp is not NULL, then *pOp is set to one of ** [SQLITE_INSERT], [SQLITE_DELETE] or [SQLITE_UPDATE], depending on the ** type of change that the iterator currently points to. *** If no error occurs, SQLITE_OK is returned. If an error does occur, an ** SQLite error code is returned. The values of the output variables may not ** be trusted in this case.
- label:** code-design
80685. **comment:** ** CAPI3REF: Apply A Changeset To A Database ** Apply a changeset to a database. This function attempts to update the ** "main" database attached to handle db with the changes found in the ** changeset passed via the second and third arguments. *** The fourth argument (xFilter) passed to this function is the "filter ** callback". If it is not NULL, then for each table affected by at least one ** change in the changeset, the filter callback is invoked with ** the table name as the second argument, and a copy of the context pointer ** passed as the sixth argument to this function as the first. If the "filter ** callback" returns zero, then no attempt is made to apply any changes to ** the table. Otherwise, if the return value is non-zero or the xFilter ** argument to this function is NULL, all changes related to the table are ** attempted. *** For each table that is not excluded by the filter callback, this function ** tests that the target database contains a compatible table. A table is ** considered compatible if all of the following are true: *** ** The table has the same name as the name recorded in the ** changeset, and ** The table has at least as many columns as recorded in the ** changeset, and ** The table has primary key columns in the same position as ** recorded in the changeset. ** *** If there is no compatible table, it is not an error, but none of the ** changes associated with the table are applied. A warning message is issued ** via the sqlite3_log() mechanism with the error code SQLITE_SCHEMA. At most ** one such warning is issued for each table in the changeset. *** For each change for which there is a compatible table, an attempt is made ** to modify the table contents according to the UPDATE, INSERT or DELETE ** change. If a change cannot be applied cleanly, the conflict handler ** function passed as the fifth argument to sqlite3changeset_apply() may be ** invoked. A description of exactly when the conflict handler is invoked for ** each type of change is below. *** Unlike the xFilter argument, xConflict may not be passed NULL. The results ** of passing anything other than a valid function pointer as the xConflict ** argument are undefined. *** Each time the conflict handler function is invoked, it must return one ** of [SQLITE_CHANGESET OMIT], [SQLITE_CHANGESET_ABORT] or ** [SQLITE_CHANGESET_REPLACE]. SQLITE_CHANGESET_REPLACE may only be returned ** if the second argument passed to the conflict handler is either ** SQLITE_CHANGESET_DATA or SQLITE_CHANGESET_CONFLICT. If the conflict-handler ** returns an illegal value, any changes already made are rolled back and ** the call to sqlite3changeset_apply() returns SQLITE_MISUSE. Different ** actions are taken by sqlite3changeset_apply() depending on the value ** returned by each invocation of the conflict-handler function. Refer to ** the documentation for the three ** [SQLITE_CHANGESET OMIT]available return values] for details. *** <dl> ** <dt>DELETE Changes<dd> ** For each DELETE change, this function checks if the target database ** contains a row with the same primary key value (or values) as the ** original row values stored in the changeset. If it does, and the values ** stored in all non-primary key columns also match the values stored in ** the changeset the row is deleted from the target database. *** If a row with matching primary key values is found, but one or more of ** the non-primary key fields contains a value different from the original ** row value stored in the changeset, the conflict-handler function is ** invoked with [SQLITE_CHANGESET DATA] as the second argument. If the ** database table has more columns than are recorded in the changeset, ** only the values of those non-primary key fields are compared against ** the current database contents - any trailing database

table columns ** are ignored. *** If no row with matching primary key values is found in the database, ** the conflict-handler function is invoked with [SQLITE_CHANGESET_NOTFOUND] ** passed as the second argument. *** If the DELETE operation is attempted, but SQLite returns SQLITE_CONSTRAINT ** (which can only happen if a foreign key constraint is violated), the ** conflict-handler function is invoked with [SQLITE_CHANGESET_CONSTRAINT] ** passed as the second argument. This includes the case where the DELETE ** operation is attempted because an earlier call to the conflict handler ** function returned [SQLITE_CHANGESET_REPLACE]. *** <dt>INSERT Changes<dd> ** For each INSERT change, an attempt is made to insert the new row into ** the database. If the changeset row contains fewer fields than the ** database table, the trailing fields are populated with their default ** values. *** If the attempt to insert the row fails because the database already ** contains a row with the same primary key values, the conflict handler ** function is invoked with the second argument set to ** [SQLITE_CHANGESET_CONFLICT]. *** If the attempt to insert the row fails because of some other constraint ** violation (e.g. NOT NULL or UNIQUE), the conflict handler function is ** invoked with the second argument set to [SQLITE_CHANGESET_CONSTRAINT]. ** This includes the case where the INSERT operation is re-attempted because ** an earlier call to the conflict handler function returned ** [SQLITE_CHANGESET_REPLACE]. *** <dt>UPDATE Changes<dd> ** For each UPDATE change, this function checks if the target database ** contains a row with the same primary key value (or values) as the ** original row values stored in the changeset. If it does, and the values ** stored in all modified non-primary key columns also match the values ** stored in the changeset the row is updated within the target database. *** If a row with matching primary key values is found, but one or more of ** the modified non-primary key fields contains a value different from an ** original row value stored in the changeset, the conflict-handler function ** is invoked with [SQLITE_CHANGESET_DATA] as the second argument. Since ** UPDATE changes only contain values for non-primary key fields that are ** to be modified, only those fields need to match the original values to ** avoid the SQLITE_CHANGESET_DATA conflict-handler callback. *** If no row with matching primary key values is found in the database, ** the conflict-handler function is invoked with [SQLITE_CHANGESET_NOTFOUND] ** passed as the second argument. *** If the UPDATE operation is attempted, but SQLite returns ** SQLITE_CONSTRAINT, the conflict-handler function is invoked with ** [SQLITE_CHANGESET_CONSTRAINT] passed as the second argument. ** This includes the case where the UPDATE operation is attempted after ** an earlier call to the conflict handler function returned ** [SQLITE_CHANGESET_REPLACE]. ** </dt> ** It is safe to execute SQL statements, including those that write to the ** table that the callback related to, from within the xConflict callback. ** This can be used to further customize the applications conflict ** resolution strategy. *** All changes made by this function are enclosed in a savepoint transaction. ** If any other error (aside from a constraint failure when attempting to ** write to the target database) occurs, then the savepoint transaction is ** rolled back, restoring the target database to its original state, and an ** SQLite error code returned.

label: code-design

80686. **comment:** ** CAPI3REF: Formatted String Printing Functions *** These routines are work-alikes of the "printf()" family of functions ** from the standard C library. ** These routines understand most of the common K&R formatting options, ** plus some additional non-standard formats, detailed below. ** Note that some of the more obscure formatting options from recent ** C-library standards are omitted from this implementation. *** ^The sqlite3_mprintf() and sqlite3_vmpfint() routines write their ** results into memory obtained from [sqlite3_malloc0]. ** The strings returned by these two routines should be ** released by [sqlite3_free0]. ^Both routines return a ** NULL pointer if [sqlite3_malloc0] is unable to allocate enough ** memory to hold the resulting string. ** *** ^The sqlite3_snprintf() routine is similar to "snprintf()" from ** the standard C library. The result is written into the ** buffer supplied as the second parameter whose size is given by ** the first parameter. Note that the order of the ** first two parameters is reversed from snprintf().)^ This is an ** historical accident that cannot be fixed without breaking ** backwards compatibility. ^Note also that sqlite3_snprintf() ** returns a pointer to its buffer instead of the number of ** characters actually written into the buffer.)^ We admit that ** the number of characters written would be a more useful return ** value but we cannot change the implementation of sqlite3_snprintf() ** now without breaking compatibility. *** ^As long as the buffer size is greater than zero, sqlite3_snprintf() ** guarantees that the buffer is always zero-terminated. ^The first ** parameter "n" is the total size of the buffer, including space for ** the zero terminator. So the longest string that can be completely ** written will be n-1 characters. *** ^The sqlite3_vsprintf() routine is a varargs version of sqlite3_snprintf(). *** These routines all implement some additional formatting ** options that are useful for constructing SQL statements. ** All of the usual printf() formatting options apply. In addition, there ** is are "%q", "%Q", "%w" and "%z" options. *** ^The %q option works like %s in that it substitutes a nul-terminated ** string from the argument list. But %q also doubles every \" character. ** %q is designed for use inside a string literal.)^ By doubling each \" ** character it escapes that character and allows it to be inserted into ** the string. *** For example, assume the string variable zText contains text as follows: *** <blockquote><pre> ** char *zText = "It's a happy day!"; ** </pre></blockquote> *** One can use this text in an SQL statement as follows: *** <blockquote><pre> ** char *zSQL = sqlite3_mprintf("INSERT INTO table VALUES(%q)", zText); ** sqlite3_exec(db, zSQL, 0, 0, 0); ** sqlite3_free(zSQL); ** </pre></blockquote> *** Because the %q format string is used, the \" character in zText ** is escaped and the SQL generated is as follows: *** <blockquote><pre> ** INSERT INTO table1 VALUES('It's a happy day!') ** </pre></blockquote> *** This is correct. Had we used %s instead of %q, the generated SQL ** would have looked like this: ** *** <blockquote><pre> ** INSERT INTO table1 VALUES('It's a happy day!'); ** </pre></blockquote> *** This second example is an SQL syntax error. As a general rule you should ** always use %q instead of %s when inserting text into a string literal. *** ^The %Q option works like %q except it also adds single quotes around ** the outside of the total string. Additionally, if the parameter in the ** argument list is a NULL pointer, %Q substitutes the text "NULL" (without ** single quotes).)^ So, for example, one could say: *** <blockquote><pre> ** char *zSQL = sqlite3_mprintf("INSERT INTO table VALUES(%Q)", zText); ** sqlite3_exec(db, zSQL, 0, 0, 0); ** sqlite3_free(zSQL); ** </pre></blockquote> *** The code above will render a correct SQL statement in the zSQL ** variable even if the zText variable is a NULL pointer. *** ^The "%w" formatting option is like "%q" except that it expects to ** be contained within double-quotes instead of single quotes, and it ** escapes the double-quote character instead of the single-quote ** character.)^ The "%w" formatting option is intended for safely inserting ** table and column names into a constructed SQL statement. *** ^The "%z" formatting option works like "%s" but with the ** addition that after the string has been read and copied into ** the result, [sqlite3_free0] is called on the input string.)^

label: code-design

80687. ** CAPI3REF: Fundamental Datatypes ** KEYWORDS: SQLITE_TEXT *** ^Every value in SQLite has one of five fundamental datatypes: *** ** 64-bit signed integer ** 64-bit IEEE floating point number ** string ** BLOB ** NULL **)^ *** These constants are codes for each of those types. *** Note that the SQLITE_TEXT constant was also used in SQLite version 2 ** for a completely different meaning. Software that links against both ** SQLite version 2 and SQLite version 3 should use SQLITE3_TEXT, not ** SQLITE_TEXT.

80688. OUTPUT: Declared data type

80689. Free idxStr using sqlite3_free0() if true

80690. ** Provide the ability to override linkage features of the interface.

80691. OUT: SQLite db handle

80692. True if this constraint is usable

80693. ** CAPI3REF: Setting The Subtype Of An SQL Function ** METHOD: sqlite3_context *** The sqlite3_result_subtype(C,T) function causes the subtype of ** the result from the [application-defined SQL function] with ** [sqlite3_context] C to be the value T. Only the lower 8 bits ** of the subtype T are preserved in current versions of SQLite; ** higher order bits are discarded. ** The number of subtype bytes preserved by SQLite might increase ** in future releases of SQLite.

80694. SQLITE3_H

80695. Estimated number of rows returned

80696. **comment:** ** CAPI3REF: Destroy A Prepared Statement Object ** DESTRUCTOR: sqlite3_stmt *** ^The sqlite3_finalize() function is called to delete a [prepared statement]. ** ^If the most recent evaluation of the statement encountered no errors ** or if the statement is never been evaluated, then sqlite3_finalize() returns ** SQLITE_OK. ^If the most recent evaluation of statement S failed, then ** sqlite3_finalize(S) returns the appropriate [error code] or ** [extended error code]. *** ^The sqlite3_finalize(S) routine can be called at any point during ** the life cycle of [prepared statement] S: ** before statement S is ever evaluated, after ** one or more calls to [sqlite3_reset0], or after any call ** to [sqlite3_step0] regardless of whether or not the statement has ** completed execution. *** ^Invoking sqlite3_finalize() on a NULL pointer is a harmless no-op. *** The application must finalize every [prepared statement] in order to avoid ** resource leaks. It is a grievous error for the application to try to use ** a prepared statement after it has been finalized. Any use of a prepared ** statement after it has been finalized can result in undefined and ** undesirable behavior such as segfaults and heap corruption.

label: code-design

80697. Callback function to invoke

80698. **comment:** ** CAPI3REF: Standard File Control Opcodes ** KEYWORDS: {file control opcodes} {file control opcode} *** These integer constants are opcodes for the xFileControl method ** of the [sqlite3_io_methods] object and for the [sqlite3_file_control0] ** interface. *** ** [[SQLITE_FCNTL_LOCKSTATE]] ** The [SQLITE_FCNTL_LOCKSTATE] opcode is used for debugging. This ** opcode causes the xFileControl method to write the current state of ** the lock (one of [SQLITE_LOCK_NONE], [SQLITE_LOCK_SHARED], ** [SQLITE_LOCK_RESERVED], [SQLITE_LOCK_PENDING], or [SQLITE_LOCK_EXCLUSIVE]) ** into an integer that the pArg argument points to. This capability ** is used during testing and is only available when the SQLITE_TEST ** compile-time option is used. *** [[SQLITE_FCNTL_SIZE_HINT]] ** The [SQLITE_FCNTL_SIZE_HINT] opcode is used by SQLite to give the VFS ** layer a hint of how large the database file will grow to be during the ** current transaction. This hint is not guaranteed to be accurate but it ** is often close. The underlying VFS might choose to preallocate database ** file space based on this hint in order to help writes to the database ** file run faster. *** [[SQLITE_FCNTL_CHUNK_SIZE]] ** The [SQLITE_FCNTL_CHUNK_SIZE] opcode is

used to request that the VFS ** extends and truncates the database file in chunks of a size specified ** by the user. The fourth argument to [sqlite3_file_control()] should ** point to an integer (type int) containing the new chunk-size to use ** for the nominated database. Allocating database file space in large ** chunks (say 1MB at a time), may reduce file-system fragmentation and ** improve performance on some systems. *** [[SQLITE_FCNTL_FILE_POINTER]] ** The [SQLITE_FCNTL_FILE_POINTER] opcode is used to obtain a pointer ** to the [sqlite3_file] object associated with a particular database ** connection. See also [SQLITE_FCNTL_JOURNAL_POINTER]. *** [[SQLITE_FCNTL_JOURNAL_POINTER]] ** The [SQLITE_FCNTL_JOURNAL_POINTER] opcode is used to obtain a pointer ** to the [sqlite3_file] object associated with the journal file (either ** the [rollback journal] or the [write-ahead log]) for a particular database ** connection. See also [SQLITE_FCNTL_FILE_POINTER]. *** [[SQLITE_FCNTL_SYNC OMITTED]] ** No longer in use. *** [[SQLITE_FCNTL_SYNC]] ** The [SQLITE_FCNTL_SYNC] opcode is generated internally by SQLite and ** sent to the VFS immediately before the xSync method is invoked on a ** database file descriptor. Or, if the xSync method is not invoked ** because the user has configured SQLite with ** [PRAGMA synchronous | PRAGMA synchronous=OFF] it is invoked in place ** of the xSync method. In most cases, the pointer argument passed with ** this file-control is NULL. However, if the database file is being synced ** as part of a multi-database commit, the argument points to a nul-terminated ** string containing the transactions master-journal file name. VFSes that ** do not need this signal should silently ignore this opcode. Applications ** should not call [sqlite3_file_control()] with this opcode as doing so may ** disrupt the operation of the specialized VFSes that do require it. *** [[SQLITE_FCNTL_COMMIT_PHASETWO]] ** The [SQLITE_FCNTL_COMMIT_PHASETWO] opcode is generated internally by SQLite ** and sent to the VFS after a transaction has been committed immediately ** but before the database is unlocked. VFSes that do not need this signal ** should silently ignore this opcode. Applications should not call ** [sqlite3_file_control()] with this opcode as doing so may disrupt the ** operation of the specialized VFSes that do require it. *** [[SQLITE_FCNTL_WIN32_AV_RETRY]] ** ^The [SQLITE_FCNTL_WIN32_AV_RETRY] opcode is used to configure automatic ** retry counts and intervals for certain disk I/O operations for the ** windows [VFS] in order to provide robustness in the presence of ** anti-virus programs. By default, the windows VFS will retry file read, ** file write, and file delete operations up to 10 times, with a delay ** of 25 milliseconds before the first retry and with the delay increasing ** by an additional 25 milliseconds with each subsequent retry. This ** opcode allows these two values (10 retries and 25 milliseconds of delay) ** to be adjusted. The values are changed for all database connections ** within the same process. The argument is a pointer to an array of two ** integers where the first integer is the new retry count and the second ** integer is the delay. If either integer is negative, then the setting ** is not changed but instead the prior value of that setting is written ** into the array entry, allowing the current retry settings to be ** interrogated. The zDbName parameter is ignored. *** [[SQLITE_FCNTL_PERSIST_WAL]] ** ^The [SQLITE_FCNTL_PERSIST_WAL] opcode is used to set or query the ** persistent [WAL | Write Ahead Log] setting. By default, the auxiliary ** write ahead log and shared memory files used for transaction control ** are automatically deleted when the latest connection to the database ** closes. Setting persistent WAL mode causes those files to persist after ** close. Persisting the files is useful when other processes that do not ** have write permission on the directory containing the database file want ** to read the database file, as the WAL and shared memory files must exist ** in order for the database to be readable. The fourth parameter to ** [sqlite3_file_control()] for this opcode should be a pointer to an integer. ** That integer is 0 to disable persistent WAL mode or 1 to enable persistent ** WAL mode. If the integer is -1, then it is overwritten with the current ** WAL persistence setting. *** [[SQLITE_FCNTL_POWERSAFE_OVERWRITE]] ** ^The [SQLITE_FCNTL_POWERSAFE_OVERWRITE] opcode is used to set or query the ** persistent "powersafe-overwrite" or "PSOW" setting. The PSOW setting ** determines the [SQLITE_IOCAP_POWERSAFE_OVERWRITE] bit of the ** xDeviceCharacteristics methods. The fourth parameter to ** [sqlite3_file_control()] for this opcode should be a pointer to an integer. ** That integer is 0 to disable zero-damage mode or 1 to enable zero-damage ** mode. If the integer is -1, then it is overwritten with the current ** zero-damage mode setting. *** [[SQLITE_FCNTL_OVERWRITE]] ** ^The [SQLITE_FCNTL_OVERWRITE] opcode is invoked by SQLite after opening ** a write transaction to indicate that, unless it is rolled back for some ** reason, the entire database file will be overwritten by the current ** transaction. This is used by VACUUM operations. *** [[SQLITE_FCNTL_VFSNAME]] ** ^The [SQLITE_FCNTL_VFSNAME] opcode can be used to obtain the names of ** all [VFSes] in the VFS stack. The names are of all VFS shims and the ** final bottom-level VFS are written into memory obtained from ** [sqlite3_malloc()] and the result is stored in the char* variable ** that the fourth parameter of [sqlite3_file_control()] points to. ** The caller is responsible for freeing the memory when done. As with ** all file-control actions, there is no guarantee that this will actually ** do anything. Callers should initialize the char* variable to a NULL ** pointer in case this file-control is not implemented. This file-control ** is intended for diagnostic use only. *** [[SQLITE_FCNTL_VFS_POINTER]] ** ^The [SQLITE_FCNTL_VFS_POINTER] opcode finds a pointer to the top-level ** [VFSes] currently in use. ^The argument X in ** sqlite3_file_control(db,SQLITE_FCNTL_VFS_POINTER,X) must be ** of type "[sqlite3_vfs]" **. This opcodes will set *X ** to a pointer to the top-level VFS. ** ^When there are multiple VFS shims in the stack, this opcode finds the ** upper-most shim only. *** [[SQLITE_FCNTL_PRAGMA]] ** ^Whenever a [PRAGMA] statement is parsed, an [SQLITE_FCNTL_PRAGMA] ** file control is sent to the open [sqlite3_file] object corresponding ** to the database file to which the pragma statement refers. ^The argument ** to the [SQLITE_FCNTL_PRAGMA] file control is an array of ** pointers to strings (char**) in which the second element of the array ** is the name of the pragma and the third element is the argument to the ** pragma or NULL if the pragma has no argument. ^The handler for an ** [SQLITE_FCNTL_PRAGMA] file control can optionally make the first element ** of the char** argument point to a string obtained from [sqlite3_mprintf()] ** or the equivalent and that string will become the result of the pragma or ** the error message if the pragma fails. ^If the ** [SQLITE_FCNTL_PRAGMA] file control returns [SQLITE_NOTFOUND], then normal ** [PRAGMA] processing continues. ^If the [SQLITE_FCNTL_PRAGMA] ** file control returns [SQLITE_OK], then the parser assumes that the ** VFS has handled the PRAGMA itself and the parser generates a no-op ** prepared statement if result string is NULL, or that returns a copy ** of the result string if the string is non-NULL. ** ^If the [SQLITE_FCNTL_PRAGMA] file control returns ** any result code other than [SQLITE_OK] or [SQLITE_NOTFOUND], that means ** that the VFS encountered an error while handling the [PRAGMA] and the ** compilation of the PRAGMA fails with an error. ^The [SQLITE_FCNTL_PRAGMA] ** file control occurs at the beginning of pragma statement analysis and so ** it is able to override built-in [PRAGMA] statements. *** [[SQLITE_FCNTL_BUSYHANDLER]] ** ^The [SQLITE_FCNTL_BUSYHANDLER] ** file-control may be invoked by SQLite on the database file handle ** shortly after it is opened in order to provide a custom VFS with access ** to the connections busy-handler callback. The argument is of type (void **) ** - an array of two (void *) values. The first (void *) actually points ** to a function of type (int (*)(void *)). In order to invoke the connections ** busy-handler, this function should be invoked with the second (void *) in ** the array as the only argument. If it returns non-zero, then the operation ** should be retried. If it returns zero, the custom VFS should abandon the ** current operation. *** [[SQLITE_FCNTL_TEMPFILENAME]] ** ^Application can invoke [SQLITE_FCNTL_TEMPFILENAME] file-control ** to have SQLite generate a ** temporary filename using the same algorithm that is followed to generate ** temporary filenames for TEMP tables and other internal uses. The ** argument should be a char** which will be filled with the filename ** written into memory obtained from [sqlite3_malloc()]. The caller should ** invoke [sqlite3_free()] on the result to avoid a memory leak. *** [[SQLITE_FCNTL_MMAP_SIZE]] ** The [SQLITE_FCNTL_MMAP_SIZE] file control is used to query or set the ** maximum number of bytes that will be used for memory-mapped I/O. ** The argument is a pointer to a value of type sqlite3_int64 that ** is an advisory maximum number of bytes in the file to memory map. The ** pointer is overwritten with the old value. The limit is not changed if ** the value originally pointed to is negative, and so the current limit ** can be queried by passing in a pointer to a negative number. This ** file-control is used internally to implement [PRAGMA mmap_size]. *** [[SQLITE_FCNTL_TRACE]] ** ^The [SQLITE_FCNTL_TRACE] file control provides advisory information ** to the VFS about what the higher layers of the SQLite stack are doing. ** This file control is used by some VFS activity tracing [shims]. ** The argument is a zero-terminated string. Higher layers in the ** SQLite stack may generate instances of this file control if ** the [SQLITE_USE_FCNTL_TRACE] compile-time option is enabled. *** [[SQLITE_FCNTL_HAS_MOVED]] ** ^The [SQLITE_FCNTL_HAS_MOVED] file control interprets its argument as a ** pointer to an integer and it writes a boolean into that integer depending ** on whether or not the file has been renamed, moved, or deleted since it ** was first opened. *** [[SQLITE_FCNTL_WIN32_GET_HANDLE]] ** ^The [SQLITE_FCNTL_WIN32_GET_HANDLE] opcode can be used to obtain the ** underlying native file handle associated with a file handle. This file ** control interprets its argument as a pointer to a native file handle and ** writes the resulting value there. *** [[SQLITE_FCNTL_WIN32_SET_HANDLE]] ** ^The [SQLITE_FCNTL_WIN32_SET_HANDLE] opcode is used for debugging. This ** opcode causes the xFileControl method to swap the file handle with the one ** pointed to by the pArg argument. This capability is used during testing ** and only needs to be supported when SQLITE_TEST is defined. *** [[SQLITE_FCNTL_WAL_BLOCK]] ** ^The [SQLITE_FCNTL_WAL_BLOCK] is a signal to the VFS layer that it might ** be advantageous to block on the next WAL lock if the lock is not immediately ** available. The WAL subsystem issues this signal during rare ** circumstances in order to fix a problem with priority inversion. ** Applications should not use this file-control. *** [[SQLITE_FCNTL_ZIPVFS]] ** ^The [SQLITE_FCNTL_ZIPVFS] opcode is implemented by zipvfs only. All other ** VFS should return SQLITE_NOTFOUND for this opcode. *** [[SQLITE_FCNTL_RBU]] ** ^The [SQLITE_FCNTL_RBU] opcode is implemented by the special VFS used by ** the RBU extension only. All other VFS should return SQLITE_NOTFOUND for ** this opcode. **

label: code-design

80699. Methods for an open file

80700. **comment:** ** This is the obsolete pcache_methods object that has now been replaced ** by sqlite3_pcache_methods2. This object is not used by SQLite. It is ** retained in the header file for backwards compatibility only.

label: code-design

80701. String, possibly obtained from sqlite3_malloc

80702. ***** CUSTOM AUXILIARY FUNCTIONS *** Virtual table implementations may overload SQL functions by implementing ** the sqlite3_module.xFindFunction() method.

80703. Fields below are only available in SQLite 3.10.0 and later

80704. Name of the shared library containing extension

80705. Destination database name

80706. Array of trailing arguments

80707. Callback

80708. ** CAPI3REF: Maximum xShmLock index *** The xShmLock method on [sqlite3_io_methods] may use values ** between 0 and this upper bound as its "offset" argument. ** The SQLite core will never attempt to acquire or release a ** lock outside of this range

80709. **comment:** ** CAPI3REF: Count The Number Of Rows Modified ** METHOD: sqlite3 *** ^This function returns the number of rows modified, inserted or ** deleted by the most recently completed INSERT, UPDATE or DELETE ** statement on the database connection specified by the only parameter. ** ^Executing any other type of SQL statement does not modify the value ** returned by this function. ** ** ^Only changes made directly by the INSERT, UPDATE or DELETE statement are ** considered - auxiliary changes caused by [CREATE TRIGGER | triggers], ** [foreign key actions] or [REPLACE] constraint resolution are not counted. ** ** Changes to a view that are intercepted by ** [INSTEAD OF trigger | INSTEAD OF triggers] are not counted. ^The value ** returned by sqlite3_changes() immediately after an INSERT, UPDATE or ** DELETE statement run on a view is always zero. Only changes made to real ** tables are counted. ** ** Things are more complicated if the sqlite3_changes() function is ** executed while a trigger program is running. This may happen if the ** program uses the [changes()] SQL function, or if some other callback ** function invokes sqlite3_changes() directly. Essentially: ** ** ** ^Before entering a trigger program the value returned by ** sqlite3_changes() function is saved. After the trigger program ** has finished, the original value is restored.)^ ** ** ^Within a trigger program each INSERT, UPDATE and DELETE ** statement sets the value returned by sqlite3_changes() ** upon completion as normal. Of course, this value will not include ** any changes performed by sub-triggers, as the sqlite3_changes() ** value will be saved and restored after each sub-trigger has run.)^ ** ** ** ^This means that if the changes() SQL function (or similar) is used ** by the first INSERT, UPDATE or DELETE statement within a trigger, it ** returns the value as set when the calling statement began executing. ** ^If it is used by the second or subsequent such statement within a trigger ** program, the value returned reflects the number of rows modified by the ** previous INSERT, UPDATE or DELETE statement within the same trigger. ** ** See also the [sqlite3_total_changes()] interface, the ** [count_changes pragma], and the [changes() SQL function]. ** ** If a separate thread makes changes on the same database connection ** while [sqlite3_changes()] is running then the value returned ** is unpredictable and not meaningful.

label: code-design

80710. Pragma Name 1st arg or NULL

80711. Module destructor function

80712. **comment:** ** The interface to the virtual-table mechanism is currently considered ** to be experimental. The interface might change in incompatible ways. ** If this is a problem for you, do not use the interface at this time. ** ** When the virtual-table mechanism stabilizes, we will declare the ** interface fixed, support it indefinitely, and remove this comment.

label: code-design

80713. Client data for xCreate/xConnect

80714. ** CAPI3REF: Write Data Into A BLOB Incrementally ** METHOD: sqlite3_blob *** ^This function is used to write data into an open [BLOB handle] from a ** caller-supplied buffer. N bytes of data are copied from the buffer Z ** into the open BLOB, starting at offset iOffset.)^ ** ** ^On success, sqlite3_blob_write() returns SQLITE_OK. ** Otherwise, an [error code] or an [extended error code] is returned.)^ ** ^Unless SQLITE_MISUSE is returned, this function sets the ** [database connection] error code and message accessible via ** [sqlite3_errcode()] and [sqlite3_errmsg()] and related functions. ** ** ^If the [BLOB handle] passed as the first argument was not opened for ** writing (the flags parameter to [sqlite3_blob_open()] was zero), ** this function returns [SQLITE_READONLY]. ** ** This function may only modify the contents of the BLOB; it is ** not possible to increase the size of a BLOB using this API. ** ^If offset iOffset is less than N bytes from the end of the BLOB, ** [SQLITE_ERROR] is returned and no data is written. The size of the ** BLOB (and hence the maximum value of N+iOffset) can be determined ** using the [sqlite3_blob_bytes()] interface. ^If N or iOffset are less ** than zero [SQLITE_ERROR] is returned and no data is written. ** ** ^An attempt to write to an expired [BLOB handle] fails with an ** error code of [SQLITE_ABORT]. ^Writes to the BLOB that occurred ** before the [BLOB handle] expired are not rolled back by the ** expiration of the handle, though of course those changes might ** have been overwritten by the statement that expired the BLOB handle ** or by other independent statements. ** ** This routine only works on a [BLOB handle] which has been created ** by a prior successful call to [sqlite3_blob_open()] and which has not ** been closed by [sqlite3_blob_close()]. Passing any other pointer in ** to this routine results in undefined and probably undesirable behavior. ** ** See also: [sqlite3_blob_read()].

80715. ifndef _SQLITE3RTREE_H_

80716. NULL Function Name

80717. ** CAPI3REF: Determine if a database is read-only ** METHOD: sqlite3 *** ^The sqlite3_db_readonly(D,N) interface returns 1 if the database N ** of connection D is read-only, 0 if it is read/write, or -1 if N is not ** the name of a database on connection D.

80718. #define SQLITE_IGNORE 2 // Also used by sqlite3_authorizer() callback

80719. ** CAPI3REF: SQLite Runtime Status *** ^These interfaces are used to retrieve runtime status information ** about the performance of SQLite, and optionally to reset various ** highwater marks. ^The first argument is an integer code for ** the specific parameter to measure. ^(Recognized integer codes ** are of the form [status parameters | SQLITE_STATUS_...].)^ ** ^The current value of the parameter is returned into *pCurrent. ** ^The highest recorded value is returned in *pHighwater. ^If the ** resetFlag is true, then the highest record value is reset after ** *pHighwater is written. ^Some parameters do not record the highest ** value. For those parameters ** nothing is written into *pHighwater and the resetFlag is ignored.)^ ** ^Other parameters record only the highwater mark and not the current ** value. For these latter parameters nothing is written into *pCurrent.)** ** ^The sqlite3_status() and sqlite3_status64() routines return ** SQLITE_OK on success and a non-zero [error code] on failure. ** ** If either the current value or the highwater mark is too large to ** be represented by a 32-bit integer, then the values returned by ** sqlite3_status() are undefined. ** ** See also: [sqlite3_db_status()]

80720. Level of current node or entry

80721. **comment:** ** CAPI3REF: Record A Database Snapshot ** EXPERIMENTAL *** ^The [sqlite3_snapshot_get(D,S,P)] interface attempts to make a ** new [sqlite3_snapshot] object that records the current state of ** schema S in database connection D. ^On success, the ** [sqlite3_snapshot_get(D,S,P)] interface writes a pointer to the newly ** created [sqlite3_snapshot] object into *P and returns SQLITE_OK. ** If there is not already a read-transaction open on schema S when ** this function is called, one is opened automatically. ** ** The following must be true for this function to succeed. If any of ** the following statements are false when sqlite3_snapshot_get() is ** called, SQLITE_ERROR is returned. The final value of *P is undefined ** in this case. ** ** ** The database handle must be in [autocommit mode]. ** ** Schema S of [database connection] D must be a [WAL mode] database. ** ** There must not be a write transaction open on schema S of database ** connection D. ** ** One or more transactions must have been written to the current wal ** file since it was created on disk (by any connection). This means ** that a snapshot cannot be taken on a wal mode database with no wal ** file immediately after it is first opened. At least one transaction ** must be written to it first. ** ** ** This function may also return SQLITE_NOMEM. If it is called with the ** database handle in autocommit mode but fails for some other reason, ** whether or not a read transaction is opened on schema S is undefined. ** ** The [sqlite3_snapshot] object returned from a successful call to ** [sqlite3_snapshot_get()] must be freed using [sqlite3_snapshot_free()] ** to avoid a memory leak. ** ** The [sqlite3_snapshot_get()] interface is only available when the ** SQLITE_ENABLE_SNAPSHOT compile-time option is used.

label: code-design

80722. ** CAPI3REF: Commit And Rollback Notification Callbacks ** METHOD: sqlite3 *** ^The sqlite3_commit_hook() interface registers a callback ** function to be invoked whenever a transaction is [COMMIT | committed]. ** ^Any callback set by a previous call to sqlite3_commit_hook() ** for the same database connection is overridden. ** ^The sqlite3_rollback_hook() interface registers a callback ** function to be invoked whenever a transaction is [ROLLBACK | rolled back]. ** ^Any callback set by a previous call to sqlite3_rollback_hook() ** for the same database connection is overridden. ** ^The pArg argument is passed through to the callback. ** ^If the callback on a commit hook function returns non-zero, ** then the commit is converted into a rollback. ** ** ^The sqlite3_commit_hook(D,C,P) and sqlite3_rollback_hook(D,C,P) functions ** return the P argument from the previous call of the same function ** on the same [database connection] D, or NULL for ** the first call for each function on D. ** ** The commit and rollback hook callbacks are not reentrant. ** The callback implementation must not do anything that will modify ** the database connection that invoked the callback. Any actions ** to modify the database connection must be deferred until after the ** completion of the [sqlite3_step()] call that triggered the commit ** or rollback hook in the first place. ** Note that running any other SQL statements, including SELECT statements, ** or merely calling [sqlite3_prepare_v2()] and [sqlite3_step()] will modify ** the database connections for the meaning of "modify" in this paragraph. ** ** ^Registering a NULL function disables the callback. ** ** ^When the commit hook callback routine returns zero, the [COMMIT] ** operation is allowed to continue normally. ^If the commit hook ** returns non-zero, then the [COMMIT] is converted into a [ROLLBACK]. ** ^The rollback hook is invoked on a rollback that results from a commit ** hook returning non-zero, just as it would be with any other rollback. ** ** ^For the purposes of this API, a transaction is said to have been ** rolled back if an explicit "ROLLBACK" statement is executed, or ** an error or

constraint causes an implicit rollback to occur. ** ^The rollback callback is not invoked if a transaction is ** automatically rolled back because the database connection is closed. ** See also the [sqlite3_update_hook()] interface.

80723. **comment:** ** CAPI3REF: Dynamically Typed Value Object ** KEYWORDS: {protected sqlite3_value} {unprotected sqlite3_value} ** ** SQLite uses the sqlite3_value object to represent all values ** that can be stored in a database table. SQLite uses dynamic typing ** for the values it stores. ^Values stored in sqlite3_value objects ** can be integers, floating point values, strings, BLOBS, or NULL. ** An sqlite3_value object may be either "protected" or "unprotected". ** Some interfaces require a protected sqlite3_value. Other interfaces ** will accept either a protected or an unprotected sqlite3_value. If SQLite is compiled to be single-threaded ** (with [SQLITE_THREADS=0] and with [sqlite3_threadsafe()] returning 0) ** or if SQLite is run in one of reduced mutex modes ** [SQLITE_CONFIG_SINGLETHREAD] or [SQLITE_CONFIG_MULTITHREAD] ** then there is no distinction between protected and unprotected ** sqlite3_value objects and they can be used interchangeably. However, ** for maximum code portability it is recommended that applications ** still make the distinction between protected and unprotected ** sqlite3_value objects even when not strictly required. ** ^The sqlite3_value objects that are passed as parameters into the ** implementation of [application-defined SQL functions] are protected. ** ^The sqlite3_value object returned by ** [sqlite3_column_value()] is unprotected. ** Unprotected sqlite3_value objects may only be used with ** [sqlite3_result_value()] and [sqlite3_bind_value()]. ** The [sqlite3_value_blob | sqlite3_value_type()] family of ** interfaces require protected sqlite3_value objects.

label: code-design

80724. ** CAPI3REF: Overload A Function For A Virtual Table ** METHOD: sqlite3 *** ^Virtual tables can provide alternative implementations of functions ** using the [xFindFunction] method of the [virtual table module]. ** But global versions of those functions ** must exist in order to be overloaded.)** ** ^This API makes sure a global version of a function with a particular ** name and number of parameters exists. If no such function exists ** before this API is called, a new function is created.)** ^The implementation ** of the new function always causes an exception to be thrown. So ** the new function is not good for anything by itself. Its only ** purpose is to be a placeholder function that can be overloaded ** by a [virtual table].

80725. OUTPUT: True if NOT NULL constraint exists

80726. Unable to open the database file

80727. ** Specify the activation key for a SEE database. Unless ** activated, none of the SEE routines will work.

80728. ** Allowed values for sqlite3_rtree_query.eWithin and .eParentWithin.

80729. **comment:** ** CAPI3REF: Streaming Versions of API functions. *** The six streaming API xxx_strm() functions serve similar purposes to the ** corresponding non-streaming API functions: *** <table border=1 style="margin-left:8ex;margin-right:8ex"> ** <tr><th>Streaming function</th>Non-streaming equivalent</th> ** <tr><td>sqlite3changeset_apply<td>[sqlite3changeset_apply] ** <tr><td>sqlite3changeset_concat<td>[sqlite3changeset_concat] ** <tr><td>sqlite3changeset_invert<td>[sqlite3changeset_invert] ** <tr><td>sqlite3changeset_start<td>[sqlite3changeset_start] ** <tr><td>sqlite3session_changeset<td>[sqlite3session_changeset] ** <tr><td>sqlite3session_patchset<td>[sqlite3session_patchset] ** </table> *** Non-streaming functions that accept changesets (or patchsets) as input ** require that the entire changeset be stored in a single buffer in memory. ** Similarly, those that return a changeset or patchset do so by returning ** a pointer to a single large buffer allocated using sqlite3_malloc(). ** Normally this is convenient. However, if an application running in a ** low-memory environment is required to handle very large changesets, the ** large contiguous memory allocations required can become onerous. ** In order to avoid this problem, instead of a single large buffer, input ** is passed to a streaming API functions by way of a callback function that ** the sessions module invokes to incrementally request input data as it is ** required. In all cases, a pair of API function parameters such as *** <pre> ** &nbs; int nChangeset, ** &nbs; void *pChangeset, ** </pre> ** Is replaced by: *** <pre> ** &nbs; int (*xInput)(void *pIn, void *pData, int *pnData), ** &nbs; void *pIn, ** </pre> ** Each time the xInput callback is invoked by the sessions module, the first ** argument passed is a copy of the supplied pIn context pointer. The second ** argument, pData, points to a buffer (*pnData) bytes in size. Assuming no ** error occurs the xInput method should copy up to (*pnData) bytes of data ** into the buffer and set (*pnData) to the actual number of bytes copied ** before returning SQLITE_OK. If the input is completely exhausted, (*pnData) ** should be set to zero to indicate this. Or, if an error occurs, an SQLite ** error code should be returned. In all cases, if an xInput callback returns ** an error, all processing is abandoned and the streaming API function ** returns a copy of the error code to the caller. *** In the case of sqlite3changeset_start_strm(), the xInput callback may be ** invoked by the sessions module at any point during the lifetime of the ** iterator. If such an xInput callback returns an error, the iterator enters ** an error state, whereby all subsequent calls to iterator functions ** immediately fail with the same error code as returned by xInput. ** Similarly, streaming API functions that return changesets (or patchsets) ** return them in chunks by way of a callback function instead of via a ** pointer to a single large buffer. In this case, a pair of parameters such ** as: *** <pre> ** &nbs; int *pnChangeset, ** &nbs; void **ppChangeset, ** </pre> ** Is replaced by: *** <pre> ** &nbs; int (*xOutput)(void *pOut, const void *pData, int nData), ** &nbs; void *pOut ** </pre> ** The xOutput callback is invoked zero or more times to return data to ** the application. The first parameter passed to each call is a copy of the ** pOut pointer supplied by the application. The second parameter, pData, ** points to a buffer nData bytes in size containing the chunk of output ** data being returned. If the xOutput callback successfully processes the ** supplied data, it should return SQLITE_OK to indicate success. Otherwise, ** it should return some other SQLite error code. In this case processing ** is immediately abandoned and the streaming API function returns a copy ** of the xOutput error code to the application. ** The sessions module never invokes an xOutput callback with the third ** parameter set to a value less than or equal to zero. Other than this, ** no guarantees are made as to the size of the chunks of data returned.

label: code-design

80730. end-of-error-codes

80731. Virtual table implementations will typically add additional fields

80732. Mask of FTSS5_TOKENIZE_* flags

80733. **comment:** ** CAPI3REF: Mutex Handle ** ** The mutex module within SQLite defines [sqlite3_mutex] to be an ** abstract type for a mutex object. The SQLite core never looks ** at the internal representation of an [sqlite3_mutex]. It only ** deals with pointers to the [sqlite3_mutex] object. ** ** Mutexes are created using [sqlite3_mutex_alloc()].

label: code-design

80734. OUT: Array of boolean - true for PK cols

80735. ** CAPI3REF: Checkpoint Mode Values ** KEYWORDS: {checkpoint mode} ** ** These constants define all valid values for the "checkpoint mode" passed ** as the third parameter to the [sqlite3_wal_checkpoint_v2()] interface. ** See the [sqlite3_wal_checkpoint_v2()] documentation for details on the ** meaning of each of these checkpoint modes.

80736. ** CAPI3REF: Extended Result Codes ** KEYWORDS: {extended result code definitions} ** ** In its default configuration, SQLite API routines return one of 30 integer ** [result codes]. However, experience has shown that many of ** these result codes are too coarse-grained. They do not provide as ** much information about problems as programmers might like. In an effort to ** address this, newer versions of SQLite (version 3.3.8 [dateof:3.3.8] ** and later) include ** support for additional result codes that provide more detailed information ** about errors. These [extended result codes] are enabled or disabled ** on a per database connection basis using the ** [sqlite3_extended_result_codes()] API. Or, the extended code for ** the most recent error can be obtained using ** [sqlite3_extended_errcode()].

80737. Methods for the module

80738. OUT: Number of entries in output array

80739. OUT: Visibility

80740. Column constrained. -1 for ROWID

80741. Rowid of row about to be deleted/updated

80742. ** CAPI3REF: Prepared Statement Object ** KEYWORDS: {prepared statement} {prepared statements} ** ** An instance of this object represents a single SQL statement that ** has been compiled into binary form and is ready to be evaluated. ** ** Think of each SQL statement as a separate computer program. The ** original SQL text is source code. A prepared statement object ** is the compiled object code. All SQL must be converted into a ** prepared statement before it can be run. ** ** The life-cycle of a prepared statement object usually goes like this: *** ** Create the prepared statement object using [sqlite3_prepare_v2()]. ** Bind values to [parameters] using the sqlite3_bind_*() ** interfaces. ** Run the SQL by calling [sqlite3_step()] one or more times. ** Reset the prepared statement using [sqlite3_reset()] then go back ** to step 2. Do this zero or more times. ** Destroy the object using [sqlite3_finalize()]. **

80743. Initialize the memory allocator

80744. **comment:** ** CAPI3REF: Convenience Routines For Running Queries ** METHOD: sqlite3 *** This is a legacy interface that is preserved for backwards compatibility. ** Use of this interface is not recommended. ** ** Definition: A result table is memory data structure created by the ** [sqlite3_get_table()] interface. A result table records the ** complete query results from one or more queries. ** ** The table conceptually has a number of rows and columns. But ** these numbers are not part of the result table itself. These ** numbers are obtained separately. Let N be the number of rows ** and M be the

number of columns. *** A result table is an array of pointers to zero-terminated UTF-8 strings. ** There are (N+1)*M elements in the array. The first M pointers point ** to zero-terminated strings that contain the names of the columns. ** The remaining entries all point to query results. NULL values result ** in NULL pointers. All other values are in their UTF-8 zero-terminated ** string representation as returned by [sqlite3_column_text()]. *** A result table might consist of one or more memory allocations. ** It is not safe to pass a result table directly to [sqlite3_free()]. ** A result table should be deallocated using [sqlite3_free_table()]. *** ^As an example of the result table format, suppose a query result ** is as follows: *** <blockquote><pre> ** Name | Age ** -----
----- ** Alice | 43 ** Bob | 28 ** Cindy | 21 ** </pre></blockquote> *** There are two column (M==2) and three rows (N==3). Thus the ** result table has 8 entries. Suppose the result table is stored ** in an array names azResult. Then azResult holds this content: *** <blockquote><pre> ** azResult[0] = "Name"; ** azResult[1] = "Age"; ** azResult[2] = "Alice"; ** azResult[3] = "43"; ** azResult[4] = "Bob"; ** azResult[5] = "28"; ** azResult[6] = "Cindy"; ** azResult[7] = "21"; ** </pre></blockquote> *** ^The sqlite3_get_table() function evaluates one or more ** semicolon-separated SQL statements in the zero-terminated UTF-8 ** string of its 2nd parameter and returns a result table to the ** pointer given in its 3rd parameter. *** After the application has finished with the result from sqlite3_get_table(), ** it must pass the result table pointer to sqlite3_free_table() in order to ** release the memory that was malloced. Because of the way the ** [sqlite3_malloc()] happens within sqlite3_get_table(), the calling ** function must not try to call [sqlite3_free()] directly. Only ** [sqlite3_free_table()] is able to release the memory properly and safely. *** ^The sqlite3_get_table() interface is implemented as a wrapper around ** [sqlite3_exec()]. The sqlite3_get_table() routine does not have access ** to any internal data structures of SQLite. It uses only the public ** interface defined here. As a consequence, errors that occur in the ** wrapper layer outside of the internal [sqlite3_exec()] call are not ** reflected in subsequent calls to [sqlite3_errcode()] or ** [sqlite3_errmsg()].

label: code-design

80745. ** CAPI3REF: Source Of Data In A Query Result ** METHOD: sqlite3_stmt ** ** ^These routines provide a means to determine the database, table, and ** table column that is the origin of a particular result column in ** [SELECT] statement. ** ^The name of the database or table or column can be returned as ** either a UTF-8 or UTF-16 string. ^The _database_ routines return ** the database name, the _table_ routines return the table name, and ** the origin_ routines return the column name. ** ^The returned string is valid until the [prepared statement] is destroyed ** using [sqlite3_finalize()] or until the statement is automatically ** reprepared by the first call to [sqlite3_step()] for a particular run ** or until the same information is requested ** again in a different encoding. ** ** ^The names returned are the original un-aliased names of the ** database, table, and column. *** ^The first argument to these interfaces is a [prepared statement]. ** ^These functions return information about the Nth result column returned by ** the statement, where N is the second function argument. ** ^The left-most column is column 0 for these routines. *** ^If the Nth column returned by the statement is an expression or ** subquery and is not a column value, then all of these functions return ** NULL. ^These routine might also return NULL if a memory allocation error ** occurs. ^Otherwise, they return the name of the attached database, table, ** or column that query result column was extracted from. *** ^As with all other SQLite APIs, those whose names end with "16" return ** UTF-16 encoded strings and the other functions return UTF-8. *** ^These APIs are only available if the library was compiled with the ** [SQLITE_ENABLE_COLUMN_METADATA] C-preprocessor symbol. *** If two or more threads call one or more of these routines against the same ** prepared statement and column at the same time then the results are ** undefined. *** If two or more threads call one or more ** [sqlite3_column_database_name | column metadata interfaces] ** for the same [prepared statement] and result column ** at the same time then the results are undefined.

80746. ** The methods above are in versions 1 and 2 of the sqlite_vfs object. ** Those below are for version 3 and greater.

80747. ** CAPI3REF: Configure an auto-checkpoint ** METHOD: sqlite3 ** ** ^The [sqlite3_wal_autocheckpoint(D,N)] is a wrapper around ** [sqlite3_wal_hook()] that causes any database on [database connection] D ** to automatically [checkpoint] ** after committing a transaction if there are N or ** more frames in the [write-ahead log] file. ^Passing zero or ** a negative value as the nFrame parameter disables automatic ** checkpoints entirely. *** ^The callback registered by this function replaces any existing callback ** registered using [sqlite3_wal_hook()]. ^Likewise, registering a callback ** using [sqlite3_wal_hook()] disables the automatic checkpoint mechanism ** configured by this function. *** ^The [wal_autocheckpoint pragma] can be used to invoke this interface ** from SQL. ** ** ^Checkpoints initiated by this mechanism are ** [sqlite3_wal_checkpoint_v2|PASSIVE]. *** ^Every new [database connection] defaults to having the auto-checkpoint ** enabled with a threshold of 1000 or [SQLITE_DEFAULT_WAL_AUTOCHECKPOINT] ** pages. The use of this interface ** is only necessary if the default setting is found to be suboptimal ** for a particular application.

80748. Number of open cursors

80749. **comment:** ** The type for a callback function. ** This is legacy and deprecated. It is included for historical ** compatibility and is not documented.

label: documentation

80750. Original SQL values of parameters

80751. **comment:** ** CAPI3REF: Loadable Extension Thunk ** ** A pointer to the opaque sqlite3_api_routines structure is passed as ** the third parameter to entry points of [loadable extensions]. This ** structure must be typedefed in order to work around compiler warnings ** on some platforms.

label: code-design

80752. The module for this virtual table

80753. callback can use this, if desired

80754. File opened that is not a database file

80755. **comment:** Warnings from sqlite3_log()

label: code-design

80756. ** CAPI3REF: Prepared Statement Scan Status Opcodes ** KEYWORDS: {scanstatus options} ** ** The following constants can be used for the T parameter to the ** [sqlite3_stmt_scanstatus(S,X,T,V)] interface. Each constant designates a ** different metric for sqlite3_stmt_scanstatus() to return. ** ** When the value returned to V is a string, space to hold that string is ** managed by the prepared statement S and will be automatically freed when ** S is finalized. *** <dl> ** [[SQLITE_SCANSTAT_NLOOP]] <dt>SQLITE_SCANSTAT_NLOOP</dt> ** <dd>^The [sqlite3_int64] variable pointed to by the T parameter will be ** set to the total number of times that the X-th loop has run.</dd> *** [[SQLITE_SCANSTAT_NVISIT]] <dt>SQLITE_SCANSTAT_NVISIT</dt> ** <dd>^The [sqlite3_int64] variable pointed to by the T parameter will be set ** to the total number of rows examined by all iterations of the X-th loop.</dd> *** <dd>[[SQLITE_SCANSTAT_EST]] <dt>SQLITE_SCANSTAT_EST</dt> ** <dd>^The "double" variable pointed to by the T parameter will be set to the ** query planner's estimate for the average number of rows output from each ** iteration of the X-th loop. If the query planner's estimates was accurate, ** then this value will approximate the quotient NVISIT/NLOOP and the ** product of this value for all prior loops with the same SELECTID will ** be the NLOOP value for the current loop. *** [[SQLITE_SCANSTAT_NAME]] <dt>SQLITE_SCANSTAT_NAME</dt> ** <dd>^The "const char **" variable pointed to by the T parameter will be set ** to a zero-terminated UTF-8 string containing the name of the index or table ** used for the X-th loop. *** [[SQLITE_SCANSTAT_EXPLAIN]] <dt>SQLITE_SCANSTAT_EXPLAIN</dt> ** <dd>^The "const char **" variable pointed to by the T parameter will be set ** to a zero-terminated UTF-8 string containing the [EXPLAIN QUERY PLAN] ** description for the X-th loop. *** [[SQLITE_SCANSTAT_SELECTID]] <dt>SQLITE_SCANSTAT_SELECTID</dt> ** <dd>^The "int" variable pointed to by the T parameter will be set to the ** "select-id" for the X-th loop. The select-id identifies which query or ** subquery the loop is part of. The main query has a select-id of zero. ** The select-id is the same value as is output in the first column ** of an [EXPLAIN QUERY PLAN] query. ** </dl>

80757. Notifications from sqlite3_log()

80758. ** CAPI3REF: Constants Returned By The Conflict Handler ** ** A conflict handler callback must return one of the following three values. ** <dl> ** <dt>SQLITE_CHANGESET OMIT<dd> ** If a conflict handler returns this value no special action is taken. The ** change that caused the conflict is not applied. The session module ** continues to the next change in the changeset. *** <dt>SQLITE_CHANGESET_REPLACE<dd> ** This value may only be returned if the second argument to the conflict ** handler was SQLITE_CHANGESET_DATA or SQLITE_CHANGESET_CONFLICT. If this ** is not the case, any changes applied so far are rolled back and the ** call to sqlite3changeset_apply() returns SQLITE_MISUSE. *** ^If CHANGESET_REPLACE is returned by an SQLITE_CHANGESET_DATA conflict ** handler, then the conflicting row is either updated or deleted, depending ** on the type of change. *** If CHANGESET_REPLACE is returned by an SQLITE_CHANGESET_CONFLICT conflict ** handler, then the conflicting row is removed from the database and a ** second attempt to apply the change is made. If this second attempt fails, ** the original row is restored to the database before continuing. *** <dt>SQLITE_CHANGESET_ABORT<dd> ** If this value is returned, any changes applied so far are rolled back ** and the call to sqlite3changeset_apply() returns SQLITE_ABORT. ** </dl>

80759. 1st argument to callback

80760. ** CAPI3REF: Status Parameters for database connections ** KEYWORDS: {SQLITE_DBSTATUS options} ** ** These constants are the available integer "verbs" that can be passed as ** the second argument to the [sqlite3_db_status()] interface. *** New verbs may be added in future releases of SQLite. Existing verbs ** might be discontinued. Applications should check the return code from ** [sqlite3_db_status()] to make sure that the call worked. ** The [sqlite3_db_status()] interface will return a non-zero error code ** if a discontinued or unsupported verb is invoked. *** <dl> ** [[SQLITE_DBSTATUS_LOOKASIDE_USED]] ^<dt>SQLITE_DBSTATUS_LOOKASIDE_USED</dt> ** <dd>This parameter returns the number of lookaside memory slots currently ** checked out.</dd> *** [[SQLITE_DBSTATUS_LOOKASIDE_HIT]] ^<dt>SQLITE_DBSTATUS_LOOKASIDE_HIT</dt> ** <dd>This parameter returns the number malloc attempts that were ** satisfied using lookaside memory.

Only the high-water value is meaningful; ** the current value is always zero.)^ *** [[SQLITE_DBSTATUS_LOOKASIDE_MISS_SIZE]] ** ^<dt>SQLITE_DBSTATUS_LOOKASIDE_MISS_SIZE</dt> ** <dd>This parameter returns the number malloc attempts that might have ** been satisfied using lookaside memory but failed due to the amount of ** memory requested being larger than the lookaside slot size. ** Only the high-water value is meaningful; ** the current value is always zero.)^ *** [[SQLITE_DBSTATUS_LOOKASIDE_MISS_FULL]] ** ^<dt>SQLITE_DBSTATUS_LOOKASIDE_MISS_FULL</dt> ** <dd>This parameter returns the number malloc attempts that might have ** been satisfied using lookaside memory but failed due to all lookaside ** memory already being in use. ** Only the high-water value is meaningful; ** the current value is always zero.)^ *** [[SQLITE_DBSTATUS_CACHE_USED]] ^<dt>SQLITE_DBSTATUS_CACHE_USED</dt> ** <dd>This parameter returns the approximate number of bytes of heap ** memory used by all pager caches associated with the database connection.)^ *** ^The highwater mark associated with SQLITE_DBSTATUS_CACHE_USED is always 0. *** [[SQLITE_DBSTATUS_CACHE_USED_SHARED]] ** ^<dt>SQLITE_DBSTATUS_CACHE_USED_SHARED</dt> ** <dd>This parameter is similar to DBSTATUS_CACHE_USED, except that if a ** pager cache is shared between two or more connections the bytes of heap ** memory used by that pager cache is divided evenly between the attached ** connections.)^ In other words, if none of the pager caches associated ** with the database connection are shared, this request returns the same ** value as DBSTATUS_CACHE_USED. Or, if one or more of the pager caches are ** shared, the value returned by this call will be smaller than that returned ** by DBSTATUS_CACHE_USED. ^The highwater mark associated with ** SQLITE_DBSTATUS_CACHE_USED_SHARED is always 0. *** [[SQLITE_DBSTATUS_SCHEMA_USED]] ^<dt>SQLITE_DBSTATUS_SCHEMA_USED</dt> ** <dd>This parameter returns the approximate number of bytes of heap ** memory used to store the schema for all databases associated ** with the connection - main, temp, and any [ATTACH]-ed databases.)^ *** ^The full amount of memory used by the schemas is reported, even if the ** schema memory is shared with other database connections due to ** [shared cache mode] being enabled. ** ^The highwater mark associated with SQLITE_DBSTATUS_SCHEMA_USED is always 0. *** [[SQLITE_DBSTATUS_STMT_USED]] ^<dt>SQLITE_DBSTATUS_STMT_USED</dt> ** <dd>This parameter returns the approximate number of bytes of heap ** and lookaside memory used by all prepared statements associated with ** the database connection.)^ *** ^The highwater mark associated with SQLITE_DBSTATUS_STMT_USED is always 0. ** </dd> *** [[SQLITE_DBSTATUS_CACHE_HIT]] ^<dt>SQLITE_DBSTATUS_CACHE_HIT</dt> ** <dd>This parameter returns the number of pager cache hits that have ** occurred.)^ ^The highwater mark associated with SQLITE_DBSTATUS_CACHE_HIT ** is always 0. ** </dd> *** [[SQLITE_DBSTATUS_CACHE_MISS]] ^<dt>SQLITE_DBSTATUS_CACHE_MISS</dt> ** <dd>This parameter returns the number of pager cache misses that have ** occurred.)^ ^The highwater mark associated with SQLITE_DBSTATUS_CACHE_MISS ** is always 0. ** </dd> *** [[SQLITE_DBSTATUS_CACHE_WRITE]] ^<dt>SQLITE_DBSTATUS_CACHE_WRITE</dt> ** <dd>This parameter returns the number of dirty cache entries that have ** been written to disk. Specifically, the number of pages written to the ** wal file in wal mode databases, or the number of pages written to the ** database file in rollback mode databases. Any pages written as part of ** transaction rollback or database recovery operations are not included. ** If an IO or other error occurs while writing a page to disk, the effect ** on subsequent SQLITE_DBSTATUS_CACHE_WRITE requests is undefined.)^ ^The ** highwater mark associated with SQLITE_DBSTATUS_CACHE_WRITE is always 0. ** </dd> *** [[SQLITE_DBSTATUS_DEFERRED_FKS]] ^<dt>SQLITE_DBSTATUS_DEFERRED_FKS</dt> ** <dd>This parameter returns zero for the current value if and only if ** all foreign key constraints (deferred or immediate) have been ** resolved.)^ ^The highwater mark is always 0. ** </dd> ** </dl>

80761. Name of db (e.g. "main")

80762. ** CAPI3REF: Set A Busy Timeout ** METHOD: sqlite3 *** ^This routine sets a [sqlite3_busy_handler | busy handler] that sleeps ** for a specified amount of time when a table is locked. ^The handler ** will sleep multiple times until at least "ms" milliseconds of sleeping ** have accumulated. ^After at least "ms" milliseconds of sleeping, ** the handler returns 0 which causes [sqlite3_step()] to return ** [SQLITE_BUSY]. *** ^Calling this routine with an argument less than or equal to zero ** turns off all busy handlers. *** ^There can only be a single busy handler for a particular ** [database connection] at any given moment. If another busy handler ** was defined (using [sqlite3_busy_handler()]) prior to calling ** this routine, that other busy handler is cleared.)^ *** See also: [PRAGMA busy_timeout]

80763. **comment:** ** CAPI3REF: Compiling An SQL Statement ** KEYWORDS: {SQL statement compiler} ** METHOD: sqlite3 *** CONSTRUCTOR: sqlite3_stmt *** To execute an SQL statement, it must first be compiled into a byte-code ** program using one of these routines. Or, in other words, these routines ** are constructors for the [prepared statement] object. *** ^The preferred routine to use is [sqlite3_prepare_v2()]. The ** [sqlite3_prepare()] interface is legacy and should be avoided. ** [sqlite3_prepare_v3()] has an extra "prepFlags" option that is used ** for special purposes. *** ^The use of the UTF-8 interfaces is preferred, as SQLite currently ** does all parsing using UTF-8. The UTF-16 interfaces are provided ** as a convenience. The UTF-16 interfaces work by converting the ** input text into UTF-8, then invoking the corresponding UTF-8 interface. *** ^The first argument, "db", is a [database connection] obtained from a ** prior successful call to [sqlite3_open()], [sqlite3_open_v2()] or ** [sqlite3_open16()]. The database connection must not have been closed. *** ^The second argument, "zSql", is the statement to be compiled, encoded ** as either UTF-8 or UTF-16. The sqlite3_prepare(), sqlite3_prepare_v2(), ** and sqlite3_prepare_v3() ** interfaces use UTF-8, and sqlite3_prepare16(), sqlite3_prepare16_v2(), ** and sqlite3_prepare16_v3() use UTF-16. *** ^If the nByte argument is negative, then zSql is read up to the ** first zero terminator. ^If nByte is positive, then it is the ** number of bytes read from zSql. ^If nByte is zero, then no prepared ** statement is generated. ** If the caller knows that the supplied string is nul-terminated, then ** there is a small performance advantage to passing an nByte parameter that ** is the number of bytes in the input string <i>including</i> ** the nul-terminator. *** ^If pzTail is not NULL then *pzTail is made to point to the first byte ** past the end of the first SQL statement in zSql. These routines only ** compile the first statement in zSql, so *pzTail is left pointing to ** what remains uncompiled. *** ^ppStmt is left pointing to a compiled [prepared statement] that can be ** executed using [sqlite3_step()]. ^If there is an error, *ppStmt is set ** to NULL. ^If the input text contains no SQL (if the input is an empty ** string or a comment) then *ppStmt is set to NULL. ** The calling procedure is responsible for deleting the compiled ** SQL statement using [sqlite3_finalize()] after it has finished with it. ** ppStmt may not be NULL. *** ^On success, the sqlite3_prepare() family of routines return [SQLITE_OK]; ** otherwise an [error code] is returned. *** ^The sqlite3_prepare_v2(), sqlite3_prepare_v3(), sqlite3_prepare16_v2(), ** and sqlite3_prepare16_v3() interfaces are recommended for all new programs. ** The older interfaces (sqlite3_prepare() and sqlite3_prepare16()) ** are retained for backwards compatibility, but their use is discouraged. *** ^In the "vX" interfaces, the prepared statement ** that is returned (the [sqlite3_stmt] object) contains a copy of the ** original SQL text. This causes the [sqlite3_step()] interface to ** behave differently in three ways: *** ** ** ** ^If the database schema changes, instead of returning [SQLITE_SCHEMA] as it ** always used to do, [sqlite3_step()] will automatically recompile the SQL ** statement and try to run it again. As many as [SQLITE_MAX_SCHEMA_RETRY] ** retries will occur before sqlite3_step() gives up and returns an error. ** ** ** ^When an error occurs, [sqlite3_step()] will return one of the detailed ** [error codes] or [extended error codes]. ^The legacy behavior was that ** [sqlite3_step()] would only return a generic [SQLITE_ERROR] result code ** and the application would have to make a second call to [sqlite3_reset()] ** in order to find the underlying cause of the problem. With the "v2" prepare ** interfaces, the underlying reason for the error is returned immediately. ** ** ** ^If the specific value bound to [parameter] host parameter] in the ** WHERE clause might influence the choice of query plan for a statement, ** then the statement will be automatically recompiled, as if there had been ** a schema change, on the first [sqlite3_step()] call following any change ** to the [sqlite3_bind_text | bindings] of that [parameter]. ** ^The specific value of WHERE-clause [parameter] might influence the ** choice of query plan if the parameter is the left-hand side of a [LIKE] ** or [GLOB] operator or if the parameter is compared to an indexed column ** and the [SQLITE_ENABLE_STAT3] compile-time option is enabled. ** ** <p>^A sqlite3_prepare_v3() differs from sqlite3_prepare_v2() only in having ** the extra prepFlags parameter, which is a bit array consisting of zero or ** more of the [SQLITE_PERSISTENT|SQLITE_PREPARE_*] flags. ^The ** sqlite3_prepare_v2() interface works exactly the same as ** sqlite3_prepare_v3() with a zero prepFlags parameter. **

label: code-design

80764. Index of loop to report on

80765. ** CAPI3REF: Attach A Table To A Session Object ** ** If argument zTab is not NULL, then it is the name of a table to attach ** to the session object passed as the first argument. All subsequent changes ** made to the table while the session object is enabled will be recorded. See ** documentation for [sqlite3session_changeset()] for further details. *** ^Or, if argument zTab is NULL, then changes are recorded for all tables ** in the database. If additional tables are added to the database (by ** executing "CREATE TABLE" statements) after this call is made, changes for ** the new tables are also recorded. *** ^Changes can only be recorded for tables that have a PRIMARY KEY explicitly ** defined as part of their CREATE TABLE statement. It does not matter if the ** PRIMARY KEY is an "INTEGER PRIMARY KEY" (rowid alias) or not. The PRIMARY ** KEY may consist of a single column, or may be a composite key. ** ** It is not an error if the named table does not exist in the database. Nor ** is it an error if the named table does not have a PRIMARY KEY. However, ** no changes will be recorded in either of these scenarios. *** ^Changes are not recorded for individual rows that have NULL values stored ** in one or more of their PRIMARY KEY columns. *** ^SQLITE_OK is returned if the call completes without error. Or, if an error ** occurs, an SQLite error code (e.g. SQLITE_NOMEM) is returned.

80766. **comment:** ** CAPI3REF: Binding Values To Prepared Statements ** KEYWORDS: {host parameter} {host parameters} {host parameter name} **

KEYWORDS: {SQL parameter} {SQL parameters} {parameter binding} ** METHOD: sqlite3_stmt *** ^In the SQL statement text input to [sqlite3_prepare_v2()] and its variants, ** literals may be replaced by a [parameter] that matches one of following ** templates: *** ** ? ** ? NNN ** :VVV ** @VVV ** \$VVV ** ** ^In the templates above, NNN represents an integer literal, ** and VVV represents an alphanumeric identifier.)^ ^The values of these ** parameters (also called "host parameter names" or "SQL parameters") ** can be set using the sqlite3_bind_*() routines defined here. *** ^The first argument to the sqlite3_bind_*() routines is always ** a pointer to the [sqlite3_stmt] object returned from **

[sqlite3_prepare_v2()] or its variants. ** ** ^The second argument is the index of the SQL parameter to be set. ** ^The leftmost SQL parameter has an index of 1. ^When the same named ** SQL parameter is used more than once, second and subsequent ** occurrences have the same index as the first occurrence. ** ^The index for named parameters can be looked up using the ** [sqlite3_bind_parameter_index()] API if desired. ^The index ** for "?NNN" parameters is the value of NNN. ** ^The NNN value must be between 1 and the [sqlite3_limit()] ** parameter [SQLITE_LIMIT_VARIABLE_NUMBER] (default value: 999). ** ** ^The third argument is the value to bind to the parameter. ** ^If the third parameter to sqlite3_bind_text() or sqlite3_bind_text16() ** or sqlite3_bind_blob() is a NULL pointer then the fourth parameter ** is ignored and the end result is the same as sqlite3_bind_null(). ** ** ^In those routines that have a fourth argument, its value is the ** number of bytes in the parameter. To be clear: the value is the ** number of <u>bytes</u> in the value, not the number of characters.)^ ** ^If the fourth parameter to sqlite3_bind_text() or sqlite3_bind_text16() ** is negative, then the length of the string is ** the number of bytes up to the first zero terminator. ** If the fourth parameter to sqlite3_bind_blob() is negative, then ** the behavior is undefined. ** If a non-negative fourth parameter is provided to sqlite3_bind_text() ** or sqlite3_bind_text16() or sqlite3_bind_text64() then ** that parameter must be the byte offset ** where the NUL terminator would occur assuming the string were NUL ** terminated. If any NUL characters occur at byte offsets less than ** the value of the fourth parameter then the resulting string value will ** contain embedded NULs. The result of expressions involving strings ** with embedded NULs is undefined. ** ** ^The fifth argument to the BLOB and string binding interfaces ** is a destructor used to dispose of the BLOB or ** string after SQLite has finished with it. ^The destructor is called ** to dispose of the BLOB or string even if the call to bind API fails. ** ^If the fifth argument is ** the special value [SQLITE_STATIC], then SQLite assumes that the ** information is in static, unmanaged space and does not need to be freed. ** ^If the fifth argument has the value [SQLITE_TRANSIENT], then ** SQLite makes its own private copy of the data immediately, before ** the sqlite3_bind_*() routine returns. ** ** ^The sixth argument to sqlite3_bind_text64() must be one of ** [SQLITE_UTF8], [SQLITE_UTF16], [SQLITE_UTF16BE], or [SQLITE_UTF16LE] ** to specify the encoding of the text in the third parameter. If ** the sixth argument to sqlite3_bind_text64() is not one of the ** allowed values shown above, or if the text encoding is different ** from the encoding specified by the sixth parameter, then the behavior ** is undefined. ** ** ^The sqlite3_bind_zeroblob() routine binds a BLOB of length N that ** is filled with zeroes. ^A zeroblob uses a fixed amount of memory ** (just an integer to hold its size) while it is being processed. ** Zeroblobs are intended to serve as placeholders for BLOBs whose ** content is later written using ** [sqlite3_blob_open | incremental BLOB I/O] routines. ** ^A negative value for the zeroblob results in a zero-length BLOB. ** ** ^The sqlite3_bind_pointer(S,I,P,T,D) routine causes the I-th parameter in ** [prepared statement] S to have an SQL value of NULL, but to also be ** associated with the pointer P of type T. ^D is either a NULL pointer or ** a pointer to a destructor function for P. ^SQLite will invoke the ** destructor D with a single argument of P when it is finished using ** P. The T parameter should be a static string, preferably a string ** literal. The sqlite3_bind_pointer() routine is part of the ** [pointer passing interface] added for SQLite 3.20.0. ** ** ^If any of the sqlite3_bind_*() routines are called with a NULL pointer ** for the [prepared statement] or with a prepared statement for which ** [sqlite3_step()] has been called more recently than [sqlite3_reset()], ** then the call will return [SQLITE_MISUSE]. If any sqlite3_bind_*() ** routine is passed a [prepared statement] that has been finalized, the ** result is undefined and probably harmful. ** ** ^Bindings are not cleared by the [sqlite3_reset()] routine. ** ^Unbound parameters are interpreted as NULL. ** ** ^The sqlite3_bind_* routines return [SQLITE_OK] on success or an ** [error code] if anything goes wrong. ** ^[SQLITE_TOOBIG] might be returned if the size of a string or BLOB ** exceeds limits imposed by [sqlite3_limit()][SQLITE_LIMIT_LENGTH] or ** [SQLITE_MAX_LENGTH]. ** ^[SQLITE_RANGE] is returned if the parameter ** index is out of range. ^[SQLITE_NOMEM] is returned if malloc() fails. ** ** See also: [sqlite3_bind_parameter_count()], ** [sqlite3_bind_parameter_name()], and [sqlite3_bind_parameter_index()].

label: code-design

80767. pContext from when function registered

80768. OUT: Buffer containing changeset

80769. **comment:** ** Undo the hack that converts floating point types to integer for ** builds on processors without floating point support.

label: code-design

80770. The largest iLevel value in the tree

80771. ** Register a 2nd-generation geometry callback named zScore that can be ** used as part of an R-Tree geometry query as follows: ** ** SELECT ... FROM <rtree> WHERE <rtree col> MATCH \$zQueryFunc(... params ...)

80772. IMP: R-51971-34154

80773. **comment:** ** CAPI3REF: Configuration Options ** KEYWORDS: {configuration option} ** ** These constants are the available integer configuration options that ** can be passed as the first argument to the [sqlite3_config()] interface. ** ** New configuration options may be added in future releases of SQLite. ** Existing configuration options might be discontinued. Applications ** should check the return code from [sqlite3_config()] to make sure that ** the call worked. The [sqlite3_config()] interface will return a ** non-zero [error code] if a discontinued or unsupported configuration option ** is invoked. ** ** <dl> **

[[SQLITE_CONFIG_SINGLETHREAD]] <dt>SQLITE_CONFIG_SINGLETHREAD</dt> ** <dd>There are no arguments to this option. ^This option sets the ** [threading mode] to Single-thread. In other words, it disables ** all mutexing and puts SQLite into a mode where it can only be used ** by a single thread. ^If SQLite is compiled with ** the [SQLITE_THREADSAFE | SQLITE_THREADS=0] compile-time option then ** it is not possible to change the [threading mode] from its default ** value of Single-thread and so [sqlite3_config()] will return ** [SQLITE_ERROR] if called with the

SQLITE_CONFIG_SINGLETHREAD ** configuration option.</dd> ** [[SQLITE_CONFIG_MULTITHREAD]]

<dt>SQLITE_CONFIG_MULTITHREAD</dt> ** <dd>There are no arguments to this option. ^This option sets the ** [threading mode] to Multi-thread. In other words, it disables ** mutexing on [database connection] and [prepared statement] objects. ** The application is responsible for serializing access to ** [database connections] and [prepared statements]. But other mutexes ** are enabled so that SQLite will be safe to use in a multi-threaded ** environment as long as no two threads attempt to use the same ** [database connection] at the same time. ^If SQLite is compiled with ** the [SQLITE_THREADSAFE |

SQLITE_THREADS=0] compile-time option then ** it is not possible to set the Multi-thread [threading mode] and ** [sqlite3_config()] will return [SQLITE_ERROR] if called with the ** SQLITE_CONFIG_MULTITHREAD configuration option.</dd> ** [[SQLITE_CONFIG_SERIALIZED]]

<dt>SQLITE_CONFIG_SERIALIZED</dt> ** <dd>There are no arguments to this option. ^This option sets the ** [threading mode] to Serialized. In other words, this option enables ** all mutexes including the recursive ** mutexes on [database connection] and [prepared statement] objects. ** In this mode (which is the default when SQLite is compiled with ** [SQLITE_THREADS=1]) the SQLite library will itself serialize access ** to [database connections] and [prepared statements] so that the ** application is free to use the same [database connection] or the ** same [prepared statement] in different threads at the same time. ** ^If SQLite is compiled with ** the [SQLITE_THREADSAFE | SQLITE_THREADS=0] compile-time option then ** it is not possible to set the Serialized [threading mode] and ** [sqlite3_config()] will return [SQLITE_ERROR] if called with the ** SQLITE_CONFIG_SERIALIZED configuration option.

</dd> ** [[SQLITE_CONFIG_MALLOC]] <dt>SQLITE_CONFIG_MALLOC</dt> ** <dd> ^The SQLITE_CONFIG_MALLOC option takes a single argument which is ** a pointer to an instance of the [sqlite3_mem_methods] structure. ** The argument specifies ** alternative low-level memory allocation routines to be used in place of ** the memory allocation routines built into SQLite.^> ^SQLite makes ** its own private copy of the content of the

[sqlite3_mem_methods] structure ** before the [sqlite3_config()] call returns.</dd> ** [[SQLITE_CONFIG_GETMALLOC]]

<dt>SQLITE_CONFIG_GETMALLOC</dt> ** <dd> ^The SQLITE_CONFIG_GETMALLOC option takes a single argument which ** is a pointer to an instance of the [sqlite3_mem_methods] structure. ** The [sqlite3_mem_methods] ** structure is filled with the currently defined memory allocation routines.</dd> ** ^This option can be used to overload the default memory allocation ** routines with a wrapper that simulates memory allocation failure or ** tracks memory usage, for example. </dd> ** [[SQLITE_CONFIG_MEMSTATUS]] <dt>SQLITE_CONFIG_MEMSTATUS</dt> ** <dd> ^The

SQLITE_CONFIG_MEMSTATUS option takes single argument of type int, ** interpreted as a boolean, which enables or disables the collection of ** memory allocation statistics. ^When memory allocation statistics are ** disabled, the following SQLite interfaces become non-operational: ** **

[sqlite3_memory_used()] ** [sqlite3_memory_highwater()] ** [sqlite3_soft_heap_limit64()] ** [sqlite3_status64()] ** ^ ** ^Memory allocation statistics are enabled by default unless SQLite is ** compiled with [SQLITE_DEFAULT_MEMSTATUS]=0 in which case memory ** allocation statistics are disabled by default. ** </dd> ** [[SQLITE_CONFIG_SCRATCH]] <dt>SQLITE_CONFIG_SCRATCH</dt> ** <dd> ^The

SQLITE_CONFIG_SCRATCH option specifies a static memory buffer ** that SQLite can use for scratch memory. ^There are three arguments ** to SQLITE_CONFIG_SCRATCH: A pointer an 8-byte ** aligned memory buffer from which the scratch allocations will be ** drawn, the size of each scratch allocation (sz), ** and the maximum number of scratch allocations (N).^ ** The first argument must be a pointer to an 8-byte aligned buffer ** of at least sz*N bytes of memory. ** ^SQLite will not use more than one scratch buffers per thread. ** ^SQLite will never request a scratch buffer that is more than 6 ** times the database page size. ** ^If SQLite needs additional ** scratch memory beyond what is provided by this configuration option, then ** [sqlite3_malloc()] will be used to obtain the memory needed.<p> ** ^When the application provides any amount of scratch memory using ** SQLITE_CONFIG_SCRATCH, SQLite avoids unnecessary large ** [sqlite3_malloc|heap allocations]. ** This can help [Robson proof|prevent memory allocation failures] due to heap ** fragmentation in low-memory embedded systems. ** </dd> ** [[SQLITE_CONFIG_PAGECACHE]] <dt>SQLITE_CONFIG_PAGECACHE</dt> ** <dd> ^The

SQLITE_CONFIG_PAGECACHE option specifies a memory pool ** that SQLite can use for the database page cache with the default page ** cache implementation. ** This configuration option is a no-op if an application-defined page ** cache implementation is loaded using the [SQLITE_CONFIG_PCAFE2]. ** ^There are three arguments to SQLITE_CONFIG_PAGECACHE: A pointer to ** 8-byte aligned memory (pMem), the size of each page cache line (sz), ** and the number of cache lines (N). ** The sz argument should be the size of the largest database page ** (a power of two between 512 and 65536) plus some extra bytes for each ** page header. ^The number of extra bytes needed by the page header ** can be determined using

[SQLITE_CONFIG_PCACHE_HDRSZ]. ** ^It is harmless, apart from the wasted memory, ** for the sz parameter to be larger than necessary. The pMem ** argument must be either a NULL pointer or a pointer to an 8-byte ** aligned block of memory of at least sz*N bytes, otherwise ** subsequent behavior is undefined. ** ^When pMem is not NULL, SQLite will strive to use the memory provided ** to satisfy page cache needs, falling back to [sqlite3_malloc()] if ** a page cache line is larger than sz bytes or if all of the pMem buffer ** is exhausted. ** ^If pMem is NULL and N is non-zero, then each database connection ** does an initial bulk allocation for page cache memory ** from [sqlite3_malloc()] sufficient for N cache lines if N is positive or ** of -1024*N bytes if N is negative. . ^If additional ** page cache memory is needed beyond what is provided by the initial ** allocation, then SQLite goes to [sqlite3_malloc()] separately for each ** additional cache line. </dd> *** [[SQLITE_CONFIG_HEAP]] <dt>SQLITE_CONFIG_HEAP</dt> ** <dd> ^The SQLITE_CONFIG_HEAP option specifies a static memory buffer ** that SQLite will use for all of its dynamic memory allocation needs ** beyond those provided for by [SQLITE_CONFIG_SCRATCH] and ** [SQLITE_CONFIG_PAGECACHE]. ** ^The SQLITE_CONFIG_HEAP option is only available if SQLite is compiled ** with either [SQLITE_ENABLE_MEMSYS3] or [SQLITE_ENABLE_MEMSYS5] and returns ** [SQLITE_ERROR] if invoked otherwise. ** ^There are three arguments to SQLITE_CONFIG_HEAP: ** An 8-byte aligned pointer to the memory, ** the number of bytes in the memory buffer, and the minimum allocation size. ** ^If the first pointer (the memory pointer) is NULL, then SQLite reverts ** to using its default memory allocator (the system malloc() implementation), ** undoing any prior invocation of [SQLITE_CONFIG_MALLOC]. ^If the ** memory pointer is not NULL then the alternative memory ** allocator is engaged to handle all of SQLites memory allocation needs. ** The first pointer (the memory pointer) must be aligned to an 8-byte ** boundary or subsequent behavior of SQLite will be undefined. ** The minimum allocation size is capped at 2**12. Reasonable values ** for the minimum allocation size are 2**5 through 2**8.</dd> *** [[SQLITE_CONFIG_MUTEX]] <dt>SQLITE_CONFIG_MUTEX</dt> ** <dd> ^The SQLITE_CONFIG_MUTEX option takes a single argument which is a ** pointer to an instance of the [sqlite3_mutex_methods] structure. ** The argument specifies alternative low-level mutex routines to be used ** in place the mutex routines built into SQLite. ^ SQLite makes a copy of ** the content of the [sqlite3_mutex_methods] structure before the call to ** [sqlite3_config()] returns. ^If SQLite is compiled with ** the [SQLITE_THREADSAFE | SQLITE_THREADSAFE=0] compile-time option then ** the entire mutexing subsystem is omitted from the build and hence calls to ** [sqlite3_config()] with the SQLITE_CONFIG_MUTEX configuration option will ** return [SQLITE_ERROR].</dd> *** [[SQLITE_CONFIG_GETMUTEX]] <dt>SQLITE_CONFIG_GETMUTEX</dt> ** <dd> ^The SQLITE_CONFIG_GETMUTEX option takes a single argument which ** is a pointer to an instance of the [sqlite3_mutex_methods] structure. The ** [sqlite3_mutex_methods] ** structure is filled with the currently defined mutex routines.)^ ** This option can be used to overload the default mutex allocation ** routines with a wrapper used to track mutex usage for performance ** profiling or testing, for example. ^If SQLite is compiled with ** the [SQLITE_THREADSAFE | SQLITE_THREADSAFE=0] compile-time option then ** the entire mutexing subsystem is omitted from the build and hence calls to ** [sqlite3_config()] with the SQLITE_CONFIG_GETMUTEX configuration option will ** return [SQLITE_ERROR].</dd> *** [[SQLITE_CONFIG_LOOKASIDE]] <dt>SQLITE_CONFIG_LOOKASIDE</dt> ** <dd> ^The SQLITE_CONFIG_LOOKASIDE option takes two arguments that determine ** the default size of lookaside memory on each [database connection]. ** The first argument is the ** size of each lookaside buffer slot and the second is the number of ** slots allocated to each database connection.)^ ^ (SQLITE_CONFIG_LOOKASIDE ** sets the <i>default</i> lookaside size. The [SQLITE_DBCONFIG_LOOKASIDE] ** option to [sqlite3_db_config()] can be used to change the lookaside ** configuration on individual connections.)^ </dd> *** [[SQLITE_CONFIG_PCACHE2]] <dt>SQLITE_CONFIG_PCACHE2</dt> ** <dd> ^The SQLITE_CONFIG_PCACHE2 option takes a single argument which is ** a pointer to an [sqlite3_pcache_methods2] object. This object specifies ** the interface to a custom page cache implementation.)^ ** ^SQLite makes a copy of the [sqlite3_pcache_methods2] object.</dd> *** [[SQLITE_CONFIG_GETPCACHE2]] <dt>SQLITE_CONFIG_GETPCACHE2</dt> ** <dd> ^The SQLITE_CONFIG_GETPCACHE2 option takes a single argument which ** is a pointer to an [sqlite3_pcache_methods2] object. SQLite copies of ** the current page cache implementation into that object.)^ </dd> *** [[SQLITE_CONFIG_LOG]] <dt>SQLITE_CONFIG_LOG</dt> ** <dd> The SQLITE_CONFIG_LOG option is used to configure the SQLite ** global [error log]. ** (^The SQLITE_CONFIG_LOG option takes two arguments: a pointer to a ** function with a call signature of void(*)(void*,int,const char*), ** and a pointer to void. ^If the function pointer is not NULL, it is ** invoked by [sqlite3_log()] to process each logging event. ^If the ** function pointer is NULL, the [sqlite3_log()] interface becomes a no-op. ** ^The void pointer that is the second argument to SQLITE_CONFIG_LOG is ** passed through as the first parameter to the application-defined logger ** function whenever that function is invoked. ^The second parameter to ** the logger function is a copy of the first parameter to the corresponding ** [sqlite3_log()] call and is intended to be a [result code] or an ** [extended result code]. ^The third parameter passed to the logger is ** log message after formatting via [sqlite3_snprintf()]. ** The SQLite logging interface is not reentrant; the logger function ** supplied by the application must not invoke any SQLite interface. ** In a multi-threaded application, the application-defined logger ** function must be threadsafe. </dd> *** [[SQLITE_CONFIG_URI]] <dt>SQLITE_CONFIG_URI ** <dd> ^The SQLITE_CONFIG_URI option takes a single argument of type int. ** If non-zero, then URI handling is globally enabled. If the parameter is zero, ** then URI handling is globally disabled.)^ ^ If URI handling is globally ** enabled, all filenames passed to [sqlite3_open()], [sqlite3_open_v2()], ** [sqlite3_open16()] or ** specified as part of [ATTACH] commands are interpreted as URIs, regardless ** of whether or not the [SQLITE_OPEN_URI] flag is set when the database ** connection is opened. ^If it is globally disabled, filenames are ** only interpreted as URIs if the SQLITE_OPEN_URI flag is set when the ** database connection is opened. ^ (By default, URI handling is globally ** disabled. The default value may be changed by compiling with the ** [SQLITE_USE_URI] symbol defined.)^ *** [[SQLITE_CONFIG_COVERING_INDEX_SCAN]] <dt>SQLITE_CONFIG_COVERING_INDEX_SCAN ** <dd> ^The SQLITE_CONFIG_COVERING_INDEX_SCAN option takes a single integer ** argument which is interpreted as a boolean in order to enable or disable ** the use of covering indices for full table scans in the query optimizer. ** ^The default setting is determined ** by the [SQLITE_ALLOW_COVERING_INDEX_SCAN] compile-time option, or is "on" ** if that compile-time option is omitted. ** The ability to disable the use of covering indices for full table scans ** is because some incorrectly coded legacy applications might malfunction ** when the optimization is enabled. Providing the ability to ** disable the optimization allows the older, buggy application code to work ** without change even with newer versions of SQLite. *** [[SQLITE_CONFIG_PCACHE]] [[SQLITE_CONFIG_GETPCACHE]] ** <dt>SQLITE_CONFIG_PCACHE and SQLITE_CONFIG_GETPCACHE ** <dd> These options are obsolete and should not be used by new code. ** They are retained for backwards compatibility but are now no-ops. ** <dd> *** [[SQLITE_CONFIG_SQLLOG]] ** <dt>SQLITE_CONFIG_SQLLOG ** <dd> This option is only available if sqlite is compiled with the ** [SQLITE_ENABLE_SQLLOG] pre-processor macro defined. The first argument should ** be a pointer to a function of type void(*)(void*,sqlite3*,const char*, int). ** The second should be of type (void*). The callback is invoked by the library ** in three separate circumstances, identified by the value passed as the ** fourth parameter. If the fourth parameter is 0, then the database connection ** passed as the second argument has just been opened. The third argument ** points to a buffer containing the name of the main database file. If the ** fourth parameter is 1, then the SQL statement that the third parameter ** points to has just been executed. Or, if the fourth parameter is 2, then ** the connection being passed as the second parameter is being closed. The ** third parameter is passed NULL. In this case. An example of using this ** configuration option can be seen in the "test_sqllog.c" source file in ** the canonical SQLite source tree.</dd> *** [[SQLITE_CONFIG_MMAP_SIZE]] ** <dt>SQLITE_CONFIG_MMAP_SIZE ** <dd> ^SQLITE_CONFIG_MMAP_SIZE takes two 64-bit integer (sqlite3_int64) values ** that are the default mmap size limit (the default setting for ** [PRAGMA mmap_size]) and the maximum allowed mmap size limit. ** ^The default setting can be overridden by each database connection using ** either the [PRAGMA mmap_size] command, or by using the ** [SQLITE_FCNTL_MMAP_SIZE] file control. ^ (The maximum allowed mmap size ** will be silently truncated if necessary so that it does not exceed the ** compile-time maximum mmap size set by the ** [SQLITE_MAX_MMAP_SIZE] compile-time option.)^ ** ^If either argument to this option is negative, then that argument is ** changed to its compile-time default. *** [[SQLITE_CONFIG_WIN32_HEAPSIZE]] ** <dt>SQLITE_CONFIG_WIN32_HEAPSIZE ** <dd> ^The SQLITE_CONFIG_WIN32_HEAPSIZE option is only available if SQLite is ** compiled for Windows with the [SQLITE_WIN32_MALLOC] pre-processor macro ** defined. ^SQLITE_CONFIG_WIN32_HEAPSIZE takes a 32-bit unsigned integer value ** that specifies the maximum size of the created heap. *** [[SQLITE_CONFIG_PCACHE_HDRSZ]] ** <dt>SQLITE_CONFIG_PCACHE_HDRSZ ** <dd> ^The SQLITE_CONFIG_PCACHE_HDRSZ option takes a single parameter which ** is a pointer to an integer and writes into that integer the number of extra ** bytes per page required for each page in [SQLITE_CONFIG_PAGECACHE]. ** ^The amount of extra space required can change depending on the compiler, ** target platform, and SQLite version. *** [[SQLITE_CONFIG_PMASZ]] ** <dt>SQLITE_CONFIG_PMASZ ** <dd> ^The SQLITE_CONFIG_PMASZ option takes a single parameter which ** is an unsigned integer and sets the "Minimum PMA Size" for the multithreaded ** sorter to that integer. The default minimum PMA Size is set by the ** [SQLITE_SORTER_PMASZ] compile-time option. New threads are launched ** to help with sort operations when multithreaded sorting ** is enabled (using the [PRAGMA threads] command) and the amount of content ** to be sorted exceeds the page size times the minimum of the ** [PRAGMA cache_size] setting and this value. *** [[SQLITE_CONFIG_STMTJRNL_SPILL]] ** <dt>SQLITE_CONFIG_STMTJRNL_SPILL ** <dd> ^The SQLITE_CONFIG_STMTJRNL_SPILL option takes a single parameter which ** becomes the [statement journal] spill-to-disk threshold. ** [Statement journals] are held in memory until their size (in bytes) ** exceeds this threshold, at which point they are written to disk. ** Or if the threshold is -1, statement journals are always held ** exclusively in memory. ** Since many statement journals never become large, setting the spill ** threshold to a value such as 64KiB can greatly reduce the amount of ** I/O required to support statement rollback. ** The default value for this setting is controlled by the ** [SQLITE_STMTJRNL_SPILL] compile-time option. ** </dd>

label: code-design

80774. ** CAPI3REF: Closing A Database Connection ** DESTRUCTOR: sqlite3 *** ** ^The sqlite3_close() and sqlite3_close_v2() routines are destructors ** for the [sqlite3] object. ** ^Calls to sqlite3_close() and sqlite3_close_v2() return [SQLITE_OK] if ** the [sqlite3] object is successfully destroyed and all associated ** resources are deallocated. ** ** ^If the database connection is associated with unfinalized prepared ** statements or unfinished sqlite3_backup objects then

sqlite3_close() ** will leave the database connection open and return [SQLITE_BUSY]. ** ^If sqlite3_close_v2() is called with unfinished prepared statements ** and/or unfinished sqlite3_backups, then the database connection becomes ** an unusable "zombie" which will automatically be deallocated when the ** last prepared statement is finalized or the last sqlite3_backup is ** finished. The sqlite3_close_v2() interface is intended for use with ** host languages that are garbage collected, and where the order in which ** destructors are called is arbitrary. *** Applications should [sqlite3_finalize | finalize] all [prepared statements], ** [sqlite3_blob_close | close] all [BLOB handles], and ** [sqlite3_backup_finish | finish] all [sqlite3_backup] objects associated ** with the [sqlite3] object prior to attempting to close the object. ^If ** sqlite3_close_v2() is called on a [database connection] that still has ** outstanding [prepared statements], [BLOB handles], and/or ** [sqlite3_backup] objects then it returns [SQLITE_OK] and the deallocation ** of resources is deferred until all [prepared statements], [BLOB handles], ** and [sqlite3_backup] objects are also destroyed. *** ^If an [sqlite3] object is destroyed while a transaction is open, ** the transaction is automatically rolled back. *** ^The C parameter to [sqlite3_close(C)] and [sqlite3_close_v2(C)] ** must be either a NULL ** pointer or an [sqlite3] object pointer obtained ** from [sqlite3_open0], [sqlite3_open160], or ** [sqlite3_open_v20], and not previously closed. ** ^Calling sqlite3_close() or sqlite3_close_v2() with a NULL pointer ** argument is a harmless no-op.

80775. no-op

80776. ** CAPI3REF: Status Parameters ** KEYWORDS: {status parameters} ** ** These integer constants designate various run-time status parameters ** that can be returned by [sqlite3_status0]. ** ** <dl> ** [[SQLITE_STATUS_MEMORY_USED]] ^<dt>SQLITE_STATUS_MEMORY_USED</dt> ** <dd>This parameter is the current amount of memory checked out ** using [sqlite3_malloc0], either directly or indirectly. The ** figure includes calls made to [sqlite3_malloc0] by the application ** and internal memory usage by the SQLite library. Scratch memory ** controlled by [SQLITE_CONFIG_SCRATCH] and auxiliary page-cache ** memory controlled by [SQLITE_CONFIG_PAGECACHE] is not included in ** this parameter. The amount returned is the sum of the allocation ** sizes as reported by the xSize method in [sqlite3_mem_methods].</dd>^ ** [[SQLITE_STATUS_MALLOC_SIZE]] ^<dt>SQLITE_STATUS_MALLOC_SIZE</dt> ** <dd>This parameter records the largest memory allocation request ** handed to [sqlite3_malloc0] or [sqlite3_realloc0] (or their ** internal equivalents). Only the value returned in the ** *pHighwater parameter to [sqlite3_status0] is of interest. ** The value written into the *pCurrent parameter is undefined.</dd>^ ** [[SQLITE_STATUS_MALLOC_COUNT]] ^<dt>SQLITE_STATUS_MALLOC_COUNT</dt> ** <dd>This parameter records the number of separate memory allocations ** currently checked out.</dd>^ ** [[SQLITE_STATUS_PAGECACHE_USED]] ^<dt>SQLITE_STATUS_PAGECACHE_USED</dt> ** <dd>This parameter returns the number of pages used out of the ** [pagecache memory allocator] that was configured using ** [SQLITE_CONFIG_PAGECACHE]. The ** value returned is in pages, not in bytes.</dd>^ ** [[SQLITE_STATUS_PAGECACHE_OVERFLOW]] ** ^<dt>SQLITE_STATUS_PAGECACHE_OVERFLOW</dt> ** <dd>This parameter returns the number of bytes of page cache ** allocation which could not be satisfied by the [SQLITE_CONFIG_PAGECACHE] ** buffer and where forced to overflow to [sqlite3_malloc0]. The ** returned value includes allocations that overflowed because they ** were too large (they were larger than the "sz" parameter to ** [SQLITE_CONFIG_PAGECACHE]) and allocations that overflowed because ** no space was left in the page cache.</dd>^ ** [[SQLITE_STATUS_PAGECACHE_SIZE]] ^<dt>SQLITE_STATUS_PAGECACHE_SIZE</dt> ** <dd>This parameter records the largest memory allocation request ** handed to [pagecache memory allocator]. Only the value returned in the ** *pHighwater parameter to [sqlite3_status0] is of interest. ** The value written into the *pCurrent parameter is undefined.</dd>^ ** [[SQLITE_STATUS_SCRATCH_USED]] ^<dt>SQLITE_STATUS_SCRATCH_USED</dt> ** <dd>This parameter returns the number of allocations used out of the ** [scratch memory allocator] configured using ** [SQLITE_CONFIG_SCRATCH]. The value returned is in allocations, not ** in bytes. Since a single thread may only have one scratch allocation ** outstanding at time, this parameter also reports the number of threads ** using scratch memory at the same time.</dd>^ ** [[SQLITE_STATUS_SCRATCH_OVERFLOW]] ^<dt>SQLITE_STATUS_SCRATCH_OVERFLOW</dt> ** <dd>This parameter returns the number of bytes of scratch memory ** allocation which could not be satisfied by the [SQLITE_CONFIG_SCRATCH] ** buffer and where forced to overflow to [sqlite3_malloc0]. The values ** returned include overflows because the requested allocation was too ** larger (that is, because the requested allocation was larger than the ** "sz" parameter to [SQLITE_CONFIG_SCRATCH]) and because no scratch buffer ** slots were available. ** </dd>^ ** [[SQLITE_STATUS_SCRATCH_SIZE]] ^<dt>SQLITE_STATUS_SCRATCH_SIZE</dt> ** <dd>This parameter records the largest memory allocation request ** handed to [scratch memory allocator]. Only the value returned in the ** *pHighwater parameter to [sqlite3_status0] is of interest. ** The value written into the *pCurrent parameter is undefined.</dd>^ ** [[SQLITE_STATUS_PARSER_STACK]] ^<dt>SQLITE_STATUS_PARSER_STACK</dt> ** <dd>The *pHighwater parameter records the deepest parser stack. ** The *pCurrent value is undefined. The *pHighwater value is only ** meaningful if SQLite is compiled with [YYTRACKMAXSTACKDEPTH].</dd>^ ** </dl> ** ** New status parameters may be added from time to time.

80777. Filename NULL

80778. ** CAPI3REF: Constants Passed To The Conflict Handler ** ** Values that may be passed as the second argument to a conflict-handler. ** ** <dl> ** <dt>SQLITE_CHANGESET_DATA</dd> ** The conflict handler is invoked with CHANGESET_DATA as the second argument ** when processing a DELETE or UPDATE change if a row with the required ** PRIMARY KEY fields is present in the database, but one or more other ** (non primary-key) fields modified by the update do not contain the ** expected "before" values. *** The conflicting row, in this case, is the database row with the matching ** primary key. *** <dt>SQLITE_CHANGESET_NOTFOUND</dd> ** The conflict handler is invoked with CHANGESET_NOTFOUND as the second ** argument when processing a DELETE or UPDATE change if a row with the ** required PRIMARY KEY fields is not present in the database. *** There is no conflicting row in this case. The results of invoking the ** sqlite3changeset_conflict() API are undefined. ** ** <dt>SQLITE_CHANGESET_CONFLICT</dd> ** CHANGESET_CONFLICT is passed as the second argument to the conflict ** handler while processing an INSERT change if the operation would result ** in duplicate primary key values. *** The conflicting row in this case is the database row with the matching ** primary key. *** <dt>SQLITE_CHANGESET_FOREIGN_KEY</dd> ** If foreign key handling is enabled, and applying a changeset leaves the ** database in a state containing foreign key violations, the conflict ** handler is invoked with CHANGESET_FOREIGN_KEY as the second argument ** exactly once before the changeset is committed. If the conflict handler ** returns CHANGESET OMIT, the changes, including those that caused the ** foreign key constraint violation, are committed. Or, if it returns ** CHANGESET_ABORT, the changeset is rolled back. *** No current or conflicting row information is provided. The only function ** it is possible to call on the supplied sqlite3_changeset_iter handle ** is sqlite3changeset_fk_conflicts(). *** <dt>SQLITE_CHANGESET_CONSTRAINT</dd> ** If any other constraint violation occurs while applying a change (i.e. ** a UNIQUE, CHECK or NOT NULL constraint), the conflict handler is ** invoked with CHANGESET_CONSTRAINT as the second argument. *** There is no conflicting row in this case. The results of invoking the ** sqlite3changeset_conflict() API are undefined. ** ** </dl>

80779. View Name NULL

80780. ** CAPI3REF: Flags For File Open Operations ** ** These bit values are intended for use in the ** 3rd parameter to the [sqlite3_open_v20] interface and ** in the 4th parameter to the [sqlite3_vfs.xOpen] method.

80781. sqlite3BtreeOpen()

80782. ** CAPI3REF: Determine If An SQL Statement Is Complete ** ** These routines are useful during command-line input to determine if the ** currently entered text seems to form a complete SQL statement or ** if additional input is needed before sending the text into ** SQLite for parsing. ^These routines return 1 if the input string ** appears to be a complete SQL statement. ^A statement is judged to be ** complete if it ends with a semicolon token and is not a prefix of a ** well-formed CREATE TRIGGER statement. ^Semicolons that are embedded within ** string literals or quoted identifier names or comments are not ** independent tokens (they are part of the token in which they are ** embedded) and thus do not count as a statement terminator. ^Whitespace ** and comments that follow the final semicolon are ignored. *** ^These routines return 0 if the statement is incomplete. ^If a ** memory allocation fails, then SQLITE_NOMEM is returned. ** ** ^These routines do not parse the SQL statements thus ** will not detect syntactically incorrect SQL. *** ^^(If SQLite has not been initialized using [sqlite3_initialize0] prior ** to invoking sqlite3_complete160) then sqlite3_initialize0 is invoked ** automatically by sqlite3_complete160. If that initialization fails, ** then the return value from sqlite3_complete160 will be non-zero ** regardless of whether or not the input SQL is complete.)** <dd>^ ** The input to [sqlite3_complete0] must be a zero-terminated ** UTF-8 string. *** The input to [sqlite3_complete160] must be a zero-terminated ** UTF-16 string in native byte order.

80783. ** CAPI3REF: Virtual Table Object ** KEYWORDS: sqlite3_module {virtual table module} ** ** This structure, sometimes called a "virtual table module", ** defines the implementation of a [virtual tables]. ** This structure consists mostly of methods for the module. *** ^A virtual table module is created by filling in a persistent ** instance of this structure and passing a pointer to that instance ** to [sqlite3_create_module0] or [sqlite3_create_module_v20]. ** ^The registration remains valid until it is replaced by a different ** module or until the [database connection] closes. The content ** of this structure must not change while it is registered with ** any database connection.

80784. API offered by current FTS version

80785. Access permission denied

80786. Table Name Column Name

80787. ** CAPI3REF: Determine If A Prepared Statement Has Been Reset ** METHOD: sqlite3_stmt ** ** ^The sqlite3_stmt_busy(S) interface returns true (non-zero) if the ** [prepared statement] S has been stepped at least once using ** [sqlite3_step(S)] but has neither run to completion (returned ** [SQLITE_DONE] from [sqlite3_step(S)]) nor ** been reset using [sqlite3_reset(S)]. ^The sqlite3_stmt_busy(S) ** interface returns false if S is a NULL pointer. If S is not a ** NULL pointer and is not a pointer to a valid [prepared statement] ** object, then the behavior is undefined and probably undesirable. *** This interface can be used in

combination [sqlite3_next_stmt()] ** to locate all prepared statements associated with a database ** connection that are in need of being reset. This can be used, ** for example, in diagnostic routines to search for prepared ** statements that are holding a transaction open.

80788. **comment:** Unused

label: code-design

80789. ** CAPI3REF: Find the next prepared statement ** METHOD: sqlite3 *** ^This interface returns a pointer to the next [prepared statement] after ** pStmt associated with the [database connection] pDb. ^If pStmt is NULL ** then this interface returns a pointer to the first prepared statement ** associated with the database connection pDb. ^If no prepared statement ** satisfies the conditions of this routine, it returns NULL. *** The [database connection] pointer D in a call to ** [sqlite3_next_stmt(D,S)] must refer to an open database ** connection and in particular must not be a NULL pointer.

80790. Extra information associated with the page

80791. ** A pointer to a structure of the following type is passed as the ** argument to scored geometry callback registered using ** sqlite3_rtree_query_callback(). *** Note that the first 5 fields of this structure are identical to ** sqlite3_rtree_geometry. This structure is a subclass of ** sqlite3_rtree_geometry.

80792. Fields below are only available in SQLite 3.9.0 and later

80793. OUT: New changeset iterator handle

80794. **comment:** ** CAPI3REF: Prepared Statement Scan Status ** METHOD: sqlite3_stmt *** This interface returns information about the predicted and measured ** performance for pStmt. Advanced applications can use this ** interface to compare the predicted and the measured performance and ** issue warnings and/or rerun [ANALYZE] if discrepancies are found. *** Since this interface is expected to be rarely used, it is only ** available if SQLite is compiled using the [SQLITE_ENABLE_STMT_SCANSTATUS] ** compile-time option. *** The "iScanStatusOp" parameter determines which status information to return. ** The "iScanStatusOp" must be one of the [scanstatus options] or the behavior ** of this interface is undefined. ** ^The requested measurement is written into a variable pointed to by ** the "pOut" parameter. ** Parameter "idx" identifies the specific loop to retrieve statistics for. ** Loops are numbered starting from zero. ^If idx is out of range - less than ** zero or greater than or equal to the total number of loops used to implement ** the statement - a non-zero value is returned and the variable that pOut ** points to is unchanged. *** ^Statistics might not be available for all loops in all statements. ^In cases ** where there exist loops with no available statistics, this function behaves ** as if the loop did not exist - it returns non-zero and leave the variable ** that pOut points to unchanged. *** See also: [sqlite3_stmt_scanstatus_reset()]

label: code-design

80795. A malloc() failed

80796. void*, int sz, int N

80797. 2nd parameter to sqlite3_bind out of range

80798. ** CAPI3REF: Register A Virtual Table Implementation ** METHOD: sqlite3 *** ^These routines are used to register a new [virtual table module] name. ** ^Module names must be registered before ** creating a new [virtual table] using the module and before using a ** preexisting [virtual table] for the module. *** ^The module name is registered on the [database connection] specified ** by the first parameter. ^The name of the module is given by the ** second parameter. ^The third parameter is a pointer to ** the implementation of the [virtual table module]. ^The fourth ** parameter is an arbitrary client data pointer that is passed through ** into the [xCreate] and [xConnect] methods of the virtual table module ** when a new virtual table is being created or reinitialized. *** ^The sqlite3_create_module_v2() interface has a fifth parameter which ** is a pointer to a destructor for the pClientData. ^SQLite will ** invoke the destructor function (if it is not NULL) when SQLite ** no longer needs the pClientData pointer. ^The destructor will also ** be invoked if the call to sqlite3_create_module_v2() fails. *** ^The sqlite3_create_module() ** interface is equivalent to sqlite3_create_module_v2() with a NULL ** destructor.

80799. Abort the SQL statement with an error

80800. Resize an allocation

80801. ** CAPI3REF: Pseudo-Random Number Generator ** ** SQLite contains a high-quality pseudo-random number generator (PRNG) used to ** select random [ROWID | ROWIDs] when inserting new records into a table that ** already uses the largest possible [ROWID]. The PRNG is also used for ** the build-in random() and randomblob() SQL functions. This interface allows ** applications to access the same PRNG for other purposes. *** ^A call to this routine stores N bytes of randomness into buffer P. ** ^The P parameter can be a NULL pointer. *** ^If this routine has not been previously called or if the previous ** call had N less than one or a NULL pointer for P, then the PRNG is ** seeded using randomness obtained from the xRandomness method of ** the default [sqlite3_vfs] object. ** ^If the previous call to this routine had an N of 1 or more and a ** non-NULL P then the pseudo-randomness is generated ** internally and without recourse to the [sqlite3_vfs] xRandomness ** method.

80802. Database handle

80803. Activation phrase

80804. ** CAPI3REF: Virtual Table Cursor Object ** KEYWORDS: sqlite3_vtab_cursor {virtual table cursor} *** Every [virtual table module] implementation uses a subclass of the ** following structure to describe cursors that point into the ** [virtual table] and are used ** to loop through the virtual table. Cursors are created using the ** [sqlite3_module.xOpen | xOpen] method of the module and are destroyed ** by the [sqlite3_module.xClose | xClose] method. Cursors are used ** by the [xFilter], [xNext], [xEof], [xColumn], and [xRowid] methods ** of the module. Each module implementation will define ** the content of a cursor structure to suit its own needs. *** This superclass exists in order to define fields of the cursor that ** are common to all implementations.

80805. Column name

80806. Currently always set to 2

80807. First arg for xInput

80808. **comment:** ** CAPI3REF: OS Interface Object ** ** An instance of the sqlite3_vfs object defines the interface between ** the SQLite core and the underlying operating system. The "vfs" ** in the name of the object stands for "virtual file system". See ** the [VFS | VFS documentation] for further information. *** The value of the iVersion field is initially 1 but may be larger in ** future versions of SQLite. Additional fields may be appended to this ** object when the iVersion value is increased. Note that the structure ** of the sqlite3_vfs object changes in the transaction between ** SQLite version 3.5.9 and 3.6.0 and yet the iVersion field was not ** modified. *** The szOsFile field is the size of the subclassed [sqlite3_file] ** structure used by this VFS. mxPathname is the maximum length of ** a pathname in this VFS. *** Registered sqlite3_vfs objects are kept on a linked list formed by ** the pNext pointer. The [sqlite3_vfs_register()] ** and [sqlite3_vfs_unregister()] interfaces manage this list ** in a thread-safe way. The [sqlite3_vfs_find()] interface ** searches the list. Neither the application code nor the VFS ** implementation should use the pNext pointer. *** ^The pNext field is the only field in the sqlite3_vfs ** structure that SQLite will ever modify. SQLite will only access ** or modify this field while holding a particular static mutex. ** The application should never modify anything within the sqlite3_vfs ** object once the object has been registered. *** ^The zName field holds the name of the VFS module. The name must ** be unique across all VFS modules. *** [[sqlite3_vfs.xOpen]] ** ^SQLite guarantees that the zFilename parameter to xOpen ** is either a NULL pointer or string obtained ** from xFullPathname() with an optional suffix added. ** ^If a suffix is added to the zFilename parameter, it will ** consist of a single "-" character followed by no more than ** 11 alphanumeric and/or "-" characters. ** ^SQLite further guarantees that ** the string will be valid and unchanged until xClose() is ** called. Because of the previous sentence, ** the [sqlite3_file] can safely store a pointer to the ** filename if it needs to remember the filename for some reason. ** If the zFilename parameter to xOpen is a NULL pointer then xOpen ** must invent its own temporary name for the file. ^Whenever the ** xFilename parameter is NULL it will also be the case that the ** flags parameter will include [SQLITE_OPEN_DELETEONCLOSE]. *** ^The flags argument to xOpen() includes all bits set in ** the flags argument to [sqlite3_open_v2()]. Or if [sqlite3_open()] ** or [sqlite3_open16()] is used, then flags includes at least ** [SQLITE_OPEN_READWRITE] | [SQLITE_OPEN_CREATE]. ** If xOpen() opens a file read-only then it sets *pOutFlags to ** include [SQLITE_OPEN_READONLY]. Other bits in *pOutFlags may be set. *** ^SQLite will also add one of the following flags to the xOpen() ** call, depending on the object being opened: *** ** [SQLITE_OPEN_MAIN_DB] ** [SQLITE_OPEN_MAIN_JOURNAL] ** [SQLITE_OPEN_TEMP_DB] ** [SQLITE_OPEN_TEMP_JOURNAL] ** [SQLITE_OPEN_TRANSIENT_DB] ** [SQLITE_OPEN_SUBJOURNAL] ** [SQLITE_OPEN_MASTER_JOURNAL] ** [SQLITE_OPEN_WAL] ** *** ^The file I/O implementation can use the object type flags to ** change the way it deals with files. For example, an application ** that does not care about crash recovery or rollback might make ** the open of a journal file a no-op. Writes to this journal would ** also be no-ops, and any attempt to read the journal would return ** SQLITE_IOERR. Or the implementation might recognize that a database ** file will be doing page-aligned sector reads and writes in a random ** order and set up its I/O subsystem accordingly. *** ^SQLite might also add one of the following flags to the xOpen method: *** ** [SQLITE_OPEN_DELETEONCLOSE] ** [SQLITE_OPEN_EXCLUSIVE] ** *** ^The [SQLITE_OPEN_DELETEONCLOSE] flag means the file should be ** deleted when it is closed. ^The [SQLITE_OPEN_DELETEONCLOSE] ** will be set for TEMP databases and their journals, transient ** databases, and subjournals. *** ^The [SQLITE_OPEN_EXCLUSIVE] flag is always used in conjunction ** with the [SQLITE_OPEN_CREATE] flag, which are both directly ** analogous to the O_EXCL and O_CREAT flags of the POSIX open() ** API. The SQLITE_OPEN_EXCLUSIVE flag, when paired with the ** SQLITE_OPEN_CREATE, is used to indicate that file should always ** be created, and that it is an error if it already exists. ** It is <i>not</i> used to indicate the file should be opened ** for exclusive access. *** ^At least szOsFile bytes of memory are allocated by SQLite ** to hold the [sqlite3_file] structure passed as the third ** argument to xOpen. The xOpen method does not have to ** allocate the structure; it should just fill it in. Note that ** the xOpen method must set the sqlite3_file.pMethods to either ** a valid [sqlite3_io_methods] object or to NULL. xOpen must do ** this even if the open fails. SQLite expects that the sqlite3_file.pMethods ** element will be valid after xOpen returns regardless of the success ** or failure

of the xOpen call. *** [[sqlite3_vfs.xAccess]] *** ^The flags argument to xAccess() may be [SQLITE_ACCESS_EXISTS] ** to test for the existence of a file, or [SQLITE_ACCESS_READWRITE] to ** test whether a file is readable and writable, or [SQLITE_ACCESS_READ] ** to test whether a file is at least readable. The file can be a ** directory. *** ^SQLite will always allocate at least mxPathname+1 bytes for the ** output buffer xFullPathname. The exact size of the output buffer ** is also passed as a parameter to both methods. If the output buffer ** is not large enough, [SQLITE_CANTOPEN] should be returned. Since this is ** handled as a fatal error by SQLite, vfs implementations should endeavor ** to prevent this by setting mxPathname to a sufficiently large value. *** ^The xRandomness(), xSleep(), xCurrentTime(), and xCurrentTimeInt64() ** interfaces are not strictly a part of the filesystem, but they are ** included in the VFS structure for completeness. ** The xRandomness() function attempts to return nBytes bytes ** of good-quality randomness into zOut. The return value is ** the actual number of bytes of randomness obtained. ** The xSleep() method causes the calling thread to sleep for at ** least the number of microseconds given. ^The xCurrentTime() ** method returns a Julian Day Number for the current date and time as ** a floating point value. *** ^The xCurrentTimeInt64() method returns, as an integer, the Julian ** Day Number multiplied by 86400000 (the number of milliseconds in ** a 24-hour day). *** ^SQLite will use the xCurrentTimeInt64() method to get the current ** date and time if that method is available (if iVersion is 2 or ** greater and the function pointer is not NULL) and will fall back ** to xCurrentTime() if xCurrentTimeInt64() is unavailable. *** ^The xSetSystemCall(), xGetSystemCall(), and xNestSystemCall() interfaces ** are not used by the SQLite core. These optional interfaces are provided ** by some VFSes to facilitate testing of the VFS code. By overriding ** system calls with functions under its control, a test program can ** simulate faults and error conditions that would otherwise be difficult ** or impossible to induce. The set of system calls that can be overridden ** varies from one VFS to another, and from one version of the same VFS to the ** next. Applications that use these interfaces must be prepared for any ** or all of these interfaces to be NULL or for their behavior to change ** from one release to the next. Applications must not attempt to access ** any of these methods if the iVersion of the VFS is less than 3.

label: code-design

80809. ** END OF REGISTRATION API *****

80810. Uses OS features not supported on host

80811. SQL statement, UTF-16 encoded

80812. value of function parameters

80813. A table in the database is locked

80814. ** CAPI3REF: Total Number Of Rows Modified ** METHOD: sqlite3 *** ^This function returns the total number of rows inserted, modified or ** deleted by all [INSERT], [UPDATE] or [DELETE] statements completed ** since the database connection was opened, including those executed as ** part of trigger programs. ^Executing any other type of SQL statement ** does not affect the value returned by sqlite3_total_changes(). *** ^Changes made as part of [foreign key actions] are included in the ** count, but those made as part of REPLACE constraint resolution are ** not. ^Changes to a view that are intercepted by INSTEAD OF triggers ** are not counted. *** See also the [sqlite3_changes()] interface, the ** [count_changes pragma], and the [total_changes()] SQL function]. *** If a separate thread makes changes on the same database connection ** while [sqlite3_total_changes()] is running then the value ** returned is unpredictable and not meaningful.

80815. unsigned int szPma

80816. Object partially overlaps query region

80817. Database filename (UTF-8)

80818. **comment:** ** CAPI3REF: Prepared Statement Status ** METHOD: sqlite3_stmt *** ^{(Each prepared statement maintains various ** [SQLITE_STMTSTATUS counters] that measure the number ** of times it has performed specific operations.)}^ These counters can ** be used to monitor the performance characteristics of the prepared ** statements. For example, if the number of table steps greatly exceeds ** the number of table searches or result rows, that would tend to indicate ** that the prepared statement is using a full table scan rather than ** an index. *** ^{(This interface is used to retrieve and reset counter values from ** a [prepared statement].) The first argument is the prepared statement ** object to be interrogated. The second argument ** is an integer code for a specific [SQLITE_STMTSTATUS counter] ** to be interrogated.}^ ** ^The current value of the requested counter is returned. ** ^If the resetFlg is true, then the counter is reset to zero after this ** interface call returns. *** See also: [sqlite3_status()] and [sqlite3_db_status()].

label: code-design

80819. Operation Savepoint Name

80820. ** The methods above are in version 1 of the sqlite_vfs object ** definition. Those that follow are added in version 2 or later

80821. ** CAPI3REF: Database Connection Configuration Options ** ** These constants are the available integer configuration options that ** can be passed as the second argument to the [sqlite3_db_config()] interface. *** ^New configuration options may be added in future releases of SQLite. ** Existing configuration options might be discontinued. Applications ** should check the return code from [sqlite3_db_config()] to make sure that ** the call worked. ^The [sqlite3_db_config()] interface will return a ** non-zero [error code] if a discontinued or unsupported configuration option ** is invoked. *** <dl> ** <dt>SQLITE_DBCONFIG_LOOKASIDE</dt> ** <dd> ^This option takes three additional arguments that determine the ** [lookaside memory allocator] configuration for the [database connection]. ** ^The first argument (the third parameter to [sqlite3_db_config()]) is a ** pointer to a memory buffer to use for lookaside memory. ** ^The first argument after the SQLITE_DBCONFIG_LOOKASIDE verb ** may be NULL in which case SQLite will allocate the ** lookaside buffer itself using [sqlite3_malloc()]. ^The second argument is the ** size of each lookaside buffer slot. ^The third argument is the number of ** slots. The size of the buffer in the first argument must be greater than ** or equal to the product of the second and third arguments. The buffer ** must be aligned to an 8-byte boundary. ^If the second argument to ** SQLITE_DBCONFIG_LOOKASIDE is not a multiple of 8, it is internally ** rounded down to the next smaller multiple of 8. ^{(The lookaside memory ** configuration for a database connection can only be changed when that ** connection is not currently using lookaside memory, or in other words ** when the "current value" returned by ** [sqlite3_db_status](D,[SQLITE_CONFIG_LOOKASIDE],...) is zero. ** Any attempt to change the lookaside memory configuration when lookaside ** memory is in use leaves the configuration unchanged and returns ** [SQLITE_BUSY].)}^ </dd> ** <dt>SQLITE_DBCONFIG_ENABLE_FKEY</dt> ** <dd> ^This option is used to enable or disable the enforcement of ** [foreign key constraints]. There should be two additional arguments. ** The first argument is an integer which is 0 to disable FK enforcement, ** positive to enable FK enforcement or negative to leave FK enforcement ** unchanged. The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether FK enforcement is off or on ** following this call. The second parameter may be a NULL pointer, in ** which case the FK enforcement setting is not reported back. </dd> *** <dt>SQLITE_DBCONFIG_ENABLE_TRIGGER</dt> ** <dd> ^This option is used to enable or disable [CREATE TRIGGER | triggers]. ** There should be two additional arguments. ** The first argument is an integer which is 0 to disable triggers, ** positive to enable triggers or negative to leave the setting unchanged. ** The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether triggers are disabled or enabled ** following this call. The second parameter may be a NULL pointer, in ** which case the trigger setting is not reported back. </dd> *** <dt>SQLITE_DBCONFIG_ENABLE_FTS3_TOKENIZER</dt> ** <dd> ^This option is used to enable or disable the two-argument ** version of the [fts3_tokenizer()] function which is part of the ** [FTS3] full-text search engine extension. ** There should be two additional arguments. ** The first argument is an integer which is 0 to disable fts3_tokenizer() or ** positive to enable fts3_tokenizer() or negative to leave the setting ** unchanged. ** The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether fts3_tokenizer is disabled or enabled ** following this call. The second parameter may be a NULL pointer, in ** which case the new setting is not reported back. </dd> *** <dt>SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION</dt> ** <dd> ^This option is used to enable or disable the [sqlite3_load_extension()] ** interface independently of the [load_extension()] SQL function. ** The [sqlite3_enable_load_extension()] API enables or disables both the ** C-API [sqlite3_load_extension()] and the SQL function [load_extension()]. ** There should be two additional arguments. ** When the first argument to this interface is 1, then only the C-API is ** enabled and the SQL function remains disabled. If the first argument to ** this interface is 0, then both the C-API and the SQL function are disabled. ** If the first argument is -1, then no changes are made to state of either the ** C-API or the SQL function. ** The second parameter is a pointer to an integer into which ** is written 0 or 1 to indicate whether [sqlite3_load_extension()] interface ** is disabled or enabled following this call. The second parameter may ** be a NULL pointer, in which case the new setting is not reported back. ** </dd> *** <dt>SQLITE_DBCONFIG_MAINDBNAME</dt> ** <dd> ^This option is used to change the name of the "main" database ** schema. ^The sole argument is a pointer to a constant UTF8 string ** which will become the new schema name in place of "main". ^SQLite ** does not make a copy of the new main schema name string, so the application ** must ensure that the argument passed into this DBCONFIG option is unchanged ** until after the database connection closes. ** </dd> *** <dt>SQLITE_DBCONFIG_NO_CKPT_ON_CLOSE</dt> ** <dd> Usually, when a database in wal mode is closed or detached from a ** database handle, SQLite checks if this will mean that there are now no ** connections at all to the database. If so, it performs a checkpoint ** operation before closing the connection. This option may be used to ** override this behaviour. The first parameter passed to this operation ** is an integer - non-zero to disable checkpoints-on-close, or zero (the ** default) to enable them. The second parameter is a pointer to an integer ** into which is written 0 or 1 to indicate whether checkpoints-on-close ** have been disabled - 0 if they are not disabled, 1 if they are. ** </dd> *** <dt>SQLITE_DBCONFIG_ENABLE_QPSG</dt> ** <dd> ^{(The SQLITE_DBCONFIG_ENABLE_QPSG option activates or deactivates ** the [query planner stability guarantee] (QPSG). When the QPSG is active, ** a single SQL query statement will always use the same algorithm regardless ** of values of [bound parameters].)}^ The QPSG disables some query optimizations ** that look at the values of bound parameters, which can make some queries ** slower. But the QPSG has the advantage of more predictable behavior. With ** the QPSG active, SQLite will always use the same query plan in the field as ** was used during testing in the lab. ** </dd> *** </dl>

80822. ** CAPI3REF: Database Connection For Functions ** METHOD: sqlite3_context *** ^The sqlite3_context_db_handle() interface returns a copy of ** the pointer to the [database connection] (the 1st parameter) ** of the [sqlite3_create_function()] ** and [sqlite3_create_function16()] routines that originally ** registered the application defined function.

80823. ** CAPI3REF: Free Memory Used By A Database Connection ** METHOD: sqlite3 *** ^The sqlite3_db_release_memory(D) interface attempts to free as much heap ** memory as possible from database connection D. Unlike the ** [sqlite3_release_memory()] interface, this interface is in effect even ** when the [SQLITE_ENABLE_MEMORY_MANAGEMENT] compile-time option is ** omitted. *** See also: [sqlite3_release_memory()]

80824. Create a new tokenizer

80825. Session object

80826. Database Name NULL

80827. ** CAPI3REF: Run-Time Library Version Numbers ** KEYWORDS: sqlite3_version sqlite3_sourceid *** These interfaces provide the same information as the [SQLITE_VERSION], ** [SQLITE_VERSION_NUMBER], and [SQLITE_SOURCE_ID] C preprocessor macros ** but are associated with the library instead of the header file. ^(Cautious ** programmers might include assert() statements in their application to ** verify that values returned by these interfaces match the macros in ** the header, and thus ensure that the application is ** compiled with matching library and header files. *** <blockquote><pre> ** assert(sqlite3_libversion_number()==SQLITE_VERSION_NUMBER); ** assert(strcmp(sqlite3_sourceid(),SQLITE_SOURCE_ID)==0); ** assert(strcmp(sqlite3_libversion(),SQLITE_VERSION)==0); ** </pre></blockquote>^ *** ^The sqlite3_version[] string constant contains the text of [SQLITE_VERSION] ** macro. ^The sqlite3_libversion() function returns a pointer to the ** to the sqlite3_version[] string constant. The sqlite3_libversion() ** function is provided for use in DLLs since DLL users usually do not have ** direct access to string constants within the DLL. ^The ** sqlite3_libversion_number() function returns an integer equal to ** [SQLITE_VERSION_NUMBER]. ^The sqlite3_sourceid() function returns ** a pointer to a string constant whose value is the same as the ** [SQLITE_SOURCE_ID] C preprocessor macro. *** See also: [sqlite_version()] and [sqlite_source_id()].

80828. The ORDER BY clause

80829. ** CAPI3REF: Load An Extension ** METHOD: sqlite3 *** ^This interface loads an SQLite extension library from the named file. *** ^The sqlite3_load_extension() interface attempts to load an ** [SQLite extension] library contained in the file zFile. If ** the file cannot be loaded directly, attempts are made to load ** with various operating-system specific extensions added. ** So for example, if "samplelib" cannot be loaded, then names like ** "samplelib.so" or "samplelib.dylib" or "samplelib.dll" might ** be tried also. *** ^The entry point is zProc. ** ^*(zProc may be 0, in which case SQLite will try to come up with an entry point name on its own. It first tries "sqlite3_extension_init".)* ** If that does not work, it constructs a name "sqlite3_X_init" where the ** X is consists of the lower-case equivalent of all ASCII alphabetic ** characters in the filename from the last "/" to the first following ** ". " and omitting any initial "lib".^ *** ^The sqlite3_load_extension() interface returns ** [SQLITE_OK] on success and [SQLITE_ERROR] if something goes wrong. ** ^If an error occurs and pzErrMsg is not 0, then the ** [sqlite3_load_extension()] interface shall attempt to ** fill *pzErrMsg with error message text stored in memory ** obtained from [sqlite3_malloc()]. The calling function ** should free this memory by calling [sqlite3_free()]. *** ^Extension loading must be enabled using ** [sqlite3_enable_load_extension()] or ** [sqlite3_db_config](db,[SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION],1,NULL) ** prior to calling this API, ** otherwise an error will be returned. *** Security warning: It is recommended that the ** [SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION] method be used to enable only this ** interface. The use of the [sqlite3_enable_load_extension()] interface ** should be avoided. This will keep the SQL function [load_extension()] ** disabled and prevent SQL injections from giving attackers ** access to extension loading capabilities. *** See also the [load_extension()] SQL function].

80830. ** CAPI3REF: Name Of A Host Parameter ** METHOD: sqlite3_stmt *** ^The sqlite3_bind_parameter_name(P,N) interface returns ** the name of the N-th [SQL parameter] in the [prepared statement] P. ** ^*(SQL parameters of the form "?NNN" or ":AAA" or "@AAA" or "\$AAA" ** have a name which is the string "?NNN" or ":AAA" or "@AAA" or "\$AAA" ** respectively. In other words, the initial ":" or "\$" or "@" or "?" ** is included as part of the name.)* ^ *** ^Parameters of the form "?" without a following integer have no name ** and are referred to as "nameless" or "anonymous parameters". *** ^The first host parameter has an index of 1, not 0. ** ^*If the value N is out of range or if the N-th parameter is ** nameless, then NULL is returned. The returned string is ** always in UTF-8 encoding even if the named parameter was ** originally specified as UTF-16 in [sqlite3_prepare16()], ** [sqlite3_prepare16_v2()], or [sqlite3_prepare16_v3()].* *** See also: [sqlite3_bind_blob][sqlite3_bind()], ** [sqlite3_bind_parameter_count()], and ** [sqlite3_bind_parameter_index()].

80831. OUT: Size of buffer at *ppChangeset

80832. Currently always set to 3

80833. Score of parent node

80834. First argument passed to xConflict

80835. Generic error

80836. ** CAPI3REF: Create A New Changegroup Object ** An sqlite3_changegroup object is used to combine two or more changesets ** (or patchsets) into a single changeset (or patchset). A single changegroup ** object may combine changesets or patchsets, but not both. The output is ** always in the same format as the input. ** ^*If successful, this function returns SQLITE_OK and populates (*pp) with ** a pointer to a new sqlite3_changegroup object before returning. The caller ** should eventually free the returned object using a call to ** sqlite3changegroup_delete(). If an error occurs, an SQLite error code ** (i.e. SQLITE_NOMEM) is returned and *pp is set to NULL.* *** ^*The usual usage pattern for an sqlite3_changegroup object is as follows:* *** ** It is created using a call to sqlite3changegroup_new(). ** Zero or more changesets (or patchsets) are added to the object ** by calling sqlite3changegroup_add(). ** The result of combining all input changesets together is obtained ** by the application via a call to sqlite3changegroup_output(). ** The object is deleted using a call to sqlite3changegroup_delete(). ** ** Any number of calls to add() and output() may be made between the calls to ** new() and delete(), and in any order. ** ^*As well as the regular sqlite3changegroup_add() and ** sqlite3changegroup_output() functions, also available are the streaming ** versions sqlite3changegroup_add_strm() and sqlite3changegroup_output_strm().*

80837. ** CAPI3REF: Flush caches to disk mid-transaction ** ^*If a write-transaction is open on [database connection] D when the ** [sqlite3_db_cacheflush(D)] interface invoked, any dirty ** pages in the pager-cache that are not currently in use are written out ** to disk. A dirty page may be in use if a database cursor created by an ** active SQL statement is reading from it, or if it is page 1 of a database ** file (page 1 is always "in use"). ^The [sqlite3_db_cacheflush(D)] ** interface flushes caches for all schemas - "main", "temp", and ** any [attached] databases. ** ^*If this function needs to obtain extra database locks before dirty pages ** can be flushed to disk, it does so. ^If those locks cannot be obtained ** immediately and there is a busy-handler callback configured, it is invoked ** in the usual manner. ^If the required lock still cannot be obtained, then ** the database is skipped and an attempt made to flush any dirty pages ** belonging to the next (if any) database. ^If any databases are skipped ** because locks cannot be obtained, but no other error occurs, this ** function returns SQLITE_BUSY. ** ^*If any other error occurs while flushing dirty pages to disk (for ** example an IO error or out-of-memory condition), then processing is ** abandoned and an SQLite [error code] is returned to the caller immediately. ** ^*Otherwise, if no error occurs, [sqlite3_db_cacheflush()] returns SQLITE_OK.* ** ^*This function does not set the database handle error code or message ** returned by the [sqlite3_errcode()] and [sqlite3_errmsg()] functions.****

80838. ** CAPI3REF: Authorizer Action Codes ** The [sqlite3_set_authorizer()] interface registers a callback function ** that is invoked to authorize certain SQL statement actions. The ** second parameter to the callback is an integer code that specifies ** what action is being authorized. These are the integer action codes that ** the authorizer callback may be passed. ** ^*These action code values signify what kind of operation is to be ** authorized. The 3rd and 4th parameters to the authorization ** callback function will be parameters or NULL depending on which of these ** codes is used as the second parameter. ^*(The 5th parameter to the ** authorizer callback is the name of the database ("main", "temp", ** etc.) if applicable.)* ^*The 6th parameter to the authorizer callback ** is the name of the inner-most trigger or view that is responsible for ** the access attempt or NULL if this access attempt is directly from ** top-level SQL code.**

80839. Entry point. Derived from zFile if 0

80840. Mask of SQLITE_INDEX_SCAN_* flags

80841. ** CAPI3REF: SQL Function Context Object ** The context in which an SQL function executes is stored in an ** sqlite3_context object. ^A pointer to an sqlite3_context object ** is always first parameter to [application-defined SQL functions]. ** ^*The application-defined SQL function implementation will pass this ** pointer through into calls to [sqlite3_result_int] | [sqlite3_result()], ** [sqlite3_aggregate_context()], [sqlite3_user_data()], ** [sqlite3_context_db_handle()], [sqlite3_get_auxdata()], ** and/or [sqlite3_set_auxdata()].*

80842. Copy of 2nd argument to xTokenize()

80843. Table Name NULL

80844. ** A pointer to a structure of the following type is passed as the first ** argument to callbacks registered using rtree_geometry_callback().

80845. **comment:** ** CAPI3REF: Obtain Conflicting Row Values From A Changeset Iterator ** This function should only be used with iterator objects passed to a ** conflict-handler callback by [sqlite3changeset_apply()] with either ** [SQLITE_CHANGESET_DATA] or [SQLITE_CHANGESET_CONFLICT]. If this function ** is called on any other iterator, [SQLITE_MISUSE] is returned and *ppValue ** is set to NULL. ** ^*Argument iVal must be greater than or equal to 0, and less than the number ** of columns in the table affected by the current change. Otherwise, ** [SQLITE_RANGE] is returned and *ppValue is set to NULL.* ** ^*If successful, this function sets *ppValue to point to a protected ** sqlite3_value object containing the iVal'th value from the ** "conflicting row" associated with the current conflict-handler callback ** and returns SQLITE_OK.* ** ^*If some other error occurs (e.g. an OOM condition), an SQLite error code ** is returned and *ppValue is set to NULL.*

label: code-design

80846. deprecated names

80847. **comment:** No longer used

label: code-design

80848. SQLITE_CHECKPOINT_* value

80849. ***** End of fts5.h *****

80850. OUT: Pointer to table name

80851. Put error message here if not 0

80852. OUT: Value from conflicting row

80853. VFS only

80854. ** CAPI3REF: Declared Datatype Of A Query Result ** METHOD: sqlite3_stmt *** ** ^The first parameter is a [prepared statement]. ** If this statement is a [SELECT] statement and the Nth column of the ** returned result set of that [SELECT] is a table column (not an ** expression or subquery) then the declared type of the table ** column is returned.^ If the Nth column of the result set is an ** expression or subquery, then a NULL pointer is returned. ** ^The returned string is always UTF-8 encoded. ** ** ^For example, given the database schema: ** ** CREATE TABLE t1(c1 VARIANT); ** ** and the following statement to be compiled: ** ** SELECT c1 + 1, c1 FROM t1; ** ** this routine would return the string "VARIANT" for the second result ** column (i==1), and a NULL pointer for the first result column (i==0).^ ** ** ^SQLite uses dynamic run-time typing. ^So just because a column ** is declared to contain a particular type does not mean that the ** data stored in that column is of the declared type. SQLite is ** strongly typed, but the typing is dynamic not static. ^Type ** is associated with individual values, not with the containers ** used to hold those values.

80855. For use by extension VFS

80856. **comment:** Do not code a test for this constraint

label: test

80857. ** CAPI3REF: Configure database connections ** METHOD: sqlite3 *** ** ^The sqlite3_db_config() interface is used to make configuration ** changes to a [database connection]. The interface is similar to ** [sqlite3_config()] except that the changes apply to a single ** [database connection] (specified in the first argument). ** ** ^The second argument to sqlite3_db_config(D,V,...) is the ** [SQLITE_DBCONFIG_LOOKASIDE | configuration verb] - an integer code ** that indicates what aspect of the [database connection] is being configured. ** Subsequent arguments vary depending on the configuration verb. ** ** ^Calls to sqlite3_db_config() return SQLITE_OK if and only if ** the call is considered successful.

80858. **comment:** ** CAPI3REF: Name Of The Folder Holding Database Files *** ** ^If this global variable is made to point to a string which is ** the name of a folder (a.k.a. directory), then all database files ** specified with a relative pathname and created or accessed by ** SQLite when using a built-in windows [sqlite3_vfs | VFS] will be assumed ** to be relative to that directory.^ If this variable is a NULL ** pointer, then SQLite assumes that all database files specified ** with a relative pathname are relative to the current directory ** for the process. Only the windows VFS makes use of this global ** variable; it is ignored by the unix VFS. ** ** Changing the value of this variable while a database connection is ** open can result in a corrupt database. ** ** ^It is not safe to read or modify this variable in more than one ** thread at a time. It is not safe to read or modify this variable ** if a [database connection] is being used at the same time in a separate ** thread. ** ^It is intended that this variable be set once ** as part of process initialization and before any SQLite interface ** routines have been called and that this variable remain unchanged ** thereafter. ** ** ^The [data_store_directory pragma] may modify this variable and cause ** it to point to memory obtained from [sqlite3_malloc]. ^Furthermore, ** the [data_store_directory pragma] always assumes that any string ** that this variable points to is held in memory obtained from ** [sqlite3_malloc] and the pragma may attempt to free that memory ** using [sqlite3_free]. ** Hence, if this variable is modified directly, either it should be ** made NULL or made to point to memory obtained from [sqlite3_malloc] ** or else the use of the [data_store_directory pragma] should be avoided.

label: code-design

80859. **comment:** ** CAPI3REF: Memory Allocation Subsystem *** ** ^The SQLite core uses these three routines for all of its own ** internal memory allocation needs. "Core" in the previous sentence ** does not include operating-system specific VFS implementation. The ** Windows VFS uses native malloc() and free() for some operations. ** ** ^The sqlite3_malloc() routine returns a pointer to a block ** of memory at least N bytes in length, where N is the parameter. ** ^If sqlite3_malloc() is unable to obtain sufficient free ** memory, it returns a NULL pointer. ^If the parameter N to ** sqlite3_malloc() is zero or negative then sqlite3_malloc() returns ** a NULL pointer. ** ** ^The sqlite3_malloc64(N) routine works just like ** sqlite3_malloc(N) except that N is an unsigned 64-bit integer instead ** of a signed 32-bit integer. ** ** ^Calling sqlite3_free() with a pointer previously returned ** by sqlite3_malloc() or sqlite3_realloc() releases that memory so ** that it might be reused. ^The sqlite3_free() routine is ** a no-op if it is called with a NULL pointer. Passing a NULL pointer ** to sqlite3_free() is harmless. After being freed, memory ** should neither be read nor written. Even reading previously freed ** memory might result in a segmentation fault or other severe error. ** ^Memory corruption, a segmentation fault, or other severe error ** might result if sqlite3_free() is called with a non-NULL pointer that ** was not obtained from sqlite3_malloc() or sqlite3_realloc(). ** ** ^The sqlite3_realloc(X,N) interface attempts to resize a ** prior memory allocation X to be at least N bytes. ** ^If the X parameter to sqlite3_realloc(X,N) ** is a NULL pointer then its behavior is identical to calling ** sqlite3_malloc(N). ** ^If the N parameter to sqlite3_realloc(X,N) is zero or ** negative then the behavior is exactly the same as calling ** sqlite3_free(X). ** ^sqlite3_realloc(X,N) returns a pointer to a memory allocation ** of at least N bytes in size or NULL if insufficient memory is available. ** ^If M is the size of the prior allocation, then min(N,M) bytes ** of the prior allocation are copied into the beginning of buffer returned ** by sqlite3_realloc(X,N) and the prior allocation is freed. ** ^If sqlite3_realloc(X,N) returns NULL and N is positive, then the ** prior allocation is not freed. ** ** ^The sqlite3_realloc64(X,N) interfaces works the same as ** sqlite3_realloc(X,N) except that N is a 64-bit unsigned integer instead ** of a 32-bit signed integer. ** ** ^If X is a memory allocation previously obtained from sqlite3_malloc(), ** sqlite3_malloc64(), sqlite3_realloc(), or sqlite3_realloc64(), then ** sqlite3_msize(X) returns the size of that memory allocation in bytes. ** ^The value returned by sqlite3_msize(X) might be larger than the number ** of bytes requested when X was allocated. ^If X is a NULL pointer then ** sqlite3_msize(X) returns zero. If X points to something that is not ** the beginning of memory allocation, or if it points to a formerly ** valid memory allocation that has now been freed, then the behavior ** of sqlite3_msize(X) is undefined and possibly harmful. ** ** ^The memory returned by sqlite3_malloc(), sqlite3_realloc(), ** sqlite3_malloc64(), and sqlite3_realloc64() ** is always aligned to at least an 8 byte boundary, or to a ** 4 byte boundary if the [SQLITE_4_BYTE_ALIGNED_MALLOC] compile-time ** option is used. ** ** ^In SQLite version 3.5.0 and 3.5.1, it was possible to define ** the SQLITE OMIT_MEMORY_ALLOCATION which would cause the built-in ** implementation of these routines to be omitted. That capability ** is no longer provided. Only built-in memory allocators can be used. ** ** Prior to SQLite version 3.7.10, the Windows OS interface layer called ** the system malloc() and free() directly when converting ** filenames between the UTF-8 encoding used by SQLite ** and whatever filename encoding is used by the particular Windows ** installation. Memory allocation errors were detected, but ** they were reported back as [SQLITE_CANTOPEN] or ** [SQLITE_IOERR] rather than [SQLITE_NOMEM]. ** ** ^The pointer arguments to [sqlite3_free()] and [sqlite3_realloc()] ** must be either NULL or else pointers obtained from a prior ** invocation of [sqlite3_malloc()] or [sqlite3_realloc()] that have ** not yet been released. ** ** ^The application must not read or write any part of ** a block of memory after it has been released using ** [sqlite3_free()] or [sqlite3_realloc()].

label: code-design

80860. For use by application VFS

80861. Size of array aParam[]

80862. beginning-of-error-codes

80863. ** CAPI3REF: Reset A Prepared Statement Object ** METHOD: sqlite3_stmt *** ** ^The sqlite3_reset() function is called to reset a [prepared statement] ** object back to its initial state, ready to be re-executed. ** ^Any SQL statement variables that had values bound to them using ** the [sqlite3_bind_blob | sqlite3_bind_*] API retain their values. ** Use [sqlite3_clear_bindings()] to reset the bindings. ** ** ^The [sqlite3_reset(S)] interface resets the [prepared statement] S ** back to the beginning of its program. ** ** ^If the most recent call to [sqlite3_step(S)] for the ** [prepared statement] S returned [SQLITE_ROW] or [SQLITE_DONE], ** or if [sqlite3_step(S)] has never before been called on S, ** then [sqlite3_reset(S)] returns [SQLITE_OK]. ** ** ^If the most recent call to [sqlite3_step(S)] for the ** [prepared statement] S indicated an error, then ** [sqlite3_reset(S)] returns an appropriate [error code]. ** ** ^The [sqlite3_reset(S)] interface does not change the values ** of any [sqlite3_bind_blob|bindings] on the [prepared statement] S.

80864. ** CAPI3REF: Move a BLOB Handle to a New Row ** METHOD: sqlite3_blob *** ** ^This function is used to move an existing [BLOB handle] so that it points ** to a different row of the same database table. ^The new row is identified ** by the rowid value passed as the second argument. Only the row can be ** changed. ^The database, table and column on which the blob handle is open ** remain the same. Moving an existing [BLOB handle] to a new row is ** faster than closing the existing handle and opening a new one. ** ** ^The new row must meet the same criteria as for [sqlite3_blob_open()] - ** it must exist and there must be either a blob or text value stored in ** the nominated column.^ If the new row is not present in the table, or if ** it does not contain a blob or text value, or if another error occurs, an ** SQLite error code is returned and the blob handle is considered aborted. ** ^All subsequent calls to [sqlite3_blob_read()], [sqlite3_blob_write()] or ** [sqlite3_blob_reopen()] on an aborted blob handle immediately return ** SQLITE_ABORT. ^Calling [sqlite3_blob_bytes()] on an aborted blob handle ** always returns zero. ** ** ^This function sets the database handle error code and message.

80865. Iterator object

80866. OUT: Buffer containing output changeset

80867. Pointer to blob containing changeset
80868. ** 2010 August 30 *** The author disclaims copyright to this source code. In place of ** a legal notice, here is a blessing: *** May you do good and not evil.
** May you find forgiveness for yourself and forgive others. ** May you share freely, never taking more than you give. **

80869. ** CAPI3REF: Reset Automatic Extension Loading ** ^This interface disables all automatic extensions previously ** registered using [sqlite3_auto_extension()].
80870. Name of VFS module to use
80871. ** CAPI3REF: String Globbing * ** ^The [sqlite3_strglob(P,X)] interface returns zero if and only if ** string X matches the [GLOB] pattern P. ** ^The definition of [GLOB] pattern matching used in ** [sqlite3_strglob(P,X)] is the same as for the "X GLOB P" operator in the ** SQL dialect understood by SQLite. ^The [sqlite3_strglob(P,X)] function ** is case sensitive. ** ** Note that this routine returns zero on a match and non-zero if the strings ** do not match, the same as [sqlite3_stricmp()] and [sqlite3_strnicmp()]. ** ** See also: [sqlite3_strlike()].
80872. ** CAPI3REF: Custom Page Cache Object ** ** ^The sqlite3_pcache_page object represents a single page in the ** page cache. The page cache will allocate instances of this ** object. Various methods of the page cache use pointers to instances ** of this object as parameters or as their return value. ** ** See [sqlite3_pcache_methods2] for additional information.
80873. Number of entries in aConstraint
80874. **comment:** ** CAPI3REF: Unlock Notification ** METHOD: sqlite3 ** ** ^When running in shared-cache mode, a database operation may fail with ** an [SQLITE_LOCKED] error if the required locks on the shared-cache or ** individual tables within the shared-cache cannot be obtained. See ** [SQLite Shared-Cache Mode] for a description of shared-cache locking. ** ^This API may be used to register a callback that SQLite will invoke ** when the connection currently holding the required lock relinquishes it. ** ^This API is only available if the library was compiled with the ** [SQLITE_ENABLE_UNLOCK_NOTIFY] C-preprocessor symbol defined. ** ** See Also: [Using the SQLite Unlock Notification Feature]. ** ** ^Shared-cache locks are released when a database connection concludes ** its current transaction, either by committing it or rolling it back. ** ** ^When a connection (known as the blocked connection) fails to obtain a ** shared-cache lock and SQLITE_LOCKED is returned to the caller, the ** identity of the database connection (the blocking connection) that ** has locked the required resource is stored internally. ^After an ** application receives an SQLITE_LOCKED error, it may call the ** sqlite3_unlock_notify() method with the blocked connection handle as ** the first argument to register for a callback that will be invoked ** when the blocking connections current transaction is concluded. ^The ** callback is invoked from within the [sqlite3_step] or [sqlite3_close] ** call that concludes the blocking connections transaction. ** ** ^If sqlite3_unlock_notify() is called in a multi-threaded application, ** there is a chance that the blocking connection will have already ** concluded its transaction by the time sqlite3_unlock_notify() is invoked. ** If this happens, then the specified callback is invoked immediately, ** from within the call to sqlite3_unlock_notify().)^ ** ** ^If the blocked connection is attempting to obtain a write-lock on a ** shared-cache table, and more than one other connection currently holds ** a read-lock on the same table, then SQLite arbitrarily selects one of ** the other connections to use as the blocking connection. ** ** ^There may be at most one unlock-notify callback registered by a ** blocked connection. If sqlite3_unlock_notify() is called when the ** blocked connection already has a registered unlock-notify callback, ** then the new callback replaces the old.)^ ^If sqlite3_unlock_notify() is ** called with a NULL pointer as its second argument, then any existing ** unlock-notify callback is canceled. ^The blocked connections ** unlock-notify callback may also be canceled by closing the blocked ** connection using [sqlite3_close()]. ** ** ^The unlock-notify callback is not reentrant. If an application invokes ** any sqlite3_xxx API functions from within an unlock-notify callback, a ** crash or deadlock may be the result. ** ** ^Unless deadlock is detected (see below), sqlite3_unlock_notify() always ** returns SQLITE_OK. ** ** Callback Invocation Details ** ** When an unlock-notify callback is registered, the application provides a ** single void* pointer that is passed to the callback when it is invoked. ** However, the signature of the callback function allows SQLite to pass ** it an array of void* context pointers. The first argument passed to ** an unlock-notify callback is a pointer to an array of void* pointers, ** and the second is the number of entries in the array. ** ** When a blocking connections transaction is concluded, there may be ** more than one blocked connection that has registered for an unlock-notify ** callback. ^If two or more such blocked connections have specified the ** same callback function, then instead of invoking the callback function ** multiple times, it is invoked once with the set of void* context pointers ** specified by the blocked connections bundled together into an array. ** This gives the application an opportunity to prioritize any actions ** related to the set of unblocked database connections. ** ** Deadlock Detection ** ** Assuming that after registering for an unlock-notify callback a ** database waits for the callback to be issued before taking any further ** action (a reasonable assumption), then using this API may cause the ** application to deadlock. For example, if connection X is waiting for ** connection Y's transaction to be concluded, and similarly connection ** Y is waiting on connection X's transaction, then neither connection ** will proceed and the system may remain deadlocked indefinitely. ** ** To avoid this scenario, the sqlite3_unlock_notify() performs deadlock ** detection. ^If a given call to sqlite3_unlock_notify() would put the ** system in a deadlocked state, then SQLITE_LOCKED is returned and no ** unlock-notify callback is registered. The system is said to be in ** a deadlocked state if connection A has registered for an unlock-notify ** callback on the conclusion of connection B's transaction, and connection ** B has itself registered for an unlock-notify callback when connection ** A's transaction is concluded. ^Indirect deadlock is also detected, so ** the system is also considered to be deadlocked if connection B has ** registered for an unlock-notify callback on the conclusion of connection ** C's transaction, where connection C is waiting on connection A. ^Any ** number of levels of indirection are allowed. ** ** The "DROP TABLE" Exception ** ** When a call to [sqlite3_step()] returns SQLITE_LOCKED, it is almost ** always appropriate to call sqlite3_unlock_notify(). There is however, ** one exception. When executing a "DROP TABLE" or "DROP INDEX" statement, ** SQLite checks if there are any currently executing SELECT statements ** that belong to the same connection. If there are, SQLITE_LOCKED is ** returned. In this case there is no "blocking connection", so invoking ** sqlite3_unlock_notify() results in the unlock-notify callback being ** invoked immediately. If the application then re-attempts the "DROP TABLE" ** or "DROP INDEX" query, an infinite loop might be the result. ** ** One way around this problem is to check the extended error code returned ** by an sqlite3_step() call. ^If there is a blocking connection, then the ** extended error code is set to SQLITE_LOCKED_SHAREDCACHE. Otherwise, in ** the special "DROP TABLE/INDEX" case, the extended error code is just ** SQLITE_LOCKED.)^
label: code-design
80875. Fields below are only available in SQLite 3.8.2 and later
80876. **comment:** OUT: Pointer to unused portion of zSql
label: code-design
80877. Abort due to constraint violation
80878. Library used incorrectly
80879. OUT: New value (or NULL pointer)
80880. const char*
80881. ** CAPI3REF: Generate A Changeset From A Session Object ** ** Obtain a changeset containing changes to the tables attached to the ** session object passed as the first argument. If successful, ** set *ppChangeset to point to a buffer containing the changeset ** and *pnChangeset to the size of the changeset in bytes before returning ** SQLITE_OK. If an error occurs, set both *ppChangeset and *pnChangeset to ** zero and return an SQLite error code. ** ** A changeset consists of zero or more INSERT, UPDATE and/or DELETE changes, ** each representing a change to a single row of an attached table. An INSERT ** change contains the values of each field of a new database row. A DELETE ** contains the original values of each field of a deleted database row. An ** UPDATE change contains the original values of each field of an updated ** database row along with the updated values for each updated non-primary-key ** column. It is not possible for an UPDATE change to represent a change that ** modifies the values of primary key columns. If such a change is made, it ** is represented in a changeset as a DELETE followed by an INSERT. ** ** Changes are not recorded for rows that have NULL values stored in one or ** more of their PRIMARY KEY columns. If such a row is inserted or deleted, ** no corresponding change is present in the changesets returned by this ** function. If an existing row with one or more NULL values stored in ** PRIMARY KEY columns is updated so that all PRIMARY KEY columns are non-NULL, ** only an INSERT is appears in the changeset. Similarly, if an existing row ** with non-NULL PRIMARY KEY values is updated so that one or more of its ** PRIMARY KEY columns are set to NULL, the resulting changeset contains a ** DELETE change only. ** ** The contents of a changeset may be traversed using an iterator created ** using the [sqlite3changeset_start()] API. A changeset may be applied to ** a database with a compatible schema using the [sqlite3changeset_apply()] ** API. ** ** Within a changeset generated by this function, all changes related to a ** single table are grouped together. In other words, when iterating through ** a changeset or when applying a changeset to a database, all changes related ** to a single table are processed before moving on to the next table. Tables ** are sorted in the same order in which they were attached (or auto-attached) ** to the sqlite3_session object. The order in which the changes related to ** a single table are stored is undefined. ** ** Following a successful call to this function, it is the responsibility of ** the caller to eventually free the buffer that *ppChangeset points to using ** [sqlite3_free()]. ** ** <h3>Changeset Generation</h3> ** ** Once a table has been attached to a session object, the session object ** records the primary key values of all new rows inserted into the table. ** It also records the original primary key and other column values of any ** deleted or updated rows. For each unique primary key value, data is only ** recorded once - the first time a row with said primary key is inserted, ** updated or deleted in the lifetime of the session. ** ** There is one exception to the previous paragraph: when a row is inserted, ** updated or deleted, if one or more of its primary key columns contain a ** NULL value, no record of the change is made. ** ** The session object therefore accumulates two types of records - those ** that consist of primary key values only (created when the user inserts ** a new record) and those that consist of the primary key values and the ** original values of other table columns (created

when the user deletes ** or updates a record). *** When this function is called, the requested changeset is created using ** both the accumulated records and the current contents of the database ** file. Specifically: *** ** For each record generated by an insert, the database is queried ** for a row with a matching primary key. If one is found, an INSERT ** change is added to the changeset. If no such row is found, no change ** is added to the changeset. *** For each record generated by an update or delete, the database is ** queried for a row with a matching primary key. If such a row is ** found and one or more of the non-primary key fields have been ** modified from their original values, an UPDATE change is added to ** the changeset. Or, if no such row is found in the table, a DELETE ** change is added to the changeset. If there is a row with a matching ** primary key in the database, but all fields contain their original ** values, no change is added to the changeset. ** *** This means, amongst other things, that if a row is inserted and then later ** deleted while a session object is active, neither the insert nor the delete ** will be present in the changeset. Or if a row is deleted and then later a ** row with the same primary key values inserted while a session object is ** active, the resulting changeset will contain an UPDATE change instead of ** a DELETE and an INSERT. *** When a session object is disabled (see the [sqlite3session_enable()] API), ** it does not accumulate records when rows are inserted, updated or deleted. ** This may appear to have some counter-intuitive effects if a single row ** is written to more than once during a session. For example, if a row ** is inserted while a session object is enabled, then later deleted while ** the same session object is disabled, no INSERT record will appear in the ** changeset, even though the delete took place while the session was disabled. ** Or, if one field of a row is updated while a session is disabled, and ** another field of the same row is updated while the session is enabled, the ** resulting changeset will contain an UPDATE change that updates both fields.

80882. Index Name NULL

80883. Copy of pContext passed to s_r_g_c()

80884. Callback function

80885. Number of bytes in buffer pA

80886. Like RESTART but also truncate WAL

80887. ***** 3rd ***** 4th *****

80888. Database name or NULL

80889. ** CAPI3REF: Start a read transaction on an historical snapshot ** EXPERIMENTAL ** ^The [sqlite3_snapshot_open(D,S,P)] interface starts a ** read transaction for schema S of ** [database connection] D such that the read transaction ** refers to historical [snapshot] P, rather than the most ** recent change to the database. ** ^The [sqlite3_snapshot_open()] interface returns SQLITE_OK on success ** or an appropriate [error code] if it fails. *** ^In order to succeed, a call to [sqlite3_snapshot_open(D,S,P)] must be ** the first operation following the [BEGIN] that takes the schema S ** out of [autocommit mode]. ** ^In other words, schema S must not currently be in ** a transaction for [sqlite3_snapshot_open(D,S,P)] to work, but the ** database connection D must be out of [autocommit mode]. ** ^A [snapshot] will fail to open if it has been overwritten by a ** [checkpoint]. ** ^A call to [sqlite3_snapshot_open(D,S,P)] will fail if the ** database connection D does not know that the database file for ** schema S is in [WAL mode]. A database connection might not know ** that the database file is in [WAL mode] if there has been no prior ** I/O on that database connection, or if the database entered [WAL mode] ** after the most recent I/O on the database connection.)** (Hint: Run "[PRAGMA application_id]" against a newly opened ** database connection in order to make it ready to use snapshots.) *** The [sqlite3_snapshot_open()] interface is only available when the ** SQLITE_ENABLE_SNAPSHOT compile-time option is used.

80890. Copy of sixth arg to _apply()

80891. ** CAPI3REF: Set a table filter on a Session Object. ** The second argument (xFilter) is the "filter callback". For changes to rows ** in tables that are not attached to the Session object, the filter is called ** to determine whether changes to the table's rows should be tracked or not. ** If xFilter returns 0, changes is not tracked. Note that once a table is ** attached, xFilter will not be called again.

80892. ** CAPI3REF: Virtual Table Indexing Information ** KEYWORDS: sqlite3_index_info *** The sqlite3_index_info structure and its substructures is used as part ** of the [virtual table] interface to ** pass information into and receive the reply from the [xBestIndex] ** method of a [virtual table module]. The fields under **Inputs** are the ** inputs to xBestIndex and are read-only. xBestIndex inserts its ** results into the **Outputs** fields. *** ^The aConstraint[] array records WHERE clause constraints of the form: ** <blockquote>column OP expr</blockquote> ** where OP is =, <, <=, >, or >=.)^ ** ^The particular operator is ** stored in aConstraint[.].op using one of the ** [SQLITE_INDEX_CONSTRAINT_EQ | SQLITE_INDEX_CONSTRAINT_values].)^ ** ^The index of the column is stored in ** aConstraint[.].iColumn.^ ^((aConstraint[.].usable is TRUE if the ** expr on the right-hand side can be evaluated (and thus the constraint ** is usable) and false if it cannot.)^ ** ^The optimizer automatically inverts terms of the form "expr OP column" ** and makes other simplifications to the WHERE clause in an attempt to ** get as many WHERE clause terms into the form shown above as possible. ** ^The aConstraint[] array only reports WHERE clause terms that are ** relevant to the particular virtual table being queried. *** ^Information about the ORDER BY clause is stored in aOrderBy[]. ** ^Each term of aOrderBy records a column of the ORDER BY clause. ** ^The colUsed field indicates which columns of the virtual table may be ** required by the current scan. Virtual table columns are numbered from ** zero in the order in which they appear within the CREATE TABLE statement ** passed to sqlite3_declare_vtab(). For the first 63 columns (columns 0-62), ** the corresponding bit is set within the colUsed mask if the column may be ** required by SQLite. If the table has at least 64 columns and any column ** to the right of the first 63 is required, then bit 63 of colUsed is also ** set. In other words, column iCol may be required if the expression ** (colUsed & ((sqlite3_uint64)1 << (iCol>=63 ? 63 : iCol))) evaluates to ** non-zero. ** ^The [xBestIndex] method must fill aConstraintUsage[] with information ** about what parameters to pass to xFilter. ^If argvIndex>0 then ** the right-hand side of the corresponding aConstraint[] is evaluated ** and becomes the argvIndex-th entry in argv. ^((If aConstraintUsage[.].omit ** is true, then the constraint is assumed to be fully handled by the ** virtual table and is not checked again by SQLite.)^ ** ^The idxNum and idxPtr values are recorded and passed into the ** [xFilter] method. ** ^[sqlite3_free()] is used to free idxPrt if and only if ** needToFreeIdxPrt is true. ** ^The orderByConsumed means that output from [xFilter]/[xNext] will occur in ** the correct order to satisfy the ORDER BY clause so that no separate ** sorting step is required. *** ^The estimatedCost value is an estimate of the cost of a particular ** strategy. A cost of N indicates that the cost of the strategy is similar ** to a linear scan of an SQLite table with N rows. A cost of log(N) ** indicates that the expense of the operation is similar to that of a ** binary search on a unique indexed field of an SQLite table with N rows. ** ^The estimatedRows value is an estimate of the number of rows that ** will be returned by the strategy. *** ^The xBestIndex method may optionally populate the idxFlags field with a ** mask of SQLITE_INDEX_SCAN_* flags. Currently there is only one such flag - ** SQLITE_INDEX_SCAN_UNIQUE. If the xBestIndex method sets this flag, SQLite ** assumes that the strategy may visit at most one row. ** ^Additionally, if xBestIndex sets the SQLITE_INDEX_SCAN_UNIQUE flag, then ** SQLite also assumes that if a call to the xUpdate() method is made as ** part of the same statement to delete or update a virtual table row and the ** implementation returns SQLITE_CONSTRAINT, then there is no need to rollback ** any database changes. In other words, if the xUpdate() returns ** SQLITE_CONSTRAINT, the database contents must be exactly as they were ** before xUpdate was called. By contrast, if SQLITE_INDEX_SCAN_UNIQUE is not ** set and xUpdate returns SQLITE_CONSTRAINT, any database changes made by ** the xUpdate method are automatically rolled back by SQLite. ** ^IMPORTANT: The estimatedRows field was added to the sqlite3_index_info ** structure for SQLite [version 3.8.2] ([dateof:3.8.2]). ** If a virtual table extension is ** used with an SQLite version earlier than 3.8.2, the results of attempting ** to read or write the estimatedRows field are undefined (but are likely ** to included crashing the application). The estimatedRows field should ** therefore only be used if [sqlite3_libversion_number()] returns a ** value greater than or equal to 3008002. Similarly, the idxFlags field ** was added for [version 3.9.0] ([dateof:3.9.0]). ** It may therefore only be used if ** sqlite3_libversion_number() returns a value greater than or equal to ** 3009000.

80893. IMP: R-03371-37637

80894. Callback routine requested an abort

80895. OUTPUT: Collation sequence name

80896. Size of subclassed sqlite3_file

80897. Source database handle

80898. Changeset iterator

80899. OUT: Size of output buffer in bytes

80900. ** The methods above are in versions 1 through 3 of the sqlite vfs object. ** New fields may be appended in future versions. The iVersion ** value will increment whenever this happens.

80901. sqlite3_step() has another row ready

80902. Rowid for current entry

80903. nil

80904. Apply change to "main" db of this handle

80905. The new key

80906. Number of values in apVal[] array

80907. **comment:** ** CAPI3REF: One-Step Query Execution Interface ** METHOD: sqlite3 ** ** The sqlite3_exec() interface is a convenience wrapper around ** [sqlite3_prepare_v2()], [sqlite3_step()], and [sqlite3_finalize()], ** that allows an application to run multiple statements of SQL ** without having to use a lot of C code. *** ^The sqlite3_exec() interface runs zero or more UTF-8 encoded, ** semicolon-separate SQL statements passed into its 2nd argument, ** in the context of the [database connection] passed in as its 1st ** argument. ^If the callback function of the 3rd argument to ** sqlite3_exec() is not NULL, then it is invoked for each result row ** coming out of the evaluated SQL statements. ^The 4th argument to ** sqlite3_exec() is relayed through to the 1st argument of each ** callback

invocation. ^If the callback pointer to sqlite3_exec() ** is NULL, then no callback is ever invoked and result rows are ** ignored. *** ^If an error occurs while evaluating the SQL statements passed into ** sqlite3_exec(), then execution of the current statement stops and ** subsequent statements are skipped. ^If the 5th parameter to sqlite3_exec() ** is not NULL then any error message is written into memory obtained ** from [sqlite3_malloc()] and passed back through the 5th parameter. ** To avoid memory leaks, the application should invoke [sqlite3_free()] ** on error message strings returned through the 5th parameter of ** sqlite3_exec() after the error message string is no longer needed. ** ^If the 5th parameter to sqlite3_exec() is not NULL and no errors ** occur, then sqlite3_exec() sets the pointer in its 5th parameter to ** NULL before returning. *** ^If an sqlite3_exec() callback returns non-zero, the sqlite3_exec() ** routine returns SQLITE_ABORT without invoking the callback again and ** without running any subsequent SQL statements. *** ^The 2nd argument to the sqlite3_exec() callback function is the ** number of columns in the result. ^The 3rd argument to the sqlite3_exec() ** callback is an array of pointers to strings obtained as if from ** [sqlite3_column_text()], one for each column. ^If an element of a ** result row is NULL then the corresponding string pointer for the ** sqlite3_exec() callback is a NULL pointer. ^The 4th argument to the ** sqlite3_exec() callback is an array of pointers to strings where each ** entry represents the name of corresponding result column as obtained ** from [sqlite3_column_name()]. *** ^If the 2nd parameter to sqlite3_exec() is a NULL pointer, a pointer ** to an empty string, or a pointer that contains only whitespace and/or ** SQL comments, then no SQL statements are evaluated and the database ** is not changed. *** ^Restrictions: *** ** The application must ensure that the 1st parameter to sqlite3_exec() ** is a valid and open [database connection]. ** The application must not close the [database connection] specified by ** the 1st parameter to sqlite3_exec() while sqlite3_exec() is running. ** The application must not modify the SQL statement text passed into ** the 2nd parameter of sqlite3_exec() while sqlite3_exec() is running. **

label: code-design

80908. ** CAPI3REF: Function Flags *** These constants may be ORed together with the ** [SQLITE_UTF8 | preferred text encoding] as the fourth argument ** to [sqlite3_create_function()], [sqlite3_create_function16()], or ** [sqlite3_create_function_v2()].

80909. Size of token in bytes

80910. Used by PRAGMA temp_store_directory

80911. Object fully contained within query region

80912. ** CAPI3REF: SQL Trace Event Codes *** KEYWORDS: SQLITE_TRACE *** These constants identify classes of events that can be monitored ** using the [sqlite3_trace_v2()] tracing logic. The third argument ** to [sqlite3_trace_v2()] is an OR-ed combination of one or more of ** the following constants. ^The first argument to the trace callback ** is one of the following constants. *** ^New tracing constants may be added in future releases. *** ^A trace callback has four arguments: xCallback(T,C,P,X). ** ^The T argument is one of the integer type codes above. ** ^The C argument is a copy of the context pointer passed in as the ** fourth argument to [sqlite3_trace_v2()]. ** ^The P and X arguments are pointers whose meanings depend on T. *** <dl> ** [[SQLITE_TRACE_STMT]] <dt>SQLITE_TRACE_STMT</dt> ** <dd>^An SQLITE_TRACE_STMT callback is invoked when a prepared statement ** first begins running and possibly at other times during the ** execution of the prepared statement, such as at the start of each ** trigger subprogram. ^The P argument is a pointer to the ** [prepared statement]. ^The X argument is a pointer to a string which ** is the unexpanded SQL text of the prepared statement or an SQL comment ** that indicates the invocation of a trigger. ^The callback can compute ** the same text that would have been returned by the legacy [sqlite3_trace()] ** interface by using the X argument when X begins with "--" and invoking ** [sqlite3_expanded_sql(P)] otherwise. *** <dd>[[SQLITE_TRACE_PROFILE]] <dt>SQLITE_TRACE_PROFILE</dt> ** <dd>^An SQLITE_TRACE_PROFILE callback provides approximately the same ** information as is provided by the [sqlite3_profile()] callback. ** ^The P argument is a pointer to the [prepared statement] and the ** X argument points to a 64-bit integer which is the estimated of ** the number of nanosecond that the prepared statement took to run. ** ^The SQLITE_TRACE_PROFILE callback is invoked when the statement finishes. *** <dd>[[SQLITE_TRACE_ROW]] <dt>SQLITE_TRACE_ROW</dt> ** <dd>^An SQLITE_TRACE_ROW callback is invoked whenever a prepared ** statement generates a single row of result. ** ^The P argument is a pointer to the [prepared statement] and the ** X argument is unused. *** <dd>[[SQLITE_TRACE_CLOSE]] <dt>SQLITE_TRACE_CLOSE</dt> ** <dd>^An SQLITE_TRACE_CLOSE callback is invoked when a database ** connection closes. ** ^The P argument is a pointer to the [database connection] object ** and the X argument is unused. ** </dl>

80913. **comment:** ** CAPI3REF: Application Defined Page Cache. *** KEYWORDS: {page cache} *** ^The [sqlite3_config()][SQLITE_CONFIG_PCACHE2, ...] interface can ** register an alternative page cache implementation by passing in an ** instance of the sqlite3_pcache_methods2 structure. ^In many applications, most of the heap memory allocated by ** SQLite is used for the page cache. ** By implementing a ** custom page cache using this API, an application can better control ** the amount of memory consumed by SQLite, the way in which ** that memory is allocated and released, and the policies used to ** determine exactly which parts of a database file are cached and for ** how long. *** ^The alternative page cache mechanism is an ** extreme measure that is only needed by the most demanding applications. ** The built-in page cache is recommended for most uses. *** ^The contents of the sqlite3_pcache_methods2 structure are copied to an ** internal buffer by SQLite within the call to [sqlite3_config]. Hence ** the application may discard the parameter after the call to ** [sqlite3_config()] returns.)^ *** <dd>[[the xInit() page cache method]] ** ^The xInit() method is called once for each effective ** call to [sqlite3_initialize()]^ ** (usually only once during the lifetime of the process). ^The xInit() ** method is passed a copy of the sqlite3_pcache_methods2.pArg value.)^ ** The intent of the xInit() method is to set up global data structures ** required by the custom page cache implementation. ** ^If the xInit() method is NULL, then the ** built-in default page cache is used instead of the application defined ** page cache.)^ *** <dd>[[the xShutdown() page cache method]] ** ^The xShutdown() method is called by [sqlite3_shutdown()]. ** It can be used to clean up ** any outstanding resources before process shutdown, if required. ** ^The xShutdown() method may be NULL. *** ^SQLite automatically serializes calls to the xInit method, ** so the xInit method need not be threadsafe. ^The ** xShutdown method is only called from [sqlite3_shutdown()] so it does ** not need to be threadsafe either. All other methods must be threadsafe ** in multithreaded applications. *** ^SQLite will never invoke xInit() more than once without an intervening ** call to xShutdown(). *** <dd>[[the xCreate() page cache methods]] ** ^SQLite invokes the xCreate() method to construct a new cache instance. ** SQLite will typically create one cache instance for each open database file, ** though this is not guaranteed. ^The ** first parameter, szPage, is the size in bytes of the pages that must ** be allocated by the cache. ^szPage will always a power of two. ^The ** second parameter szExtra is a number of bytes of extra storage ** associated with each page cache entry. ^The szExtra parameter will ** a number less than 250. SQLite will use the ** extra szExtra bytes on each page to store metadata about the underlying ** database page on disk. The value passed into szExtra depends ** on the SQLite version, the target platform, and how SQLite was compiled. ** ^The third argument to xCreate(), bPurgeable, is true if the cache being ** created will be used to cache database pages of a file stored on disk, or ** false if it is used for an in-memory database. The cache implementation ** does not have to do anything special based with the value of bPurgeable; ** it is purely advisory. ^On a cache where bPurgeable is false, SQLite will ** never invoke xUnpin() except to deliberately delete a page. ** ^In other words, calls to xUnpin() on a cache with bPurgeable set to ** false will always have the "discard" flag set to true. ** ^Hence, a cache created with bPurgeable false will ** never contain any unpinned pages. *** <dd>[[the xCachesize() page cache method]] ** ^The xCachesize() method may be called at any time by SQLite to set the ** suggested maximum cache-size (number of pages stored by) the cache ** instance passed as the first argument. This is the value configured using ** the SQLite "[PRAGMA cache_size]" command.)^ As with the bPurgeable ** parameter, the implementation is not required to do anything with this ** value; it is advisory only. *** <dd>[[the xPagecount() page cache methods]] ** The xPagecount() method must return the number of pages currently ** stored in the cache, both pinned and unpinned. *** <dd>[[the xFetch() page cache methods]] ** The xFetch() method locates a page in the cache and returns a pointer to ** an sqlite3_pcache_page object associated with that page, or a NULL pointer. ** The pBuf element of the returned sqlite3_pcache_page object will be a ** pointer to a buffer of szPage bytes used to store the content of a ** single database page. The pExtra element of sqlite3_pcache_page will be ** a pointer to the szExtra bytes of extra storage that SQLite has requested ** for each entry in the page cache. *** ^The page to be fetched is determined by the key. ^The minimum key value ** is 1. After it has been retrieved using xFetch, the page is considered ** to be "pinned". *** ^If the requested page is already in the page cache, then the page cache ** implementation must return a pointer to the page buffer with its content ** intact. If the requested page is not already in the cache, then the ** cache implementation should use the value of the createFlag ** parameter to help it determined what action to take: *** <table border=1 width=85% align=center> ** <tr><th> createFlag </th> Behavior when page is not already in cache ** <tr><td> 0 </td> Do not allocate a new page. Return NULL. ** <tr><td> 1 </td> Allocate a new page if it easy and convenient to do so. ** Otherwise return NULL. ** <tr><td> 2 </td> Make every effort to allocate a new page. Only return ** NULL if allocating a new page is effectively impossible. ** </table> *** ^SQLite will normally invoke xFetch() with a createFlag of 0 or 1. SQLite ** will only use a createFlag of 2 after a prior call with a createFlag of 1 ** failed.)^ In between the to xFetch() calls, SQLite may ** attempt to unpin one or more cache pages by spilling the content of ** pinned pages to disk and synching the operating system disk cache. *** <dd>[[the xUnpin() page cache method]] ** ^xUnpin() is called by SQLite with a pointer to a currently pinned page ** as its second argument. If the third parameter, discard, is non-zero, ** then the page must be evicted from the cache. ** ^If the discard parameter is ** zero, then the page may be discarded or retained at the discretion of ** page cache implementation. ^The page cache implementation ** may choose to evict unpinned pages at any time. *** ^The cache must not perform any reference counting. A single ** call to xUnpin() unpins the page regardless of the number of prior calls ** to xFetch(). *** <dd>[[the xRekey() page cache methods]] ** The xRekey() method is used to change the key value associated with the ** page passed as the second argument. If the cache ** previously contains an entry associated with newKey, it must be ** discarded. ^Any prior cache entry associated with newKey is guaranteed not ** to be pinned. *** ^When SQLite calls the xTruncate() method, the cache must discard all ** existing cache entries with page numbers (keys) greater than or equal ** to the value of the iLimit parameter passed to xTruncate(). If any ** of these pages are pinned, they are implicitly unpinned, meaning that ** they can be safely discarded. *** <dd>[[the xDestroy() page cache method]] ** ^The xDestroy() method is used to delete a cache allocated by xCreate(). ** All resources associated with the specified cache should be freed. ^After ** calling the xDestroy() method, SQLite considers the [sqlite3_pcache*] ** handle invalid, and will not use it with any other sqlite3_pcache_methods2 **

functions. *** [[the xShrink() page cache method]] ** ^SQLite invokes the xShrink() method when it wants the page cache to ** free up as much of heap memory as possible. The page cache implementation ** is not obligated to free any memory, but well-behaved implementations should ** do their best.

label: code-design

80914. void*, int nByte, int min

80915. ** CAPI3REF: Database Snapshot ** KEYWORDS: {snapshot} {sqlite3_snapshot} ** EXPERIMENTAL ** ** An instance of the snapshot object records the state of a [WAL mode] ** database for some specific point in history. *** In [WAL mode], multiple [database connections] that are open on the ** same database file can each be reading a different historical version ** of the database file. When a [database connection] begins a read ** transaction, that connection sees an unchanging copy of the database ** as it existed for the point in time when the transaction first started. ** Subsequent changes to the database from other connections are not seen ** by the reader until a new read transaction is started. *** The sqlite3_snapshot object records state information about an historical ** version of the database file so that it is possible to later open a new read ** transaction that sees that historical version of the database rather than ** the most recent version. *** The constructor for this object is [sqlite3_snapshot_get()]. The ** [sqlite3_snapshot_open()] method causes a fresh read transaction to refer ** to an historical snapshot (if possible). The destructor for ** sqlite3_snapshot objects is [sqlite3_snapshot_free()].

80916. **comment:** ** CAPI3REF: Function Auxiliary Data ** METHOD: sqlite3_context ** ** These functions may be used by (non-aggregate) SQL functions to ** associate metadata with argument values. If the same value is passed to ** multiple invocations of the same SQL function during query execution, under ** some circumstances the associated metadata may be preserved. An example ** of where this might be useful is in a regular-expression matching ** function. The compiled version of the regular expression can be stored as ** metadata associated with the pattern string. ** Then as long as the pattern string remains the same, ** the compiled regular expression can be reused on multiple ** invocations of the same function. *** ^The sqlite3_get_auxdata(C,N) interface returns a pointer to the metadata ** associated by the sqlite3_set_auxdata(C,N,P,X) function with the Nth argument ** value to the application-defined function. ^N is zero for the left-most ** function argument. ^If there is no metadata ** associated with the function argument, the sqlite3_get_auxdata(C,N) interface ** returns a NULL pointer. *** ^The sqlite3_set_auxdata(C,N,P,X) interface saves P as metadata for the N-th ** argument of the application-defined function. ^Subsequent ** calls to sqlite3_get_auxdata(C,N) return P from the most recent ** sqlite3_set_auxdata(C,N,P,X) call if the metadata is still valid or ** NULL if the metadata has been discarded. ** ^After each call to sqlite3_set_auxdata(C,N,P,X) where X is not NULL, ** SQLite will invoke the destructor function X with parameter P exactly ** once, when the metadata is discarded. ** SQLite is free to discard the metadata at any time, including: ** ^when the corresponding function parameter changes)^, or ** ^when [sqlite3_reset()] or [sqlite3_finalize()] is called for the ** SQL statement)^, or ** ^when sqlite3_set_auxdata() is invoked again on the same ** parameter)^, or ** ^during the original sqlite3_set_auxdata() call when a memory ** allocation error occurs.^ *** Note the last bullet in particular. The destructor X in ** sqlite3_set_auxdata(C,N,P,X) might be called immediately, before the ** sqlite3_set_auxdata() interface even returns. Hence sqlite3_set_auxdata() ** should be called near the end of the function implementation and the ** function implementation should not make any use of P after ** sqlite3_set_auxdata() has been called. *** ^In practice, metadata is preserved between function calls for ** function parameters that are compile-time constants, including literal ** values and [parameters] and expressions composed from the same.)^ *** ^The value of the N parameter to these interfaces should be non-negative. ** Future enhancements may make use of negative N values to define new ** kinds of function caching behavior. *** ^These routines must be called from the same thread in which ** the SQL function is running.

label: code-design

80917. ** CAPI3REF: Change group Handle

80918. OUT: Pointer to output buffer

80919. Waiting connection

80920. ** CAPI3REF: Result Codes ** KEYWORDS: {result code definitions} ** ** Many SQLite functions return an integer result code from the set shown ** here in order to indicate success or failure. *** ^New error codes may be added in future versions of SQLite. *** ^See also: [extended result code definitions]

80921. OUT: SQLITE_INSERT, DELETE or UPDATE

80922. ** CAPI3REF: Run-Time Library Compilation Options Diagnostics ** ** ^The sqlite3_compileoption_used() function returns 0 or 1 ** indicating whether the specified option was defined at ** compile time. ^The SQLITE_ prefix may be omitted from the ** option name passed to sqlite3_compileoption_used(). *** ^The sqlite3_compileoption_get() function allows iterating ** over the list of options that were defined at compile time by ** returning the N-th compile time option string. ^If N is out of range, ** sqlite3_compileoption_get() returns a NULL pointer. ^The SQLITE_ ** prefix is omitted from any strings returned by ** sqlite3_compileoption_get(). *** ^Support for the diagnostic functions sqlite3_compileoption_used() ** and sqlite3_compileoption_get() may be omitted by specifying the ** [SQLITE OMIT COMPILEOPTION_DIAGS] option at compile time. *** ^See also: SQL functions [sqlite_compileoption_used()] and ** [sqlite_compileoption_get()] and the [compile_options pragma].

80923. ** CAPI3REF: Recover snapshots from a wal file ** EXPERIMENTAL ** ** If all connections disconnect from a database file but do not perform ** a checkpoint, the existing wal file is opened along with the database ** file the next time the database is opened. At this point it is only ** possible to successfully call sqlite3_snapshot_open() to open the most ** recent snapshot of the database (the one at the head of the wal file), ** even though the wal file may contain other valid snapshots for which ** clients have sqlite3_snapshot handles. *** This function attempts to scan the wal file associated with database zDb ** of database handle db and make all valid snapshots available to ** sqlite3_snapshot_open(). It is an error if there is already a read ** transaction open on the database, or if the database is not a wal mode ** database. *** ^SQLITE_OK is returned if successful, or an SQLite error code otherwise.

80924. ** CAPI3REF: Index Of A Parameter With A Given Name ** METHOD: sqlite3_stmt ** ** ^Return the index of an SQL parameter given its name. ^The ** index value returned is suitable for use as the second ** parameter to [sqlite3_bind_blob|sqlite3_bind()]. ^A zero ** is returned if no matching parameter is found. ^The parameter ** name must be given in UTF-8 even if the original statement ** was prepared from UTF-16 text using [sqlite3_prepare16_v2()] or ** [sqlite3_prepare16_v3()]. *** ^See also: [sqlite3_bind_blob|sqlite3_bind()], ** [sqlite3_bind_parameter_count()], and ** [sqlite3_bind_parameter_name()].

80925. **comment:** ** CAPI3REF: Online Backup API. ** ** The backup API copies the content of one database into another. ** It is useful either for creating backups of databases or ** for copying in-memory databases to or from persistent files. *** ^See Also: [Using the SQLite Online Backup API] ** ** ^SQLite holds a write transaction open on the destination database file ** for the duration of the backup operation. ** ^The source database is read-locked only while it is being read; ** it is not locked continuously for the entire backup operation. ** ^Thus, the backup may be performed on a live source database without ** preventing other database connections from ** reading or writing to the source database while the backup is underway. *** ^To perform a backup operation: ** ** sqlite3_backup_init(is called once to initialize the ** backup, ** sqlite3_backup_step(is called one or more times to transfer ** the data between the two databases, and finally ** sqlite3_backup_finish(is called to release all resources ** associated with the backup operation. ** ^ ** There should be exactly one call to sqlite3_backup_finish() for each ** successful call to sqlite3_backup_init(). *** ^[[sqlite3_backup_init()]] sqlite3_backup_init(** ** ^The D and N arguments to sqlite3_backup_init(D,N,S,M) are the ** [database connection] associated with the destination database ** and the database name, respectively. ** ^The database name is "main" for the main database, "temp" for the ** temporary database, or the name specified after the AS keyword in ** an [ATTACH] statement for an attached database. ** ^The S and M arguments passed to ** sqlite3_backup_init(D,N,S,M) identify the [database connection] ** and database name of the source database, respectively. ** ^The source and destination [database connections] (parameters S and D) ** must be different or else sqlite3_backup_init(D,N,S,M) will fail with ** an error. *** ^A call to sqlite3_backup_init() will fail, returning NULL, if ** there is already a read or read-write transaction open on the ** destination database. *** ^If an error occurs within sqlite3_backup_init(D,N,S,M), then NULL is ** returned and an error code and error message are stored in the ** destination [database connection] D. ** ^The error code and message for the failed call to sqlite3_backup_init() ** can be retrieved using the [sqlite3_errcode()], [sqlite3_errmsg()], and/or ** [sqlite3_errmsg16()] functions. ** ^A successful call to sqlite3_backup_init() returns a pointer to an ** [sqlite3_backup] object. ** ^The [sqlite3_backup] object may be used with the sqlite3_backup_step() and ** sqlite3_backup_finish() functions to perform the specified backup ** operation. *** ^[[sqlite3_backup_step()]] sqlite3_backup_step(** ** ^Function sqlite3_backup_step(B,N) will copy up to N pages between ** the source and destination databases specified by [sqlite3_backup] object B. ** ^If N is negative, all remaining source pages are copied. ** ^If sqlite3_backup_step(B,N) successfully copies N pages and there ** are still more pages to be copied, then the function returns [SQLITE_OK]. ** ^If sqlite3_backup_step(B,N) successfully finishes copying all pages ** from source to destination, then it returns [SQLITE_DONE]. ** ^If an error occurs while running sqlite3_backup_step(B,N), ** then an [error code] is returned. ^As well as [SQLITE_OK] and ** [SQLITE_DONE], a call to sqlite3_backup_step() may return [SQLITE_READONLY], ** [SQLITE_NOMEM], [SQLITE_BUSY], [SQLITE_LOCKED], or an ** [SQLITE_IOERR_ACCESS | SQLITE_IOERR_XXX] extended error code. *** ^The sqlite3_backup_step() might return [SQLITE_READONLY] if ** ** the destination database was opened read-only, or ** the destination database is using write-ahead-log journaling ** and the destination and source page sizes differ, or ** the destination database is an in-memory database and the ** destination and source page sizes differ. ** ^ ** ^If sqlite3_backup_step() cannot obtain a required file-system lock, then ** the [sqlite3_busy_handler | busy-handler function] ** is invoked (if one is specified). ^If the ** busy-handler returns non-zero before the lock is available, then ** [SQLITE_BUSY] is returned to the caller. ^In this case the call to ** sqlite3_backup_step() can be retried later. ^If the source ** [database connection] ** is being used to write to the source database when sqlite3_backup_step() ** is called, then [SQLITE_LOCKED] is returned immediately. ^Again, in this ** case the call to sqlite3_backup_step() can be retried later on. ^If ** [SQLITE_IOERR_ACCESS | SQLITE_IOERR_XXX], [SQLITE_NOMEM], or ** [SQLITE_READONLY] is returned, then ** there is no point in retrying the call to sqlite3_backup_step(). These ** errors are considered fatal.)^ The application must accept ** that the backup operation has failed and pass the backup operation handle ** to the sqlite3_backup_finish() to release associated resources. *** ^The first call to sqlite3_backup_step() obtains an exclusive lock ** on

the destination file. ^The exclusive lock is not released until either ** sqlite3_backup_finish() is called or the backup operation is complete ** and sqlite3_backup_step() returns [SQLITE_DONE]. ^Every call to ** sqlite3_backup_step() obtains a [shared lock] on the source database that ** lasts for the duration of the sqlite3_backup_step() call. ** ^Because the source database is not locked between calls to ** sqlite3_backup_step(), the source database may be modified mid-way ** through the backup process. ^If the source database is modified by an ** external process or via a database connection other than the one being ** used by the backup operation, then the backup will be automatically ** restarted by the next call to sqlite3_backup_step(). ^If the source ** database is modified by the using the same database connection as is used ** by the backup operation, then the backup database is automatically ** updated at the same time. *** [[sqlite3_backup_finish()]] sqlite3_backup_finish() *** When sqlite3_backup_step() has returned [SQLITE_DONE], or when the ** application wishes to abandon the backup operation, the application ** should destroy the [sqlite3_backup] by passing it to sqlite3_backup_finish(). ** ^The sqlite3_backup_finish() interfaces releases all ** resources associated with the [sqlite3_backup] object. ** ^If sqlite3_backup_step() has not yet returned [SQLITE_DONE], then any ** active write-transaction on the destination database is rolled back. ** The [sqlite3_backup] object is invalid ** and may not be used following a call to sqlite3_backup_finish(). *** ^The value returned by sqlite3_backup_finish is [SQLITE_OK] if no ** sqlite3_backup_step() errors occurred, regardless or whether or not ** sqlite3_backup_step() completed. ** ^If an out-of-memory condition or IO error occurred during any prior ** sqlite3_backup_step() call on the same [sqlite3_backup] object, then ** sqlite3_backup_finish() returns the corresponding [error code]. *** ^A return of [SQLITE_BUSY] or [SQLITE_LOCKED] from sqlite3_backup_step() ** is not a permanent error and does not affect the return value of ** sqlite3_backup_finish(). *** [[sqlite3_backup_remaining()]] [[sqlite3_backup_pagecount()]] ** sqlite3_backup_remaining() and sqlite3_backup_pagecount() *** ^The sqlite3_backup_remaining() routine returns the number of pages still ** to be backed up at the conclusion of the most recent sqlite3_backup_step(). ** ^The sqlite3_backup_pagecount() routine returns the total number of pages ** in the source database at the conclusion of the most recent ** sqlite3_backup_step(). ** ^The values returned by these functions are only updated by ** sqlite3_backup_step(). If the source database is modified in a way that ** changes the size of the source database or the number of pages remaining, ** those changes are not reflected in the output of sqlite3_backup_pagecount() ** and sqlite3_backup_remaining() until after the next ** sqlite3_backup_step().)*** Concurrent Usage of Database Handles *** ^The source [database connection] may be used by the application for other ** purposes while a backup operation is underway or being initialized. ** ^If SQLite is compiled and configured to support threadsafe database ** connections, then the source database connection may be used concurrently ** from within other threads. *** ^However, the application must guarantee that the destination ** [database connection] is not passed to any other API (by any thread) after ** sqlite3_backup_init() is called and before the corresponding call to ** sqlite3_backup_finish(). SQLite does not currently check to see ** if the application incorrectly accesses the destination [database connection] ** and so no error code is reported, but the operations may malfunction ** nevertheless. Use of the destination database connection while a ** backup is in progress might also cause a mutex deadlock. *** ^If running in [shared cache mode], the application must ** guarantee that the shared cache used by the destination database ** is not accessed while the backup is running. In practice this means ** that the application must guarantee that the disk file being ** backed up to is not accessed by any connection within the process, ** not just the specific connection that was passed to sqlite3_backup_init(). *** ^The [sqlite3_backup] object itself is partially threadsafe. Multiple ** threads may safely make multiple concurrent calls to sqlite3_backup_step(). ** However, the sqlite3_backup_remaining() and sqlite3_backup_pagecount() ** APIs are not strictly speaking threadsafe. If they are invoked at the ** same time as another thread is invoking sqlite3_backup_step() it is ** possible that they return invalid values.

label: code-design

80926. ** Register a geometry callback named zGeom that can be used as part of an ** R-Tree geometry query as follows: *** ** SELECT ... FROM <rtree> WHERE <rtree col> MATCH \$zGeom(... params ...) 80927. ** CAPI3REF: Zero Scan-Status Counters ** METHOD: sqlite3_stmt ** ** ^Zero all [sqlite3_stmt_scanstatus()] related event counters. *** This API is only available if the library is built with pre-processor ** symbol [SQLITE_ENABLE_STMT_SCANSTATUS] defined. 80928. ** CAPI3REF: Testing Interface ** ** ^The sqlite3_test_control() interface is used to read out internal ** state of SQLite and to inject faults into SQLite for testing ** purposes. ^The first parameter is an operation code that determines ** the number, meaning, and operation of all subsequent parameters. *** This interface is not for use by applications. It exists solely ** for verifying the correct operation of the SQLite library. Depending ** on how the SQLite library is compiled, this interface might not exist. *** ^The details of the operation codes, their meanings, the parameters ** they take, and what they do are all subject to change without notice. ** Unlike most of the SQLite API, this function is not guaranteed to ** operate consistently from one release to the next.

80929. Database Name Table Name

80930. Internal logic error in SQLite

80931. Pointer to buffer containing changeset B

80932. Additional methods may be added in future releases

80933. First argument passed to xFilter

80934. **comment:** ** CAPI3REF: Mutex Methods Object ** ** An instance of this structure defines the low-level routines ** used to allocate and use mutexes. *** Usually, the default mutex implementations provided by SQLite are ** sufficient, however the application has the option of substituting a custom ** implementation for specialized deployments or systems for which SQLite ** does not provide a suitable implementation. In this case, the application ** creates and populates an instance of this structure to pass ** to sqlite3_config() along with the [SQLITE_CONFIG_MUTEX] option. ** Additionally, an instance of this structure can be used as an ** output variable when querying the system for the current mutex ** implementation, using the [SQLITE_CONFIG_GETMUTEX] option. *** ^The xMutexInit method defined by this structure is invoked as ** part of system initialization by the sqlite3_initialize() function. ** ^The xMutexEnd routine is called by SQLite exactly once for each ** effective call to [sqlite3_initialize()]. *** ^The xMutexEnd method defined by this structure is invoked as ** part of system shutdown by the sqlite3_shutdown() function. The ** implementation of this method is expected to release all outstanding ** resources obtained by the mutex methods implementation, especially ** those obtained by the xMutexInit method. ^The xMutexEnd() ** interface is invoked exactly once for each call to [sqlite3_shutdown()]. *** ^The remaining seven methods defined by this structure (xMutexAlloc, ** xMutexFree, xMutexEnter, xMutexTry, xMutexLeave, xMutexHeld and ** xMutexNoheld) implement the following interfaces (respectively): *** ** [sqlite3_mutex_alloc()] ** [sqlite3_mutex_free()] ** [sqlite3_mutex_enter()] ** [sqlite3_mutex_try()] ** [sqlite3_mutex_leave()] ** [sqlite3_mutex_held()] ** [sqlite3_mutex_noheld()] ** *** ^The only difference is that the public sqlite3_XXX functions enumerated ** above silently ignore any invocations that pass a NULL pointer instead ** of a valid mutex handle. The implementations of the methods defined ** by this structure are not required to handle this case, the results ** of passing a NULL pointer instead of a valid mutex handle are undefined ** (i.e. it is acceptable to provide an implementation that segfaults if ** it is passed a NULL pointer). *** ^The xMutexInit() method must be threadsafe. It must be harmless to ** invoke xMutexInit() multiple times within the same process and without ** intervening calls to xMutexEnd(). Second and subsequent calls to ** xMutexInit() must be no-ops. *** ^xMutexInit() must not use SQLite memory allocation ([sqlite3_malloc()] ** and its associates). Similarly, xMutexAlloc() must not use SQLite memory ** allocation for a static mutex. ^However xMutexAlloc() may use SQLite ** memory allocation for a fast or recursive mutex. *** ^SQLite will invoke the xMutexEnd() method when [sqlite3_shutdown()] is ** called, but only if the prior call to xMutexInit returned SQLITE_OK. ** If xMutexInit fails in any way, it is expected to clean up after itself ** prior to returning.

label: code-design

80935. ** CAPI3REF: Concatenate Two Changeset Objects ** ** This function is used to concatenate two changesets, A and B, into a ** single changeset. The result is a changeset equivalent to applying ** changeset A followed by changeset B. *** This function combines the two input changesets using an ** sqlite3_changegroup object. Calling it produces similar results as the ** following code fragment: *** ** sqlite3_changegroup *pGrp; ** rc = sqlite3_changegroup_new(&pGrp); ** if(rc==SQLITE_OK) rc = sqlite3changegroup_add(pGrp, nA, pA); ** if(rc==SQLITE_OK) rc = sqlite3changegroup_add(pGrp, nB, pB); ** if(rc==SQLITE_OK){ ** rc = sqlite3changegroup_output(pGrp, pnOut, ppOut); ** }else{ ** ppOut = 0; ** *pnOut = 0; ** } *** Refer to the sqlite3_changegroup documentation below for details.

80936. ** CAPI3REF: Checkpoint a database ** METHOD: sqlite3 ** ** ^The sqlite3_wal_checkpoint(D,X) is equivalent to ** [sqlite3_wal_checkpoint_v2](D,X, [SQLITE_CHECKPOINT_PASSIVE],0,0).)*** ^In brief, sqlite3_wal_checkpoint(D,X) causes the content in the ** [write-ahead log] for database X on [database connection] D to be ** transferred into the database file and for the write-ahead log to ** be reset. See the [checkpointing] documentation for addition ** information. *** ^This interface used to be the only way to cause a checkpoint to ** occur. But then the newer and more powerful [sqlite3_wal_checkpoint_v2] ** interface was added. This interface is retained for backwards ** compatibility and as a convenience for applications that need to manually ** start a callback but which do not need the full power (and corresponding ** complication) of [sqlite3_wal_checkpoint_v2].

80937. ***** End of sqlite3session.h *****

80938. ** CAPI3REF: Set the Last Insert Rowid value. ** METHOD: sqlite3 ** ** The sqlite3_set_last_insert_rowid(D, R) method allows the application to ** set the value returned by calling sqlite3_last_insert_rowid(D) to R ** without inserting a row into the database.

80939. ** CAPI3REF: Mutex Verification Routines ** ** The sqlite3_mutex_held() and sqlite3_mutex_noheld() routines ** are intended for use inside assert() statements. The SQLite core ** never uses these routines except inside an assert() and applications ** are advised to follow the lead of the core. The SQLite core only ** provides implementations for these routines when it is compiled ** with the SQLITE_DEBUG flag. External mutex implementations ** are only required to provide these routines if SQLITE_DEBUG is ** defined and if NDEBUG is not defined. *** ^These routines should return true if the mutex in their argument ** is held or not held, respectively, by the calling thread. *** ^The implementation is not required to provide versions of these ** routines that actually work. If

the implementation does not provide working ** versions of these routines, it should at least provide stubs that always ** return true so that one does not get spurious assertion failures. *** If the argument to sqlite3_mutex_held() is a NULL pointer then ** the routine should return 1. This seems counter-intuitive since ** clearly the mutex cannot be held if it does not exist. But ** the reason the mutex does not exist is because the build is not ** using mutexes. And we do not want the assert() containing the ** call to sqlite3_mutex_held() to fail, so a non-zero return is ** the appropriate thing to do. The sqlite3_mutex_notheld() ** interface should also return 1 when given a NULL pointer.

80940. ** Make sure we can call this stuff from C++.

80941. ** CAPI3REF: Flags for the xShmLock VFS method *** These integer constants define the various locking operations ** allowed by the xShmLock method of [sqlite3_io_methods]. The ** following are the only legal combinations of flags to the ** xShmLock method: *** ** SQLITE_SHM_LOCK | SQLITE_SHM_SHARED ** SQLITE_SHM_LOCK | SQLITE_SHM_EXCLUSIVE ** SQLITE_SHM_UNLOCK | SQLITE_SHM_SHARED ** SQLITE_SHM_UNLOCK | SQLITE_SHM_EXCLUSIVE ** *** When unlocking, the same SHARED or EXCLUSIVE flag must be supplied as ** was given on the corresponding lock. *** The xShmLock method can transition between unlocked and SHARED or ** between unlocked and EXCLUSIVE. It cannot transition between SHARED ** and EXCLUSIVE.

80942. Deprecated

80943. ** CAPI3REF: String Comparison *** ^The [sqlite3_stricmp()] and [sqlite3_strnicmp()] APIs allow applications ** and extensions to compare the contents of two buffers containing UTF-8 ** strings in a case-independent fashion, using the same definition of "case ** independence" that SQLite uses internally when comparing identifiers.

80944. Needed for the definition of va_list

80945. OUT: Old value (or NULL pointer)

80946. sqlite3PageMalloc()

80947. ** CAPI3REF: Enable Or Disable Extended Result Codes ** METHOD: sqlite3 *** ^The sqlite3_extended_result_codes() routine enables or disables the ** [extended result codes] feature of SQLite. ^The extended result ** codes are disabled by default for historical compatibility.

80948. SQLITE_UPDATE, DELETE or INSERT

80949. **comment:** ** CAPI3REF: Flags for the xAccess VFS method *** These integer constants can be used as the third parameter to ** the xAccess method of an [sqlite3_vfs] object. They determine ** what kind of permissions the xAccess method is looking for. ** With SQLITE_ACCESS_EXISTS, the xAccess method ** simply checks whether the file exists. ** With SQLITE_ACCESS_READWRITE, the xAccess method ** checks whether the named directory is both readable and writable ** (in other words, if files can be added, removed, and renamed within ** the directory). ** The SQLITE_ACCESS_READWRITE constant is currently used only by the ** [temp_store_directory pragma], though this could change in a future ** release of SQLite. ** With SQLITE_ACCESS_READ, the xAccess method ** checks whether the file is readable. The SQLITE_ACCESS_READ constant is ** currently unused, though it might be used in a future release of ** SQLite.

label: code-design

80950. ** Ensure these symbols were not defined by some previous header file.

80951. Scan visits at most 1 row

80952. For use by built-in VFS

80953. New rowid value (for a rowid UPDATE)

80954. ** CAPI3REF: Last Insert Rowid ** METHOD: sqlite3 *** ^Each entry in most SQLite tables (except for [WITHOUT ROWID] tables) ** has a unique 64-bit signed ** integer key called the [ROWID | "rowid"]. ^The rowid is always available ** as an undeclared column named ROWID, OID, or _ROWID_ as long as those ** names are not also used by explicitly declared columns. ^If ** the table has a column of type [INTEGER PRIMARY KEY] then that column ** is another alias for the rowid. *** ^The sqlite3_last_insert_rowid(D) interface usually returns the [rowid] of ** the most recent successful [INSERT] into a rowid table or [virtual table] ** on database connection D. ^Inserts into [WITHOUT ROWID] tables are not ** recorded. ^If no successful [INSERT]s into rowid tables have ever occurred ** on the database connection D, then sqlite3_last_insert_rowid(D) returns ** zero. *** As well as being set automatically as rows are inserted into database ** tables, the value returned by this function may be set explicitly by ** [sqlite3_set_last_insert_rowid()] ** ** Some virtual table implementations may INSERT rows into rowid tables as ** part of committing a transaction (e.g. to flush data accumulated in memory ** to disk). In this case subsequent calls to this function return the rowid ** associated with these internal INSERT operations, which leads to ** unintuitive results. Virtual table implementations that do write to rowid ** tables in this way can avoid this problem by restoring the original ** rowid value using [sqlite3_set_last_insert_rowid()] before returning ** control to the user. *** ^If an [INSERT] occurs within a trigger then this routine will ** return the [rowid] of the inserted row as long as the trigger is ** running. Once the trigger program ends, the value returned ** by this routine reverts to what it was before the trigger was fired. ^** ** ^An [INSERT] that fails due to a constraint violation is not a ** successful [INSERT] and does not change the value returned by this ** routine. ^Thus INSERT OR FAIL, INSERT OR IGNORE, INSERT OR ROLLBACK, ** and INSERT OR ABORT make no changes to the return value of this ** routine when their insertion fails. ^When INSERT OR REPLACE ** encounters a constraint violation, it does not fail. The ** INSERT continues to completion after deleting rows that caused ** the constraint problem so INSERT OR REPLACE will always change ** the return value of this interface.)^** ** ^For the purposes of this routine, an [INSERT] is considered to ** be successful even if it is subsequently rolled back. *** This function is accessible to SQL statements via the ** [last_insert_rowid()] SQL function]. *** If a separate thread performs a new [INSERT] on the same ** database connection while the [sqlite3_last_insert_rowid()] ** function is running and thus changes the last insert [rowid], ** then the value returned by [sqlite3_last_insert_rowid()] is ** unpredictable and might not equal either the old or the new ** last insert [rowid].

80955. SQL to be evaluated

80956. ** CAPI3REF: Virtual Table Scan Flags

80957. ** CAPI3REF: Testing Interface Operation Codes *** These constants are the valid operation code parameters used ** as the first argument to [sqlite3_test_control()]. *** These parameters and their meanings are subject to change ** without notice. These values are for testing purposes only. ** Applications should not use any of these parameters or the ** [sqlite3_test_control()] interface.

80958. ** CAPI3REF: Determine The Number Of Foreign Key Constraint Violations *** This function may only be called with an iterator passed to an ** SQLITE_CHANGESET_FOREIGN_KEY conflict handler callback. In this case ** it sets the output variable to the total number of known foreign key ** violations in the destination database and returns SQLITE_OK. ** ** In all other cases this function returns SQLITE_MISUSE.

80959. Some kind of disk I/O error occurred

80960. ** CAPI3REF: Obtain The Primary Key Definition Of A Table *** For each modified table, a changeset includes the following: *** ** The number of columns in the table, and ** Which of those columns make up the tables PRIMARY KEY. ** *** This function is used to find which columns comprise the PRIMARY KEY of ** the table modified by the change that iterator pIter currently points to. ** If successful, *pabPK is set to point to an array of nCol entries, where ** nCol is the number of columns in the table. Elements of *pabPK are set to ** 0x01 if the corresponding column is part of the tables primary key, or ** 0x00 if it is not. *** If argument pnCol is not NULL, then *pnCol is set to the number of columns ** in the table. *** If this function is called when the iterator does not point to a valid ** entry, SQLITE_MISUSE is returned and the output variables zeroed. Otherwise, ** SQLITE_OK is returned and the output variables populated as described ** above.

80961. Maximum length of zSql in bytes.

80962. Largest defined DBSTATUS

80963. ** CAPI3REF: Custom Page Cache Object *** The sqlite3_pcachec type is opaque. It is implemented by ** the pluggable module. The SQLite core has no knowledge of ** its size or internal structure and never deals with the ** sqlite3_pcachec object except by holding and passing pointers ** to the object. *** See [sqlite3_pcachec_methods2] for additional information.

80964. ***** Begin file fts5.h *****

80965. First arg to pass to pApi functions

80966. **comment:** ** CAPI3REF: Mutexes *** The SQLite core uses these routines for thread ** synchronization. Though they are intended for internal ** use by SQLite, code that links against SQLite is ** permitted to use any of these routines. ** ** The SQLite source code contains multiple implementations ** of these mutex routines. An appropriate implementation ** is selected automatically at compile-time. The following ** implementations are available in the SQLite core: *** ** SQLITE_MUTEX_PTHREADS ** SQLITE_MUTEX_W32 ** SQLITE_MUTEX_NOOP ** *** The SQLITE_MUTEX_NOOP implementation is a set of routines ** that does no real locking and is appropriate for use in ** a single-threaded application. The SQLITE_MUTEX_PTHREADS and ** SQLITE_MUTEX_W32 implementations are appropriate for use on Unix ** and Windows. ** ** If SQLite is compiled with the SQLITE_MUTEX_APPDEF preprocessor ** macro defined (with "-DSQLITE_MUTEX_APPDEF=1"), then no mutex ** implementation is included with the library. In this case the ** application must supply a custom mutex implementation using the ** [SQLITE_CONFIG_MUTEX] option of the sqlite3_config() function ** before calling sqlite3_initialize() or any other public sqlite3_ ** function that calls sqlite3_initialize(). ** ** ^The sqlite3_mutex_alloc() routine allocates a new ** mutex and returns a pointer to it. ^The sqlite3_mutex_alloc() ** routine returns NULL if it is unable to allocate the requested ** mutex. The argument to sqlite3_mutex_alloc() must one of these ** integer constants: *** ** SQLITE_MUTEX_FAST **

SQLITE_MUTEX_RECURSIVE ** SQLITE_MUTEX_STATIC_MASTER ** SQLITE_MUTEX_STATIC_MEM ** SQLITE_MUTEX_STATIC_OPEN ** SQLITE_MUTEX_STATIC_PRNG ** SQLITE_MUTEX_STATIC_LRU ** SQLITE_MUTEX_STATIC_PMEM ** SQLITE_MUTEX_STATIC_APP1 ** SQLITE_MUTEX_STATIC_APP2 ** SQLITE_MUTEX_STATIC_APP3 ** SQLITE_MUTEX_STATIC_VFS1 ** SQLITE_MUTEX_STATIC_VFS2 ** SQLITE_MUTEX_STATIC_VFS3 ** *** ^The first two constants (SQLITE_MUTEX_FAST and SQLITE_MUTEX_RECURSIVE) ** cause sqlite3_mutex_alloc() to create ** a new mutex. ^The new mutex is recursive when SQLITE_MUTEX_RECURSIVE ** is used but not necessarily so when SQLITE_MUTEX_FAST is used. ** The mutex implementation does not need to make a distinction ** between SQLITE_MUTEX_RECURSIVE and SQLITE_MUTEX_FAST if it does ** not want to. SQLite will only request a recursive mutex in ** cases where it really needs one. If a faster non-recursive mutex ** implementation is available on the host platform, the mutex subsystem ** might return such a mutex in response to SQLITE_MUTEX_FAST. ** ** ^The other allowed parameters to sqlite3_mutex_alloc() (anything other ** than SQLITE_MUTEX_FAST and SQLITE_MUTEX_RECURSIVE) each return ** a pointer to a static preexisting mutex. ^Nine static mutexes are ** used by the current version of SQLite. Future versions of SQLite ** may add additional static mutexes. Static mutexes are for internal ** use by SQLite only. Applications that use SQLite mutexes should ** use only the dynamic mutexes returned by SQLITE_MUTEX_FAST or ** SQLITE_MUTEX_RECURSIVE. ** ** ^Note that if one of the dynamic mutex parameters (SQLITE_MUTEX_FAST ** or SQLITE_MUTEX_RECURSIVE) is used then sqlite3_mutex_alloc() ** returns a different mutex on every call. ^For the static ** mutex types, the same mutex is returned on every call that has ** the same type number. ** ** ^The sqlite3_mutex_free() routine deallocates a previously ** allocated dynamic mutex. Attempting to deallocate a static ** mutex results in undefined behavior. ** ** ^The sqlite3_mutex_enter() and sqlite3_mutex_try() routines attempt ** to enter a mutex. ^If another thread is already within the mutex, ** sqlite3_mutex_enter() will block and sqlite3_mutex_try() will return ** SQLITE_BUSY. ^The sqlite3_mutex_try() interface returns [SQLITE_OK] ** upon successful entry. ^(Mutexes created using ** SQLITE_MUTEX_RECURSIVE can be entered multiple times by the same thread. ** In such cases, the ** mutex must be exited an equal number of times before another thread ** can enter.)^ If the same thread tries to enter any mutex other ** than an SQLITE_MUTEX_RECURSIVE more than once, the behavior is undefined. ** ** ^Some systems (for example, Windows 95) do not support the operation ** implemented by sqlite3_mutex_try(). On those systems, sqlite3_mutex_try() ** will always return SQLITE_BUSY. The SQLite core only ever uses ** sqlite3_mutex_try() as an optimization so this is acceptable ** behavior.)** ** ^The sqlite3_mutex_leave() routine exits a mutex that was ** previously entered by the same thread. The behavior ** is undefined if the mutex is not currently entered by the ** calling thread or is not currently allocated. ** ** ^If the argument to sqlite3_mutex_enter(), sqlite3_mutex_try(), or ** sqlite3_mutex_leave() is a NULL pointer, then all three routines ** behave as no-ops. ** ** See also: [sqlite3_mutex_held()] and [sqlite3_mutex_notheld()].

label: code-design

80967. Like FULL but wait for readers

80968. OUT: Number of FK violations

80969. Name of the module

80970. Flags that may be passed as the third argument to xTokenize()

80971. SQLite connection to register module with

80972. void* int int

80973. The following fields are only available in 3.8.11 and later

80974. ** CAPI3REF: Interrupt A Long-Running Query ** METHOD: sqlite3 *** ^This function causes any pending database operation to abort and ** return at its earliest opportunity. This routine is typically ** called in response to a user action such as pressing "Cancel" ** or Ctrl-C where the user wants a long query operation to halt ** immediately. ** ** ^It is safe to call this routine from a thread different from the ** thread that is currently running the database operation. But it ** is not safe to call this routine with a [database connection] that ** is closed or might close before sqlite3_interrupt() returns. ** ** ^If an SQL operation is very nearly finished at the time when ** sqlite3_interrupt() is called, then it might not have an opportunity ** to be interrupted and might continue to completion. ** ** ^An SQL operation that is interrupted will return [SQLITE_INTERRUPT]. ** ^If the interrupted SQL operation is an INSERT, UPDATE, or DELETE ** that is inside an explicit transaction, then the entire transaction ** will be rolled back automatically. ** ** ^The sqlite3_interrupt(D) call is in effect until all currently running ** SQL statements on [database connection] D complete. ^Any new SQL statements ** that are started after the sqlite3_interrupt() call and before the ** running statements reaches zero are interrupted as if they had been ** running prior to the sqlite3_interrupt() call. ^New SQL statements ** that are started after the running statement count reaches zero are ** not effected by the sqlite3_interrupt(). ** ^A call to sqlite3_interrupt(D) that occurs when there are no running ** SQL statements is a no-op and has no effect on SQL statements ** that are started after the sqlite3_interrupt() call returns.

80975. comment: NOT USED

label: code-design

80976. comment: ** CAPI3REF: Advance A Changeset Iterator ** ** This function may only be used with iterators created by function ** [sqlite3changeset_start()]. If it is called on an iterator passed to ** a conflict-handler callback by [sqlite3changeset_apply()], SQLITE_MISUSE ** is returned and the call has no effect. ** ** Immediately after an iterator is created by sqlite3changeset_start(), it ** does not point to any change in the changeset. Assuming the changeset ** is not empty, the first call to this function advances the iterator to ** point to the first change in the changeset. Each subsequent call advances ** the iterator to point to the next change in the changeset (if any). If ** no error occurs and the iterator points to a valid change after a call ** to sqlite3changeset_next() has advanced it, SQLITE_ROW is returned. ** Otherwise, if all changes in the changeset have already been visited, ** SQLITE_DONE is returned. ** ** If an error occurs, an SQLite error code is returned. Possible error ** codes include SQLITE_CORRUPT (if the changeset buffer is corrupt) or ** SQLITE_NOMEM.

label: code-design

80977. ** CAPI3REF: Compile-Time Authorization Callbacks ** METHOD: sqlite3 ** KEYWORDS: {authorizer callback} ** ** ^This routine registers an authorizer callback with a particular ** [database connection], supplied in the first argument. ** ^The authorizer callback is invoked as SQL statements are being compiled ** by [sqlite3_prepare()] or its variants [sqlite3_prepare_v2()], ** [sqlite3_prepare_v3()], [sqlite3_prepare160], [sqlite3_prepare16_v2()], ** and [sqlite3_prepare16_v3()]. ^At various ** points during the compilation process, as logic is being created ** to perform various actions, the authorizer callback is invoked to ** see if those actions are allowed. ^The authorizer callback should ** return [SQLITE_OK] to allow the action, [SQLITE_IGNORE] to disallow the ** specific action but allow the SQL statement to continue to be ** compiled, or [SQLITE_DENY] to cause the entire SQL statement to be ** rejected with an error. ^If the authorizer callback returns ** any value other than [SQLITE_IGNORE], [SQLITE_OK], or [SQLITE_DENY] ** then the [sqlite3_prepare_v2()] or equivalent call that triggered ** the authorizer will fail with an error message. ** ** ^When the callback returns [SQLITE_OK], that means the operation ** requested is ok. ^When the callback returns [SQLITE_DENY], the ** [sqlite3_prepare_v2()] or equivalent call that triggered the ** authorizer will fail with an error message explaining that ** access is denied. ** ** ^The first parameter to the authorizer callback is a copy of the third ** parameter to the sqlite3_set_authorizer() interface. ^The second parameter ** to the callback is an integer [SQLITE_COPY | action code] that specifies ** the particular action to be authorized. ^The third through sixth parameters ** to the callback are either NULL pointers or zero-terminated strings ** that contain additional details about the action to be authorized. ** Applications must always be prepared to encounter a NULL pointer in any ** of the third through the sixth parameters of the authorization callback. ** ** ^If the action code is [SQLITE_READ] ** and the callback returns [SQLITE_IGNORE] then the ** [prepared statement] statement is constructed to substitute ** a NULL value in place of the table column that would have ** been read if [SQLITE_OK] had been returned. The [SQLITE_IGNORE] ** return can be used to deny an untrusted user access to individual ** columns of a table. ** ^When a table is referenced by a [SELECT] but no column values are ** extracted from that table (for example in a query like ** "SELECT count(*) FROM tab") then the [SQLITE_READ] authorizer callback ** is invoked once for that table with a column name that is an empty string. ** ^If the action code is [SQLITE_DELETE] and the callback returns ** [SQLITE_IGNORE] then the [DELETE] operation proceeds but the ** [truncate optimization] is disabled and all rows are deleted individually. ** ** An authorizer is used when [sqlite3_prepare | preparing] ** SQL statements from an untrusted source, to ensure that the SQL statements ** do not try to access data they are not allowed to see, or that they do not ** try to execute malicious statements that damage the database. For ** example, an application may allow a user to enter arbitrary ** SQL queries for evaluation by a database. But the application does ** not want the user to be able to make arbitrary changes to the ** database. An authorizer could then be put in place while the ** user-entered SQL is being [sqlite3_prepare | prepared] that ** disallows everything except [SELECT] statements. ** ** Applications that need to process SQL from untrusted sources ** might also consider lowering resource limits using [sqlite3_limit()] ** and limiting database size using the [max_page_count] [PRAGMA] ** in addition to using an authorizer. ** ** ^Only a single authorizer can be in place on a database connection ** at a time. Each call to sqlite3_set_authorizer overrides the ** previous call.)^ ^Disable the authorizer by installing a NULL callback. ** The authorizer is disabled by default. ** ** The authorizer callback must not do anything that will modify ** the database connection for the meaning of "modify" in this paragraph. ** ** ^When [sqlite3_prepare_v2()] is used to prepare a statement, the ** statement might be re-prepared during [sqlite3_step()] due to a ** schema change. Hence, the application should ensure that the ** correct authorizer callback remains in place during the [sqlite3_step()]. ** ** ^Note that the authorizer callback is invoked only during ** [sqlite3_prepare()] or its variants. Authorization is not ** performed during statement evaluation in [sqlite3_step()], unless ** as stated in the previous paragraph, sqlite3_step() invokes ** sqlite3_prepare_v2() to reprepare a statement after a schema change.

80978. Argument to pass to xNotify

80979. **comment:** ** CAPI3REF: Destroy a snapshot *** EXPERIMENTAL *** ^The [sqlite3_snapshot_free(P)] interface destroys [sqlite3_snapshot] P. ** The application must eventually free every [sqlite3_snapshot] object ** using this routine to avoid a memory leak. *** The [sqlite3_snapshot_free()] interface is only available when the ** SQLITE_ENABLE_SNAPSHOT compile-time option is used.
- label:** code-design
80980. NULL NULL
80981. Authorization denied
80982. OUT: Statement handle
80983. ** CAPI3REF: Create A New Session Object *** Create a new session object attached to database handle db. If successful, ** a pointer to the new object is written to *ppSession and SQLITE_OK is ** returned. If an error occurs, *ppSession is set to NULL and an SQLite ** error code (e.g. SQLITE_NOMEM) is returned. *** It is possible to create multiple session objects attached to a single ** database handle. *** Session objects created using this function should be deleted using the ** [sqlite3session_delete()] function before the database handle that they ** are attached to is itself closed. If the database handle is closed before ** the session object is deleted, then the results of calling any session ** module function, including [sqlite3session_delete()] on the session object ** are undefined. *** Because the session module uses the [sqlite3_preupdate_hook()] API, it ** is not possible for an application to register a pre-update hook on a ** database handle that has one or more session objects attached. Nor is ** it possible to create a session object attached to a database handle for ** which a pre-update hook is already defined. The results of attempting ** either of these things are undefined. *** The session object will be used to create changesets for tables in ** database zDb, where zDb is either "main", or "temp", or the name of an ** attached database. It is not an error if database zDb is not attached ** to the database when the session object is created.
80984. ** CAPI3REF: Deprecated Soft Heap Limit Interface *** DEPRECATED *** This is a deprecated version of the [sqlite3_soft_heap_limit64()] ** interface. This routine is provided for historical compatibility ** only. All new applications should use the ** [sqlite3_soft_heap_limit64()] interface rather than this one.
80985. Context passed to xToken()
80986. Deinitialize the memory allocator
80987. ** CAPI3REF: Close A BLOB Handle ** DESTRUCTOR: sqlite3_blob *** ^This function closes an open [BLOB handle]. ^The BLOB handle is closed ** unconditionally. Even if this routine returns an error code, the ** handle is still closed.^ ** ^If the blob handle being closed was opened for read-write access, and if ** the database is in auto-commit mode and there are no other open read-write ** blob handles or active write statements, the current transaction is ** committed. ^If an error occurs while committing the transaction, an error ** code is returned and the transaction rolled back. *** Calling this function with an argument that is not a NULL pointer or an ** open blob handle results in undefined behaviour. ^Calling this routine ** with a null pointer (such as would be returned by a failed call to ** [sqlite3_blob_open()]) is a harmless no-op. ^Otherwise, if this function ** is passed a valid open blob handle, the values returned by the ** sqlite3_errcode() and sqlite3errmsg() functions are set before returning.
80988. ** CAPI3REF: Write-Ahead Log Commit Hook ** METHOD: sqlite3 *** ^The [sqlite3_wal_hook()] function is used to register a callback that ** is invoked each time data is committed to a database in wal mode. *** ^The callback is invoked by SQLite after the commit has taken place and ** the associated write-lock on the database released^, so the implementation ** may read, write or [checkpoint] the database as required. *** ^The first parameter passed to the callback function when it is invoked ** is a copy of the third parameter passed to sqlite3_wal_hook() when ** registering the callback. ^The second is a copy of the database handle. ** ^The third parameter is the name of the database that was written to - ** either "main" or the name of an [ATTACH]-ed database. ^The fourth parameter ** is the number of pages currently in the write-ahead log file, ** including those that were just committed. *** ^The callback function should normally return [SQLITE_OK]. ^If an error ** code is returned, that error will propagate back up through the ** SQLite code base to cause the statement that provoked the callback ** to report an error, though the commit will have still occurred. If the ** callback returns [SQLITE_ROW] or [SQLITE_DONE], or if it returns a value ** that does not correspond to any valid SQLite error code, the results ** are undefined. *** ^A single database handle may have at most a single write-ahead log callback ** registered at one time. ^Calling [sqlite3_wal_hook()] replaces any ** previously registered write-ahead log callback. ^Note that the ** [sqlite3_wal_autocheckpoint()] interface and the ** [wal_autocheckpoint pragma] both invoke [sqlite3_wal_hook()] and will ** overwrite any prior [sqlite3_wal_hook()] settings.
80989. **comment:** ** CAPI3REF: Suspend Execution For A Short Time *** The sqlite3_sleep() function causes the current thread to suspend execution ** for at least a number of milliseconds specified in its parameter. *** ^If the operating system does not support sleep requests with ** millisecond time resolution, then the time will be rounded up to ** the nearest second. The number of milliseconds of sleep actually ** requested from the operating system is returned. *** ^SQLite implements this interface by calling the xSleep() ** method of the default [sqlite3_vfs] object. If the xSleep() method ** of the default VFS is not implemented correctly, or not implemented at ** all, then the behavior of sqlite3_sleep() may deviate from the description ** in the previous paragraphs.
- label:** code-design
80990. **comment:** ** CAPI3REF: Attempt To Free Heap Memory *** ^The sqlite3_release_memory() interface attempts to free N bytes ** of heap memory by deallocating non-essential memory allocations ** held by the database library. Memory used to cache database ** pages to improve performance is an example of non-essential memory. ** ^sqlite3_release_memory() returns the number of bytes actually freed, ** which might be more or less than the amount requested. ** ^The sqlite3_release_memory() routine is a no-op returning zero ** if SQLite is not compiled with [SQLITE_ENABLE_MEMORY_MANAGEMENT]. *** See also: [sqlite3_db_release_memory()]
- label:** code-design
80991. **comment:** ** CAPI3REF: OS Interface File Virtual Methods Object *** Every file opened by the [sqlite3_vfs.xOpen] method populates an ** [sqlite3_file] object (or, more commonly, a subclass of the ** [sqlite3_file] object) with a pointer to an instance of this object. ** This object defines the methods used to perform various operations ** against the open file represented by the [sqlite3_file] object. *** ^If the [sqlite3_vfs.xOpen] method sets the sqlite3_file.pMethods element ** to a non-NULL pointer, then the sqlite3_io_methods.xClose method ** may be invoked even if the [sqlite3_vfs.xOpen] reported that it failed. The ** only way to prevent a call to xClose following a failed [sqlite3_vfs.xOpen] ** is for the [sqlite3_vfs.xOpen] to set the sqlite3_file.pMethods element ** to NULL. *** ^The flags argument to xSync may be one of [SQLITE_SYNC_NORMAL] or ** [SQLITE_SYNC_FULL]. The first choice is the normal fsync(). ** The second choice is a Mac OS X style fullsync. The [SQLITE_SYNC_DATAONLY] ** flag may be ORed in to indicate that only the data of the file ** and not its inode needs to be synced. *** ^The integer values to xLock() and xUnlock() are one of ** ** [SQLITE_LOCK_NONE], ** [SQLITE_LOCK_SHARED], ** [SQLITE_LOCK_RESERVED], ** [SQLITE_LOCK_PENDING], or ** [SQLITE_LOCK_EXCLUSIVE]. ** ** xLock() increases the lock. xUnlock() decreases the lock. ** ^The xCheckReservedLock() method checks whether any database connection, ** either in this process or in some other process, is holding a RESERVED, ** PENDING, or EXCLUSIVE lock on the file. It returns true ** if such a lock exists and false otherwise. *** ^The xFileControl() method is a generic interface that allows custom ** VFS implementations to directly control an open file using the ** [sqlite3_file_control()] interface. The second "op" argument is an ** integer opcode. The third argument is a generic pointer intended to ** point to a structure that may contain arguments or space in which to ** write return values. Potential uses for xFileControl() might be ** functions to enable blocking locks with timeouts, to change the ** locking strategy (for example to use dot-file locks), to inquire ** about the status of a lock, or to break stale locks. The SQLite ** core reserves all opcodes less than 100 for its own use. ** ^A [file control opcodes | list of opcodes] less than 100 is available. ** ^Applications that define a custom xFileControl method should use opcodes ** greater than 100 to avoid conflicts. VFS implementations should ** return [SQLITE_NOTFOUND] for file control opcodes that they do not ** recognize. *** ^The xSectorSize() method returns the sector size of the ** device that underlies the file. The sector size is the ** minimum write that can be performed without disturbing ** other bytes in the file. The xDeviceCharacteristics() ** method returns a bit vector describing behaviors of the ** underlying device. *** ** [SQLITE_IOCAP_ATOMIC] ** [SQLITE_IOCAP_ATOMIC512] ** [SQLITE_IOCAP_ATOMIC1K] ** [SQLITE_IOCAP_ATOMIC2K] ** [SQLITE_IOCAP_ATOMIC4K] ** [SQLITE_IOCAP_ATOMIC8K] ** [SQLITE_IOCAP_ATOMIC16K] ** [SQLITE_IOCAP_ATOMIC32K] ** [SQLITE_IOCAP_ATOMIC64K] ** [SQLITE_IOCAP_SAFE_APPEND] ** [SQLITE_IOCAP_SEQUENTIAL] ** [SQLITE_IOCAP_UNDELETABLE_WHEN_OPEN] ** [SQLITE_IOCAP_POWERSAFE_OVERWRITE] ** [SQLITE_IOCAP_IMMUTABLE] ** ** ^The SQLITE_IOCAP_ATOMIC property means that all writes of ** any size are atomic. The SQLITE_IOCAP_ATOMICnnn values ** mean that writes of blocks that are nnn bytes in size and ** are aligned to an address which is an integer multiple of ** nnn are atomic. The SQLITE_IOCAP_SAFE_APPEND value means ** that when data is appended to a file, the data is appended ** first then the size of the file is extended, never the other ** way around. The SQLITE_IOCAP_SEQUENTIAL property means that ** information is written to disk in the same order as calls ** to xWrite(). *** ^If xRead() returns SQLITE_IOERR_SHORT_READ it must also fill ** in the unread portions of the buffer with zeros. A VFS that ** fails to zero-fill short reads might seem to work. However, ** failure to zero-fill short reads will eventually lead to ** database corruption.
- label:** code-design
80992. Free a prior allocation
80993. ** CAPI3REF: Enable Or Disable A Session Object *** Enable or disable the recording of changes by a session object. When ** enabled, a session object records changes made to the database. When ** disabled - it does not. A newly created session object is enabled. ** Refer to the documentation for [sqlite3session_changeset()] for further ** details regarding how enabling and disabling a session object affects ** the eventual changesets. *** ^Passing zero to this function disables the session. Passing a value ** greater than zero enables it. Passing a value less than zero is a ** no-op, and may be used to query the current state of the session. *** ^The return value indicates the final state of the session object: 0 if ** the session is disabled, or 1 if it is enabled.

80994. Find an existing tokenizer

80995. ** CAPI3REF: Set Or Clear the Indirect Change Flag *** Each change recorded by a session object is marked as either direct or ** indirect. A change is marked as indirect if either: *** ** The session object "indirect" flag is set when the change is ** made, or ** The change is made by an SQL trigger or foreign key action ** instead of directly as a result of a users SQL statement. ** *** If a single row is affected by more than one operation within a session, ** then the change is considered indirect if all operations meet the criteria ** for an indirect change above, or direct otherwise. ** This function is used to set, clear or query the session object indirect ** flag. If the second argument passed to this function is zero, then the ** indirect flag is cleared. If it is greater than zero, the indirect flag ** is set. Passing a value less than zero does not modify the current value ** of the indirect flag, and may be used to query the current state of the ** indirect flag for the specified session object. ** The return value indicates the final state of the indirect flag: 0 if ** it is clear, or 1 if it is set.

80996. OUT: Number of bytes in output changeset

80997. **comment:** ** CAPI3REF: Test To See If The Library Is Threadsafe *** ^The sqlite3_threadsafe() function returns zero if and only if ** SQLite was compiled with mutexing code omitted due to the ** [SQLITE_THREADS] compile-time option being set to 0. ** SQLite can be compiled with or without mutexes. When ** the [SQLITE_THREADS] C preprocessor macro is 1 or 2, mutexes ** are enabled and SQLite is threadsafe. When the ** [SQLITE_THREADS] macro is 0, ** the mutexes are omitted. Without the mutexes, it is not safe ** to use SQLite concurrently from more than one thread. ** Enabling mutexes incurs a measurable performance penalty. ** So if speed is of utmost importance, it makes sense to disable ** the mutexes. But for maximum safety, mutexes should be enabled. ** ^The default behavior is for mutexes to be enabled. ** This interface can be used by an application to make sure that the ** version of SQLite that it is linking against was compiled with ** the desired setting of the [SQLITE_THREADS] macro. ** This interface only reports on the compile-time mutex setting ** of the [SQLITE_THREADS] flag. If SQLite is compiled with ** SQLITE_THREADS=1 or =2 then mutexes are enabled by default but ** can be fully or partially disabled using a call to [sqlite3_config()] ** with the verbs [SQLITE_CONFIG_SINGLETHREAD], [SQLITE_CONFIG_MULTITHREAD], ** or [SQLITE_CONFIG_SERIALIZED]. ^The return value of the ** sqlite3_threadsafe() function shows only the compile-time setting of ** thread safety, not any run-time changes to that setting made by ** sqlite3_config(). In other words, the return value from sqlite3_threadsafe() ** is unchanged by calls to sqlite3_config(). ^ ** See the [threading mode] documentation for additional information.

label: requirement

80998. Don't allow access, but don't generate an error

80999. ** CAPI3REF: Compile-Time Library Version Numbers *** ^The [SQLITE_VERSION] C preprocessor macro in the sqlite3.h header ** evaluates to a string literal that is the SQLite version in the ** format "X.Y.Z" where X is the major version number (always 3 for ** SQLite3) and Y is the minor version number and Z is the release number.)^ ** ^The [SQLITE_VERSION_NUMBER] C preprocessor macro resolves to an integer ** with the value (X*1000000 + Y*1000 + Z) where X, Y, and Z are the same ** numbers used in [SQLITE_VERSION].)^ ** The SQLITE_VERSION_NUMBER for any given release of SQLite will also ** be larger than the release from which it is derived. Either Y will ** be held constant and Z will be incremented or else Y will be incremented ** and Z will be reset to zero. ** Since [version 3.6.18] ([dateof:3.6.18]), ** SQLite source code has been stored in the ** Fossil configuration management ** system. ^The SQLITE_SOURCE_ID macro evaluates to ** a string which identifies a particular check-in of SQLite ** within its configuration management system. ^The SQLITE_SOURCE_ID ** string contains the date and time of the check-in (UTC) and a SHA1 ** or SHA3-256 hash of the entire source tree. ** See also: [sqlite3_libversion()], ** [sqlite3_libversion_number()], [sqlite3_sourceid()], ** [sqlite_version()] and [sqlite_source_id()].

81000. ***** Begin file sqlite3session.h *****

81001. Methods above are valid for version 3

81002. sqlite3_step() has finished executing

81003. True if output is already ordered

81004. Round up request size to allocation size

81005. **comment:** ** CAPI3REF: Obtain new.* Values From A Changeset Iterator *** The pIter argument passed to this function may either be an iterator ** passed to a conflict-handler by [sqlite3changeset_apply()], or an iterator ** created by [sqlite3changeset_start()]. In the latter case, the most recent ** call to [sqlite3changeset_next()] must have returned SQLITE_ROW. ** Furthermore, it may only be called if the type of change that the iterator ** currently points to is either [SQLITE_UPDATE] or [SQLITE_INSERT]. Otherwise, ** this function returns [SQLITE_MISUSE] and sets *ppValue to NULL. ** Argument iVal must be greater than or equal to 0, and less than the number ** of columns in the table affected by the current change. Otherwise, ** [SQLITE_RANGE] is returned and *ppValue is set to NULL. ** If successful, this function sets *ppValue to point to a protected ** sqlite3_value object containing the iVal'th value from the vector of ** new row values stored as part of the UPDATE or INSERT change and ** returns SQLITE_OK. If the change is an UPDATE and does not include ** a new value for the requested column, *ppValue is set to NULL and ** SQLITE_OK returned. The name of the function comes from the fact that ** this is similar to the "new.*" columns available to update or delete ** triggers. ** If some other error occurs (e.g. an OOM condition), an SQLite error code ** is returned and *ppValue is set to NULL.

label: code-design

81006. Size of changeset blob in bytes

81007. Reserved: 0x0FF00000

81008. **comment:** ** CAPI3REF: Open A BLOB For Incremental I/O ** METHOD: sqlite3 ** CONSTRUCTOR: sqlite3_blob *** ^This interfaces opens a [BLOB handle | handle] to the BLOB located ** in row iRow, column zColumn, table zTable in database zDb; ** in other words, the same BLOB that would be selected by: *** <pre> ** SELECT zColumn FROM zDb.zTable WHERE [rowid] = iRow; ** </pre>^ ** ^Parameter zDb is not the filename that contains the database, but ** rather the symbolic name of the database. For attached databases, this is ** the name that appears after the AS keyword in the [ATTACH] statement. ** For the main database file, the database name is "main". For TEMP ** tables, the database name is "temp").)^ ** ^If the flags parameter is non-zero, then the BLOB is opened for read ** and write access. ^If the flags parameter is zero, the BLOB is opened for ** read-only access. ** ^On success, [SQLITE_OK] is returned and the new [BLOB handle] is stored ** in *ppBlob. Otherwise an [error code] is returned and, unless the error ** code is SQLITE_MISUSE, *ppBlob is set to NULL.)^ ** This means that, provided ** the API is not misused, it is always safe to call [sqlite3_blob_close()] ** on *ppBlob after this function it returns. ** This function fails with SQLITE_ERROR if any of the following are true: ** ** ^The database zDb does not exist), ** ^The table zTable does not exist within database zDb), ** ^The table zTable is a WITHOUT ROWID table), ** ^The column zColumn does not exist), ** ^Row iRow is not present in the table), ** ^The specified column of row iRow contains a value that is not ** a TEXT or BLOB value), ** ^Column zColumn is part of an index, PRIMARY KEY or UNIQUE ** constraint and the blob is being opened for read/write access), ** ^([foreign key constraints | Foreign key constraints] are enabled, ** column zColumn is part of a [child key] definition and the blob is ** being opened for read/write access). ** ** ^Unless it returns SQLITE_MISUSE, this function sets the ** [database connection] error code and message accessible via ** [sqlite3_errcode()] and [sqlite3_errmsg()] and related functions. ** ^A BLOB referenced by sqlite3_blob_open() may be read using the ** [sqlite3_blob_read()] interface and modified by using ** [sqlite3_blob_write()]. The [BLOB handle] can be moved to a ** different row of the same table using the [sqlite3_blob_reopen()] ** interface. However, the column, table, or database of a [BLOB handle] ** cannot be changed after the [BLOB handle] is opened. ** ^If the row that a BLOB handle points to is modified by an ** [UPDATE], [DELETE], or by [ON CONFLICT] side-effects ** then the BLOB handle is marked as "expired". ** This is true if any column of the row is changed, even a column ** other than the one the BLOB handle is open on.)^ ** ^Calls to [sqlite3_blob_read()] and [sqlite3_blob_write()] for ** an expired BLOB handle fail with a return code of [SQLITE_ABORT]. ** ^Changes written into a BLOB prior to the BLOB expiring are not ** rolled back by the expiration of the BLOB. Such changes will eventually ** commit if the transaction continues to completion.)^ ** ^Use the [sqlite3_blob_bytes()] interface to determine the size of ** the opened blob. ^The size of a blob may not be changed by this ** interface. Use the [UPDATE] SQL command to change the size of a ** blob. ** ^The [sqlite3_bind_zeroblob()] and [sqlite3_result_zeroblob()] interfaces ** and the built-in [zeroblob] SQL function may be used to create a ** zero-filled blob to read or write using the incremental-blob interface. ** ^To avoid a resource leak, every open [BLOB handle] should eventually ** be released by a call to [sqlite3_blob_close()]. ** See also: [sqlite3_blob_close()], ** [sqlite3_blob_reopen()], [sqlite3_blob_read()], ** [sqlite3_blob_bytes()], [sqlite3_blob_write()].

label: code-design

81009. Input: Mask of columns used by statement

81010. ** CAPI3REF: Number of columns in a result set ** METHOD: sqlite3_stmt *** ^The sqlite3_data_count(P) interface returns the number of columns in the ** current row of the result set of [prepared statement] P. ** ^If prepared statement P does not have results ready to return ** (via calls to the [sqlite3_column_int | sqlite3_column_*] of ** interfaces) then sqlite3_data_count(P) returns 0. ** ^The sqlite3_data_count(P) routine also returns 0 if P is a NULL pointer. ** ^The sqlite3_data_count(P) routine returns 0 if the previous call to ** [sqlite3_step](P) returned [SQLITE_DONE]. ^The sqlite3_data_count(P) ** will return non-zero if previous call to [sqlite3_step](P) returned ** [SQLITE_ROW], except in the case of the [PRAGMA incremental_vacuum] ** where it always returns zero since each step of that multi-step ** pragma returns 0 columns of data. ** See also: [sqlite3_column_count()]

81011. Handle describing change and conflict

81012. **comment:** ** CAPI3REF: Define New Collating Sequences ** METHOD: sqlite3 *** ^These functions add, remove, or modify a [collation] associated ** with the [database connection] specified as the first argument. ** ^The name of the collation is a UTF-8 string ** for sqlite3_create_collation()

sqlite3_create_collation_v2() ** and a UTF-16 string in native byte order for sqlite3_create_collation16(). ** ^Collation names that compare equal according to [sqlite3_strnicmp()] are ** considered to be the same name. ** ** ^The third argument (eTextRep) must be one of the constants: ** ** [SQLITE_UTF8], ** [SQLITE_UTF16LE], ** [SQLITE_UTF16BE], ** [SQLITE_UTF16], or ** [SQLITE_UTF16_ALIGNED]. **)^ ** ^The eTextRep argument determines the encoding of strings passed ** to the collating function callback, xCallback. ** ^The [SQLITE_UTF16] and [SQLITE_UTF16_ALIGNED] values for eTextRep ** force strings to be UTF16 with native byte order. ** ^The [SQLITE_UTF16_ALIGNED] value for eTextRep forces strings to begin ** on an even byte address. ** ** ^The fourth argument, pArg, is an application data pointer that is passed ** through as the first argument to the collating function callback. ** ** ^The fifth argument, xCallback, is a pointer to the collating function. ** ^Multiple collating functions can be registered using the same name but ** with different eTextRep parameters and SQLite will use whichever ** function requires the least amount of data transformation. ** ^If the xCallback argument is NULL then the collating function is ** deleted. ^When all collating functions having the same name are deleted, ** that collation is no longer usable. ** ** ^The collating function callback is invoked with a copy of the pArg ** application data pointer and with two strings in the encoding specified ** by the eTextRep argument. The collating function must return an ** integer that is negative, zero, or positive ** if the first string is less than, equal to, or greater than the second, ** respectively. A collating function must always return the same answer ** given the same inputs. If two or more collating functions are registered ** to the same collation name (using different eTextRep values) then all ** must give an equivalent answer when invoked with equivalent strings. ** The collating function must obey the following properties for all ** strings A, B, and C: ** ** ** If A==B then B==A. ** If A==B and B==C then A==C. ** If A<B THEN B>A. ** If A<B and B<C then A<C. ** ** ** If a collating function fails any of the above constraints and that ** collating function is registered and used, then the behavior of SQLite ** is undefined. ** ** ^The sqlite3_create_collation_v20 works like sqlite3_create_collation() ** with the addition that the xDestroy callback is invoked on pArg when ** the collating function is deleted. ** ^Collating functions are deleted when they are overridden by later ** calls to the collation creation functions or when the ** [database connection] is closed using [sqlite3_close()]. ** ** ^The xDestroy callback is <u>not</u> called if the ** sqlite3_create_collation_v2() function fails. Applications that invoke ** sqlite3_create_collation_v2() with a non-NULL xDestroy argument should ** check the return code and dispose of the application data pointer ** themselves rather than expecting SQLite to deal with it for them. ** This is different from every other SQLite interface. The inconsistency ** is unfortunate but cannot be changed without breaking backwards ** compatibility. ** ** See also: [sqlite3_collation_needed()] and [sqlite3_collation_needed16()].

label: code-design

81013. ** CAPI3REF: Determine The Virtual Table Conflict Policy ** ** This function may only be called from within a call to the [xUpdate] method ** of a [virtual table] implementation for an INSERT or UPDATE operation. ^The ** value returned is one of [SQLITE_ROLLBACK], [SQLITE_IGNORE], [SQLITE_FAIL], ** [SQLITE_ABORT], or [SQLITE_REPLACE], according to the [ON CONFLICT] mode ** of the SQL statement that triggered the call to the [xUpdate] method of the ** [virtual table].

81014. _FTS5_H

81015. ** CUSTOM AUXILIARY FUNCTIONS *****

81016. Table name

81017. int int*

81018. ** CAPI3REF: Obtain Values For URI Parameters ** ** These are utility routines, useful to VFS implementations, that check ** to see if a database file was a URI that contained a specific query ** parameter, and if so obtains the value of that query parameter. ** ** If F is the database filename pointer passed into the xOpen() method of ** a VFS implementation when the flags parameter to xOpen() has one or ** more of the [SQLITE_OPEN_URI] or [SQLITE_OPEN_MAIN_DB] bits set and ** P is the name of the query parameter, then ** sqlite3_uri_parameter(F,P) returns the value of the P ** parameter if it exists or a NULL pointer if P does not appear as a ** query parameter on F. If P is a query parameter of F ** has no explicit value, then sqlite3_uri_parameter(F,P) returns ** a pointer to an empty string. ** ** The sqlite3_uri_boolean(F,P,B) routine assumes that P is a boolean ** parameter and returns true (1) or false (0) according to the value ** of P. The sqlite3_uri_boolean(F,P,B) routine returns true (1) if the ** value of query parameter P is one of "yes", "true", or "on" in any ** case or if the value begins with a non-zero number. The ** sqlite3_uri_boolean(F,P,B) routines returns false (0) if the value of ** query parameter P is one of "no", "false", or "off" in any case or ** if the value begins with a numeric zero. If P is not a query ** parameter on F or if the value of P is does not match any of the ** above, then sqlite3_uri_boolean(F,P,B) returns (B!=0). ** ** The sqlite3_uri_int64(F,P,D) routine converts the value of P into a ** 64-bit signed integer and returns that integer, or D if P does not ** exist. If the value of P is something other than an integer, then ** zero is returned. ** ** If F is a NULL pointer, then sqlite3_uri_parameter(F,P) returns NULL and ** sqlite3_uri_boolean(F,P,B) returns B. If F is not a NULL pointer and ** is not a database file pathname pointer that SQLite passed into the xOpen ** VFS method, then the behavior of this routine is undefined and probably ** undesirable.

81019. True for DESC. False for ASC.

81020. Coordinates of node or entry to check

81021. Structure version number (currently 3)

81022. Table of WHERE clause constraints

81023. ** CAPI3REF: Changeset Iterator Handle

81024. Successful result

81025. Prepared statement for which info desired

81026. Next registered VFS

81027. DATA, MISSING, CONFLICT, CONSTRAINT

81028. For use by application

81029. Result written here

81030. ** CAPI3REF: Generate A Patchset From A Session Object ** ** The differences between a patchset and a changeset are that: ** ** ** DELETE records consist of the primary key fields only. The ** original values of other fields are omitted. ** The original values of any modified fields are omitted from ** UPDATE records. ** ** ** A patchset blob may be used with up to date versions of all ** sqlite3changeset_xxx API functions except for sqlite3changeset_invert(), ** which returns SQLITE_CORRUPT if it is passed a patchset. Similarly, ** attempting to use a patchset blob with old versions of the ** sqlite3changeset_xxx APIs also provokes an SQLITE_CORRUPT error. ** ** Because the non-primary key "old.*" fields are omitted, no ** SQLITE_CHANGESET_DATA conflicts can be detected or reported if a patchset ** is passed to the sqlite3changeset_apply() API. Other conflict types work ** in the same way as for changesets. ** ** Changes within a patchset are ordered in the same way as for changesets ** generated by the sqlite3session_changeset() function (i.e. all changes for ** a single table are grouped together, tables appear in the order in which ** they were attached to the session object).

81031. Destination database handle

81032. **comment:** ** CAPI3REF: Obtain Aggregate Function Context ** METHOD: sqlite3_context ** ** Implementations of aggregate SQL functions use this ** routine to allocate memory for storing their state. ** ** ^The first time the sqlite3_aggregate_context(C,N) routine is called ** for a particular aggregate function, SQLite ** allocates N of memory, zeroes out that memory, and returns a pointer ** to the new memory. ^On second and subsequent calls to ** sqlite3_aggregate_context() for the same aggregate function instance, ** the same buffer is returned. Sqlite3_aggregate_context() is normally ** called once for each invocation of the xStep callback and then one ** last time when the xFinal callback is invoked. ^When no rows match ** an aggregate query, the xStep() callback of the aggregate function ** implementation is never called and xFinal() is called exactly once. ** In those cases, sqlite3_aggregate_context() might be called for the ** first time from within xFinal().)^ ** ** ^The sqlite3_aggregate_context(C,N) routine returns a NULL pointer ** when first called if N is less than or equal to zero or if a memory ** allocate error occurs. ** ** ^The amount of space allocated by sqlite3_aggregate_context(C,N) is ** determined by the N parameter on first successful call. Changing the ** value of N in subsequent call to sqlite3_aggregate_context() within ** the same aggregate function instance will not resize the memory ** allocation.)^ Within the xFinal callback, it is customary to set ** N=0 in calls to sqlite3_aggregate_context(C,N) so that no ** pointless memory allocations occur. ** ** ^SQLite automatically frees the memory allocated by ** sqlite3_aggregate_context() when the aggregate query concludes. ** ** The first parameter must be a copy of the ** [sqlite3_context | SQL function context] that is the first parameter ** to the xStep or xFinal callback routine that implements the aggregate ** function. ** ** This routine must be called from the same thread in which ** the aggregate SQL function is running.

label: code-design

81033. ** END OF CUSTOM TOKENIZERS *****

81034. Error msg written here

81035. ** CAPI3REF: Return The Size Of An Open BLOB ** METHOD: sqlite3_blob ** ** ^Returns the size in bytes of the BLOB accessible via the ** successfully opened [BLOB handle] in its only argument. ^The ** incremental blob I/O routines can only read or overwriting existing ** blob content; they cannot change the size of a blob. ** ** This routine only works on a [BLOB handle] which has been created ** by a prior successful call to [sqlite3_blob_open()] and which has not ** been closed by [sqlite3_blob_close()]. Passing any other pointer in ** to this routine results in undefined and probably undesirable behavior.

81036. Insertion failed because database is full

81037. The key

81038. ** CAPI3REF: Find The Database Handle Of A Prepared Statement ** METHOD: sqlite3_stmt ** ** ^The sqlite3_db_handle interface returns the [database connection] handle ** to which a [prepared statement] belongs. ^The [database connection] ** returned by sqlite3_db_handle is the same [database connection] **

that was the first argument ** to the [sqlite3_prepare_v2()] call (or its variants) that was used to ** create the statement in the first place.

81039. xSqllog, void*

81040. ** CAPI3REF: Reset All Bindings On A Prepared Statement ** METHOD: sqlite3_stmt ** ** ^Contrary to the intuition of many, [sqlite3_reset()] does not reset ** the [sqlite3_bind_blob | bindings] on a [prepared statement]. ** ^Use this routine to reset all host parameters to NULL.

81041. Methods above are valid for version 1

81042. **comment:** ** CAPI3REF: Column Names In A Result Set ** METHOD: sqlite3_stmt ** ** ^These routines return the name assigned to a particular column ** in the result set of a [SELECT] statement. ^The sqlite3_column_name() ** interface returns a pointer to a zero-terminated UTF-8 string ** and sqlite3_column_name16() returns a pointer to a zero-terminated ** UTF-16 string. ^The first parameter is the [prepared statement] ** that implements the [SELECT] statement. ^The second parameter is the ** column number. ^The leftmost column is number 0. ** ** ^The returned string pointer is valid until either the [prepared statement] ** is destroyed by [sqlite3_finalize()] or until the statement is automatically ** reprepared by the first call to [sqlite3_step()] for a particular run ** or until the next call to ** sqlite3_column_name() or sqlite3_column_name16() on the same column. ** ** ^If sqlite3_malloc() fails during the processing of either routine ** (for example during a conversion from UTF-8 to UTF-16) then a ** NULL pointer is returned. ** ** ^The name of a result column is the value of the "AS" clause for ** that column, if there is an AS clause. If there is no AS clause ** then the name of the column is unspecified and may change from ** one release of SQLite to the next.

label: code-design

81043. Parameters passed to SQL geom function

81044. Mask of FTSS5_TOKEN_* flags

81045. ** CAPI3REF: Enable Or Disable Extension Loading ** METHOD: sqlite3 *** ** ^So as not to open security holes in older applications that are ** unprepared to deal with [extension loading], and as a means of disabling ** [extension loading] while evaluating user-entered SQL, the following API ** is provided to turn the [sqlite3_load_extension()] mechanism on and off. ** ** ^Extension loading is off by default. ** ^Call the sqlite3_enable_load_extension() routine with onoff==1 ** to turn extension loading on and call it with onoff==0 to turn ** it back off again. ** ** ^This interface enables or disables both the C-API ** [sqlite3_load_extension()] and the SQL function [load_extension()]. ** ^Use [sqlite3_db_config](db, [SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION],...) ** to enable or disable only the C-API.^ ** ** Security warning: It is recommended that extension loading ** be disabled using the [SQLITE_DBCONFIG_ENABLE_LOAD_EXTENSION] method ** rather than this interface, so the [load_extension()] SQL function ** remains disabled. This will prevent SQL injections from giving attackers ** access to extension loading capabilities.

81046. ** CAPI3REF: Impose A Limit On Heap Size *** ** ^The sqlite3_soft_heap_limit64() interface sets and/or queries the ** soft limit on the amount of heap memory that may be allocated by SQLite. ** ^SQLite strives to keep heap memory utilization below the soft heap ** limit by reducing the number of pages held in the page cache ** as heap memory usages approaches the limit. ** ^The soft heap limit is "soft" because even though SQLite strives to stay ** below the limit, it will exceed the limit rather than generate ** an [SQLITE_NOMEM] error. In other words, the soft heap limit ** is advisory only. ** ** ^The return value from sqlite3_soft_heap_limit64() is the size of ** the soft heap limit prior to the call, or negative in the case of an ** error. ^If the argument N is negative ** then no change is made to the soft heap limit. Hence, the current ** size of the soft heap limit can be determined by invoking ** sqlite3_soft_heap_limit64() with a negative argument. ** ** ^If the argument N is zero then the soft heap limit is disabled. ** ** ^The soft heap limit is not enforced in the current implementation ** if one or more of following conditions are true: ** ** ** The soft heap limit is set to zero. ** Memory accounting is disabled using a combination of the ** [sqlite3_config]([SQLITE_CONFIG_MEMSTATUS],...) start-time option and ** the [SQLITE_DEFAULT_MEMSTATUS] compile-time option. ** An alternative page cache implementation is specified using ** [sqlite3_config]([SQLITE_CONFIG_PCACHE2],...). ** The page cache allocates from its own memory pool supplied ** by [sqlite3_config]([SQLITE_CONFIG_PAGECACHE],...) rather than ** from the heap. ** ^ ** ** Beginning with SQLite [version 3.7.3] ([dateof:3.7.3]), ** the soft heap limit is enforced ** regardless of whether or not the [SQLITE_ENABLE_MEMORY_MANAGEMENT] ** compile-time option is invoked. With [SQLITE_ENABLE_MEMORY_MANAGEMENT], ** the soft heap limit is enforced on every memory allocation. Without ** [SQLITE_ENABLE_MEMORY_MANAGEMENT], the soft heap limit is only enforced ** when memory is allocated by the page cache. Testing suggests that because ** the page cache is the predominate memory user in SQLite, most ** applications will achieve adequate soft heap limit enforcement without ** the use of [SQLITE_ENABLE_MEMORY_MANAGEMENT]. ** ** The circumstances under which SQLite will enforce the soft heap limit may ** changes in future releases of SQLite.

81047. **comment:** TODO: Do we really need to unlock the mutex, before * destroying it, if it's destroyed by the thread currently * owning the mutex?

label: code-design

81048. HCP24: moving all timers > next_time

81049. HCP24: if is_relative = 0 and next_time < now * action will be called so fast as possible * if additional period > 0 * action will be called so fast as possible * n times until (next_time + (n * period)) > now * then the period is working * Solution: * if next_time < now then we set next_time = now. * The first callback will be so fast as possible (now) * but the next callback on period

81050. Timer thread

81051. _WIN32

81052. Current size of timer list

81053. Timer thread ID

81054. Protects timer lists

81055. List of timers

81056. GetTickCount returns milliseconds since system start as * unsigned 32 bit value. It will wrap around every 49.7 days. * We need to use a 64 bit counter (will wrap in 500 mio. years), * by adding the 32 bit difference since the last call to a * 64 bit counter. This algorithm will only work, if this * function is called at least once every 7 weeks.

81057. Start timer thread

81058. You can not set timers into the past

81059. This file is part of the CivetWeb web server. * See <https://github.com/civetweb/civetweb/> * (C) 2014-2017 by the CivetWeb authors, MIT license.

81060. Insert new timer into a sorted list

81061. **comment:** The linear list is still most efficient for short lists (small * number of timers) - if there are many timers, different * algorithms will work better.
label: code-design

81062. 10 ms seems reasonable. * A faster loop (smaller sleep value) increases CPU load, * a slower loop (higher sleep value) decreases timer accuracy.

81063. End of timer.inl

81064. Test case description: This test contains 100 small images in a table. Once a browser opens the html file, it will request all these images from the server very quickly. Depending on the "keep-alive" settings, it will either open/close 100 connections quite rapidly if keep-alive=no, or otherwise establish only a few connections, typically one or two, and reuse them. If the test succeeds, all 100 images are displayed. The loading time is measured automatically in the browser using JavaScript. Note that the load times also differs between HTTP and HTTPS.

81065. http://www.restpatterns.org/HTTP_Methods <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

81066. l/bin/sh

81067. .bM=/?/,bN=<script>b[^\<]*?:(?:!<\!<script><[^<]*></script>/gi,bO=/^(?:select|textarea)/i,bP=/s+/,bQ=/([?&])_==[^&]*/,bR=/^([\w\+\.\-]+:)?:(?:\d+)?/?,bS=f.fn.load,bT={},bU={},bV,bW,bX=[\\"/\"]+["*"],try{bV=e.href}catch(bY){bV=c.createElement("a"),bV.href=""},bV=bV.href||bR.exec(bV.toLowerCase())||[],f.fn.extend({load:function(a,c,d){if(typeof a!="string"&&bS) return bS.apply(this,arguments);if(!this.length) return this;var e=a.indexOff(" ");if(e>=0){var g=a.slice(e,a.length);a=a.slice(0,e)}var h="GET";c&&(f.isFunction(c)?(d=c,c=b):typeof c=="object"&&(c=f.param(c,f.ajaxSettings.traditional),h="POST"));var i=this;f.ajax({url:a,type:h,dataType:"html",data:c,complete:function(a,b,c){c=a.responseText,a.isResolved()&&(a.done(function(a){c=a}),i.html(g?f(" <div>").append(c.replace(bN,"")).find(g)):c))},return this},serialize:function(){return f.param(this.serializeArray())},serializeArray:function(){return this.map(function(){return this.elements?f.makeArray(this.elements):this}).filter(function(){return this.name&&!this.disabled&&!(this.checked||bO.test(this.nodeName)||bL.test(this.type))}).map(function(a,b){var c=f(this).val();return c==null?null:f.isArray(c)?f.map(c,function(a){return{name:b,value:a.replace(bF,"\\r\\n")}}):{name:b,value:c.replace(bF,"\\r\\n")}}).get()}},f.each("ajaxStart ajaxStop ajaxComplete ajaxError ajaxSuccess ajaxSend".split(" "),function(a,b){f.fn[b]=function(a){return this.on(b,a)}}),f.extend({getScript:function(a,c){return f.get(a,b,c,"script")},getJSON:function(a,b,c){return f.get(a,b,c,"json")},ajaxSetup:function(a,b){b?b._a(f.ajaxSettings).b_a(b):a},ajaxSettings:{url:bV,isLocal:bJ,test:bW[1],global:!0,type:"GET",contentType:"application/x-www-form-urlencoded",processData:!0,async:!0,accepts:{xml:"application/xml, text/xml",html:"text/html",text:"text/plain",json:"application/json, text/javascript","*":bX},contents:{xml:/xml/,html:/html/,text:/text/,json:/json/},responseFields:{xml:"responseXML",text:"responseText"},converters:{'*':a.String,"text html":!0,"text json":f.parseJSON,"text xml":f.parseXML},flatOptions:{context:!0,url:!0}},ajaxPrefilter:bZ(bT),ajaxTransport:bZ(bU),ajax:function(a,c){function w(a,c,l,m)

```

{if(s!==2){s=2,q&&clearTimeout(q),p=b,n=m||"",v.readyState=a>0?4:0;var o,r,u,w=c,x=l?cb(d,v,l):b,y,z;if(a>=200&&a<300||a==304){if(d.ifModified){if(y=v.getResponseHeader("Last-Modified"))f.lastModified[k]=y;if(z=v.getResponseHeader("Etag"))f.etag[k]=z;if(a==304)w="notmodified",o!=!0;else try{r=cc(d,x),w="success",o!=!0}catch(A){w="parsererror",u=A} }else{u=w;if(!w||a)w="error",a<0&&(a=0)v.status=a,v.statusText=""+(c||w),?h.resolveWith(e,[r,w,y]):h.rejectWith(e,[v,w,u]),v.statusCode(j).j=b,t&&g.trigger("ajax"+"(o?"Success":"Error"),[v,d,o?r:u],i.fireWith(e,[v,w]),t&&(g.trigger("ajaxComplete",[v,d]),-f.active||f.event.trigger("ajaxStop")))}typeof a=="object"&&(c=a,b),c=c||{};var d=f.ajaxSetup({}),c.e=d.context||d,g=e==d&&(e.nodeType||e instanceof f)?f(e):h.isDeferred(),i=f.callbacks("once memory"),j=d.statusCode||{},k={},l={},m={},n,o,p,q,r,s=0,t,u=v={readyState:0,setRequestHeader:function(a,b){if(!s){var c=a.toLowerCase();a=m[c]?m[c]:a,l[a]=b}return this},getAllResponseHeaders:function(){return s==2?n:null},getResponseHeader:function(a){var c;if(s==2){if(!o){o={};while(c=bH.exec(n))o[c[1].toLowerCase()]=c[2]}c=o[a.toLowerCase()]}}return c==b?null:c},overrideMimeType:function(a){s||(d.mime_type=a);return this},abort:function(a){a=a||"abort",p&&p.abort(a),w(0,a);return this};h.promise(v),v.success=v.done,v.error=v.fail,v.complete=i.add,v.statusCode=function(a){if(a){var b;if(s<2)for(b in a)b[j][b]=[j[b],a[b]];else b=a[v.status],v.then(b,b);return this},d.url=((a||d.url)+"").replace(bG,"").replace(bL,bW[1]+"\\"),d.dataTypes=f.trim(d.dataType)||"").toLowerCase().split(bP),d.crossDomain==null&&(r=bR.exec(d.url.toLowerCase()),d.crossDomain!=!(r[r[1]]==bW[1]&&r[2]==bW[2]&&(r[3]||(r[1]==="http:"?80:443))==(bW[3]||(bW[1]==="http:"?80:443))),d.data&&d.processData&&typeof d.data!="string"&&(d.data=f.param(d.data,d.traditional)),b$(bT,d,c,v);if(s==2){return !1;t=d.global,d.type=d.type.toUpperCase(),d.hasContent=!bK.test(d.type),t&&f.active+++=0&&(d.data&&(d.url+=(bM.test(d.url))?"&":"?")+d.data,delete d.data),k=d.url;if(d.cache==!=1){var x=f.now(),y=d.url.replace(bQ,"$1_="+x);d.url=y+(y==d.url?(bM.test(d.url))?"&":"?")+"_=_+x:"})}(d.data&&d.hasContent&&d.contentType!=!=1||c.contentType)&&v.setRequestHeader("Content-Type",d.contentType),d.ifModified&&(k=d.url,f.lastModified[k]&&v.setRequestHeader("If-Modified-Since",f.lastModified[k]),f.etag[k]&&v.setRequestHeader("If-None-Match",f.etag[k]),v.setRequestHeader("Accept",d.dataTypes[0])&&d.accepts[d.dataTypes[0]]?d.accepts[d.dataTypes[0]]+(d.dataTypes[0]!==="*"?","+bX+"; q=0.01":""":d.accepts["*"]);for(u in d.headers)v.setRequestHeader(u,d.headers[u]);if(d.beforeSend&&(d.beforeSend.call(e,v,d)==!=1||s==2)){v.abort();return!1}for(u in{success:1,error:1,complete:1})v[u](d[u]);p=b$(bU,d,c,v);if(!p)w(-1,"No Transport");else{v.readyState=1,t&&g.trigger("ajaxSend",[v,d]),d.async&&d.timeout>0&&(q=setTimeout(function(){v.abort("timeout")},d.timeout));try{s=1,p.send(l,w)}catch(z){if(s<2)w(-1,z);else throw z}}return v},param:function(a,c){var d=[];e=function(a,b){b=f.isFunction(b)?b:0,d[d.length]=encodeURIComponent(a)+"="+encodeURIComponent(b)};c==b&&(c=f.ajaxSettings.traditional);if(f.isArray(a)||a.jquery&&!f.isPlainObject(a))f.each(a,function(){e(this.name,this.value)});else for(var g in a)ca(g,a[g],c,e);return d.join("&").replace(bD,"+"))},f.extend({active:0,lastModified:{},etag:{}},var cd=f.now(),ce=(/=(?:=|)?(&|$)|?)/,?i:f.ajaxSetup({jsonp:"callback",jsonpCallback:function(){return f.expandos+"_"+cd++}},f.ajaxPrefilter("json jsonp",function(b,c,d){var e=b.contentType==="application/x-www-form-urlencoded"&&typeof b.data=="string";if(b.dataTypes[0]==="jsonp")b.jsonp!=!=1&&(ce.test(b.url)||e&&ce.test(b.data))}{var g,h=b.jsonpCallback=fisFunction(b.jsonpCallback)?b.jsonpCallbackCallback,b=jsonpCallback||b.jsonpCallback,i=b.url,k=b.data,l="$1"+h+$2";b.jsonp!=!=1&&(j=j.replace(ce,l),b.url===j&&(e&&(k=k.replace(ce,l)),b.data==k&&(j+==(?/.test(j)?&:"?")+b.jsonp+"_"+h)),b.url=j,b.data=k,a[h]=function(a){g=[a]},d.always(function(){a[h]=i,g&&f.isFunction(i)&&a[h](g[0]))},b.converters["script json"]=function(){g||f.error(h+" was not called");return g[0]},b.dataTypes[0]==="json";return"script"}),f.ajaxSetup({accepts:{script:"text/javascript,application/javascript,application/ecmascript,application/x-ecmascript"},contents:{script:/javascript|ecmascript/},converters:{"text script":function(a){f.globalEval(a);return a}}},f.ajaxPrefilter("script",function(a){a.cache==b&&(a.cache!=1),a.crossDomain&&(a.type=="GET",a.global!=1)},f.ajaxTransport("script",function(a){if(a.crossDomain){var d,e=c.head||c.getElementsByTagName("head")}[0]||c.documentElement;return{send:function(f,g){d=c.createElement("script"),d.async="async",a.scriptCharset&&(d.charset=a.scriptCharset),d.src=a.url,d.onload=d.onreadystatechange=function(a){if(c||!d.readyState||loaded||complete||test(d.readyState))d.onload=d.onreadystatechange=null,e&&d.parentNode&&e.removeChild(d),d=b,c||g(200,"success"),e.insert(d&&d.onload(0,1))}}},var cf=a.ActiveXObject?function(){for(var a in ch)ch[a](0,1):11,cg=0,ch.f.ajaxSettings.xhr=a.ActiveXObject?function(){return this.isLocal&&c[i][cj]||ci,funciton(a)f.extend(f.support,{ajax:!1,cors:!1,&&"withCredentials"in a})}(f.ajaxSettings.xhr()),f.support.ajax&&f.ajaxTransport(function(c){if(c.crossDomain||f.support.cors){var d;return{send:function(e,g){var h=c.xhr(),i;j;c.username?h.open(c.type,c.url,c.async,c.username,c.password):h.open(c.type,c.url,c.async);if(c.xhrFields)for(j in c.xhrFields)h[j]=c.xhrFields[j];c.mimeType&&h.overrideMimeType(c.mimeType),!c.crossDomain&&e["X-Requested-With"]=="XMLHttpRequest";try{for(j in ejh.setRequestHeader(j,ejj)}catch(k){}h.send(c.hasContent&&c.data||null),d=function(a,e){var j,k,l,m,n;try{if(d&&(e|h.readyState==4))d=b,j&&(h.onreadystatechange=f.noop,cf&&delete ch[i]);if(e|h.readyState==4&&h.abort());else{j=h.status,l=h.getAllResponseHeaders(),m={},n=h.responseXML,n&&n.documentElement&&(m.xml=n),m.text=h.responseText;try{h.crossDomain?m.text?200:404:j==1223&&(j=204)}},catch(p){e||g(-1,p)m&&g(j,k,m,l),!c.async||h.readyState==4?d():(i+++,cg,cf&&(ch||(ch={}),f(a).unload(cf),ch[i]=d),h.onreadystatechange=d),abort:function(){d&&d(0,1)}}}},var ck={},cl,cm,cn=/^(?:toggle|show|hide)$/,co=/^([+-]+)=([+-]+)([a-z%]*$/i,ip,cq=[[{"height":"marginTop","marginBottom":"paddingTop","paddingBottom":"paddingLeft","width":"marginLeft","marginRight":"paddingRight"},["opacity"]],cr,f.fn.extend({show:function(a,b,c){var d,e;if(a||a==0)return this.animate(cu("show",3),a,b,c);for(var g=0,h=this.length;g<h;g++)d=this[g],f_data(d,"olddisplay")&&e=="none"&&(e=d.style.display=="none"&&f.css(d,"display")=="none")&&f_data(d,"olddisplay","cv(d.nodeName));for(g=0;g<h;g++)d=this[g];if(d.style){e=d.style.display;if(e=="none")d.style.display=f_data(d,"olddisplay")||""}return this},hide:function(a,b,c){if(a||a==0)return this.animate(cu("hide",3),a,b,c);var d,e,g=0,h=this.length;for(g<h;g++)d=this[g],d.style&&(e=f.css(d,"display"),e!="none"&&!f_data(d,"olddisplay")&&f_data(d,"olddisplay",e));for(g=0;g<h;g++)this[g].style.display="none";return this},_toggle:f.fn.toggle.toggle:function(a,b,c){var d=typeof a=="boolean",f.isFunction(a)&&f.isFunction(b)?this._toggle.apply(this,arguments):a==null||d?this.each(function(){var b=d?a:f(this).is(":hidden");f(this)[b?"show":"hide"]()});this.animate(cu("toggle",3),a,b,c);return this},fadeTo:function(a,b,c,d){return this.filter(":hidden").css("opacity",0).show().end().animate({opacity:b},a,c,d)},animate:function(a,b,c){function g(){e.queue==!=1&&f._mark(this);var b=f.extend({},e),c=this.nodeType==1,d=c&&f(this).is(":hidden"),g,h,i,j,k,l,m,n,o;b.animatedProperties={};for(i in a){g=f.camelCase(i),i==g&&(a[g]=a[i],delete a[i],h=a[g],f.isArray(h))}(b.animatedProperties[g]=h||b.animatedProperties[g]);b.specialEasing&&b.specialEasing[g]||b.easing||"swing";if(h=="hide"&&d|h=="show"&&b.complete.call(this);c&&(g=="height"||g=="width")&&(b.overflow=[this.style.overflow,this.style.overflowX,this.style.overflowY],f.css(this,"display")=="inline"&&f.css(this,"float")=="none"&&(!f.support.inlineBlockNeedsLayout||cv(this.nodeName)=="inline")?this.style.display="inline-block":this.style.zoom=1),b.overflow!=null&&(this.style.overflow="hidden");for(i in a){new f.fx(this,b,i),h=a[i],cn.test(h)?o=f_data(this,"toggle"+i)|(h=="toggle"?d:"show":"hide":0),o?{f_data(this,"toggle"+i,o=="show"?":show":"hide")}:j[o]():k=co.exec(h),l=j.cur(),n=k[m=parseFloat(k[2]),n=k[3]]||f.cssNumber[i]?"px":px,"&&(f.style(this,i,[m||1]+n),l=(m||1).j.cur()*l,f.style(this,i,[n||1]+n),k[1]&&(m=(k[1]===?1:1)*m+l),j.custom(l,m,""));return l0}var e=f.speed(b,c,d);if(f.isEmptyObject(a))return this.each(e.complete,[!1]);a=f.extend({},a);return e.queue==!=1?this.each(g):this.queue(e.queue,g),stop:function(a,c,d){typeof a=="string"&&(d=c,a=a-b),c&&a!=!=1&&this.queue(a||"fx",!1);return this},each(function(){function h(a,b,c){var e=b[c];f.removeData(a,c,!0),e.stop(d)}var b,c;for(b=c=1,e=f.timers,g=f._data(this),d=f._unmark(!0,this);if(a==null)for(b in g)g[b]&&g[b].stop&&b.indexOff("run")===b.length-4&&h(this,g,b);else g[b="a"+".run"]&&g[b].stop&&h(this,g,b);for(b=e.length;b--;)e[b].elem==this&&(a==null||e[b].queue==!=a)&&(d=e[b](!0).j.saveState(),c=!=0,e.splice(b,1));},f.each({slideDown:cu("show",1),slideUp:cu("hide",1),slideToggle:cu("toggle",1),fadeIn:{opacity:"show"},fadeOut:{opacity:"hide"},fadeToggle:{opacity:"toggle"},function(a,b){f.fn[a]=function(c,d){return this.animate(b,a,c,d)}}},f.extend({speed:function(a,b,c){var d=a,b=c,c=a,b=d},duration:f.fx.off?0:typeof d.duration=="number"?d.duration:d.duration in f.fx.speeds},f.fx.speeds[d.duration].f.fx.speeds._default;if(d.queue==null||d.queue==!=0)d.queue="fx",d.old=d.complete,d.complete=function(a){f.isFunction(d.old)&&d.old.call(this),d.queue?f.dequeue(this,d.queue):a!=!=1&&f._unmark(this);return d},easing:{linear:function(a,b,c,d){return c+d*a},swing:function(a,b,c,d){return-(Math.cos(a*Math.PI)/2+.5)*d+c}},timers:[],fx:function(a,b,c){this.options=b,thi.elem=a,thi.prop=c,orig=b.orig},f.fx.prototype={update:function(){this.options.step&&this.options.step.call(this.elem,this.now,this),(f.fx.step[this.prop])||f.fx.step._default}(this),cur:function(){if(this.elem[this.prop]==null&&(this.elem.style[this.prop]==null))return this.elem[this.prop];var a,b=f.css(this.elem,this.prop);return isNaN(a	parseFloat(b))?!b||b=="auto"?0:a},custom:function(a,c,d){function h(a){return e.step(a)}var e=this,g=f.fx;this.startTime=cr||cs(),this.end=c,thi.now=this.start=a,thi.pos=this.state=0,thi.unit=d||this.unit||f.cssNumber[this.prop]?"px":px,h.queue=this.options.queue,h.elem=this.elem,h.saveState=function(){e.options.hide&&f._data(e.elem,"fxshow"+e.prop)===b&&f._data(e.elem,"fxshow"+e.prop,e.start)},h)&&f.timers.push(h)&&cp&&(cp=setInterval(g.tick,g.interval)),show:function(){var a=f._data(this.elem,"fxshow"+this.prop);this.options.orig[this.prop]=a||f.style(this.elem,this.prop),this.options.show!=!0,a==b}

```

this.custom(this.cur(),a):this.custom(this.prop==="width"||this.prop==="height"?1:0,this.cur()),f(this.elem).show(),hide:function()
{this.options.orig[this.prop]=f._data(this.elem,"fxshow"+this.prop)||f.style(this.elem,this.prop),this.options.hide!=0,this.custom(this.cur(),0)},step:function(a){var
b,c,d,e=cr|cs(),g=10,h=this.elem,i=this.options;if(a|>=i.duration+this.startTime)
{this.now=this.end,this.state=1,this.update(),i.animatedProperties[this.prop]=!0;for(b in i.animatedProperties)i.animatedProperties[b]==!0&&(g!=1);if(g
{i.overflow!=null&&!f.support.shrinkWrapBlocks&&f.each(["","X","Y"],function(a,b)
{h.style["overflow"+b]=i.overflow[a],i.hide&&f(h).hide();if(i.hide||i.show)for(b in
i.animatedProperties)f.style(h,b,i.orig[b]),f.removeData(h,"fxshow"+b,!0),f.removeData(h,"toggle"+b,!0);d=i.complete,d&&
(i.complete!=1,d.call(h))}return!1}i.duration==Infinity?this.now=e:(c=e-this.startTime,this.state=c/i.duration,this.pos=f.easing[i.animatedProperties[this.prop]]
(this.state,c,0,1,i.duration),this.now=this.start+(this.end-this.start)*this.pos),this.update();return!0}}},f.extend(f.fx,{tick:function(){var
a,b=f.timers,c=0;for(c<b.length;c++)a=b[c],!a&&b.splice(c--,1),b.length||f.fx.stop(),interval:13,stop:function(){clearInterval(cp),cp=null},speeds:
{slow:600,fast:200,_default:400},step:{opacity:function(a){f.style(a.elem,"opacity",a.now)},_default:function(a){a.elem.style&&a.elem.style[a.prop]!=null?
a.elem.style[a.prop]=a.now+a.unit:a.elem[a.prop]=a.now}}}),f.each(["width","height"],function(a,b){f.fx.step[b]=function(a)
{f.style(a.elem,b,Math.max(0,a.now)+a.unit)}},f.expr&&f.expr.filters&&f.expr.filters.animated=function(a){return f.grep(f.timers,function(b){return
a===(b.elem).length});var cw=/^t(?:able|dl)\$|i,cx=/^(?:body|html)\$|i,"getBoundingClientRect"in c.documentElement?f.fn.offset=function(a){var
b=this[0],c;if(a)return this.each(function(b){f.offset.setOffset(this,a.b)});if(!b||b.ownerDocument)return null;if(b===(b.ownerDocument.body))return
f.offset.bodyOffset(b);try{c=b.getBoundingClientRect()}catch(d){var e=b.ownerDocument,g=e.documentElement;if(!c||!e.contains(g,b))return c?
(top:c.top,left:c.left}:{top:0,left:0};var
h=e.body,i=cy(e),j=g.clientWidth|h.clientWidth||0,k=g.clientHeight||h.clientHeight||0,l=i.pageYOffset||f.support.boxModel&&g.scrollTop||h.scrollTop,m=i.pageXOffset||f.sup
j,o=c.left+m-k;return{top:n:left:o}:f.fn.offset=function(a){var b=this[0];if(a)return this.each(function(b)
{f.offset.setOffset(this,a.b)});if(!b||b.ownerDocument)return null;if(b===(b.ownerDocument.body))return f.offset.bodyOffset(b);var
c,d=b.offsetParent,e=b,g=b.ownerDocument,h=g.documentElement,i=g.body,j=g.defaultView,k=j?
j.getComputedStyle(b,null):b.currentStyle,l=b.offsetTop,m=b.offsetLeft;while((b=b.parentNode)&&b!=i&&b!=h)
{if(f.support.fixedPosition&&k.position=="fixed")break;c=j?j.getComputedStyle(b,null):b.currentStyle,l-=b.scrollTop,m-=b.scrollLeft,b===(d=&&
(l+=b.offsetTop,m+=b.offsetLeft,f.support.doesNotAddBorder&&(!f.support.addBorderForTableAndCells)||cw.test(b.nodeName))&&
(l+=parseFloat(c.borderTopWidth)||0,m+=parseFloat(c.borderLeftWidth)||0),e=d,d=b.offsetParent),f.support.subtractBorderForOverflowNotVisible&&c.overflow!=
(l+=parseFloat(c.borderTopWidth)||0,m+=parseFloat(c.borderLeftWidth)||0),k=c;if(k.position=="relative"||k.position=="static")l+=i.offsetTop,m+=i.offsetLeft,f.s
(l+=Math.max(h.scrollTop,i.scrollTop),m+=Math.max(h.scrollLeft,i.scrollLeft));return{top:l,left:m}},f.offset={bodyOffset:function(a){var
b=a.offsetTop,c=a.offsetLeft,f.support.doesNotIncludeMarginInBodyOffset&&
(b+=parseFloat(f.css(a,"marginTop"))||0,c+=parseFloat(f.css(a,"marginLeft"))||0);return{top:b,left:c},setOffset:function(a,b,c){var
d=f.css(a,"position");d==="static"&&(a.style.position="relative");var e=f(a),g=e.offset(),h=f.css(a,"top"),i=f.css(a,"left"),j=
(d==="absolute"||d==="fixed")&&f.inArray("auto",[h,i])>-1,k={},l={},m,n;j?(l=e.position(),m=l.top,n=l.left):
(m=parseFloat(h)||0,n=parseFloat(i)||0,f.isFunction(b)&&(b=b.call(a,g)),b.top!=null&&(k.top=b.top-g.top+m),b.left!=null&&(k.left=b.left-g.left+n),"using"
in
b?b.using(a,k):e.css(k)},f.fn.extend({position:function(){if([this[0]]return null;var a=this[0],b=this.offsetParent(),c=cx.test(b[0].nodeName)?
{top:0,left:0}:b.offset():c.top=parseFloat(f.css(a,"marginTop"))||0,c.left=
-parseFloat(f.css(a,"marginLeft"))||0,d.top+=parseFloat(f.css(b[0],"borderTopWidth"))||0,d.left+=parseFloat(f.css(b[0],"borderLeftWidth"))||0;return{top:c.top-
d.top,left:c.left-d.left},offsetParent:function(){return this.map(function(){var
a=this.offsetParent||c.body;while(a&&cx.test(a.nodeName)&&f.css(a,"position")=="static")a=a.offsetParent;return a})}),f.each(["Left","Top"],function(a,c)
{var d="scroll"+c;f.fn[d]=function(){var e,g;if(c==="Left")e=this[0];if(!e)return null;g=cy(e);return g?"pageXOffset":g?
g.pageYOffset:"pageXOffset"}]:f.support.boxModel&&g.documentElement.documentElement[d]:d) return this.each(function(){g=cy(this),g?
g.scrollTop||(f.g.scrollLeft:c.a?f.g.scrollTop()):this[d]):c});f.each(["Height","Width"],function(a,c){var d=c.toLowerCase();f.fn["inner"+c]=function(){var
a=this[0];return a?a.style.parseFloat(f.css(a,d,"padding")):this[d](0);},f.fn["outer"+c]=function(a){var b=this[0];return b?b.style?
parseFloat(f.css(b,d,"margin")):"border")):this[d](0);},f.fn[d]=function(a){var e=this[0];if(!e)return a==null?null:this;if(f.isFunction(a))return
this.each(function(b){var c=f(this);c[d](a.call(this,b,c[d]())));if(f.isWindow(e)){var g=e.documentElement.documentElement["client"+c],h=e.documentElement.body;return
e.documentElement.compatMode=="CSS1Compat"&&g||h&&h["client"+c][g]?f.e.nodeType==9?return
Math.max(e.documentElement["client"+c],e.body["scroll"+c],e.documentElement["offset"+c]),e.documentElement["offset"+c]:if(a==b){var
i=f.css(e,d),j=parseFloat(i);return f.isNumeric(j)?j:i}return this.css(d,typeof a=="string"?a+"px":{})),a.jQuery=a.\$=f.typeof
define=="function"&&!define.amd&&define.amd.jQuery&&define("jquery",[],function(){return f}))})(window);
81068. ! jquery v1.7.1 jquery.com | jquery.org/license
81069. function mbox() { alert('Javascript OK'); } var totalCount = 10; var pendingCount = 0; var errorCount = 0; var pushCount = 0; var allErrorCount = 0; var autoTest =
false; var testType = "cgi"; function NextTestType() { if (testType == "cgi") testType = "lp"; else if (testType == "lp") testType = "lua"; else testType = "cgi"; }
function runTest(method, isAsync) { ++pushCount; document.getElementById('start').innerHTML = 'Test: ' + pushCount;
document.getElementById('resTotal').innerHTML = 'running'; for (var i = 1; i <= totalCount; ++i) { document.getElementById('res'+i).innerHTML = "ready"; }
errorCount = 0; pendingCount = totalCount; for (var i = 1; i <= totalCount; ++i) { fetch(i, method, isAsync); } function runAutoTest() { if (autoTest) {
runTest("POST", true) setTimeout("runAutoTest()", 250) } } function fetch(id, method, isAsync) { document.getElementById('res'+id).innerHTML = "pending";
\$.ajax({ async: isAsync, url: 'echo.' + testType + '?id=' + id, type: method, timeout: 2000, data: { 'id': id, 'longText1':
"adfsdfasdklkjlgasfdfjkhq345sdafbmkanq3trsdghkjqw4etrjlkabsdfkabvauiregtlkjasdbvabl4btrjawebbfjsdhbjk342r5bjksdbfklijbhasdfbhj234qjhasdg76k11234jhv900ac
longText2":
"bsdfsfasdklkjlgasdfjkhq345sdafbmkanq3trsdghkjqw4etrjlkabsdfkabvauiregtlkjasdbvabl4btrjawebbfjsdhbjk342r5bjksdbfklijbhasdfbhj234qjhasdg76k11234jhv900ac
longText3":
"sdfsadagsdklkjlgasdfjkhq345sdafbmkanq3trsdghkjqw4etrjlkabsdfkabvauiregtlkjasdbvabl4btrjawebbfjsdhbjk342r5bjksdbfklijbhasdfbhj234qjhasdg76k11234jhv900a
longText4":
"q34sdfas3fhbkjlgasdfjkhq345sdafbmkanq3trsdghkjqw4etrjlkabsdfkabvauiregtlkjasdbvabl4btrjawebbfjsdhbjk342r5bjksdbfklijbhasdfbhj234qjhasdg76k11234jhv900a
longText5":
"askj2kjcvxychklgasdfjkhq345sdafbmkanq3trsdghkjqw4etrjlkabsdfkabvauiregtlkjasdbvabl4btrjawebbfjsdhbjk342r5bjksdbfklijbhasdfbhj234qjhasdg76k11234jhv900
longText6":
"asdfjklhlkjhv8Ã¶ajsdfjkhq345sdafbmkanq3trsdghkjqw4etrjlkabsdfkabvauiregtlkjasdbvabl4btrjawebbfjsdhbjk342r5bjksdbfklijbhasdfbhj234qjhasdg76k11234jhv900
'async' : isAsync }, dataType: 'json', succes: function(data) { }, error: function() { ++errorCount; }, complete: function(jqXHR, textStatus) { --pendingCount;
document.getElementById('res'+id).innerHTML = textStatus; console.log('id: ' + id + ' (' + pendingCount + '/' + totalCount + ')', status: ' + textStatus); if
(pendingCount == 0) { document.getElementById('resTotal').innerHTML = 'done'; console.log('complete, error count: ' + errorCount); allErrorCount =
allErrorCount + errorCount; document.getElementById('resAll').innerHTML = 'total errors: ' + allErrorCount + "/" + (pushCount*totalCount); } }); } // /
81070. -->
81071. !/bin/sh
81072. Read all data from stdin and send it to stdout
81073. Write the response header with \r\n
81074. Headline for generated reply:
81075. Disable until Warning returns to Travis CI or AppVeyor #pragma GCC diagnostic ignored "-Wunknown-pragmas" #pragma GCC diagnostic ignored "-Wno-variadic-macros" #pragma GCC diagnostic ignored "-Wreserved-id-macro"
81076. A default timeout for all tests running multiple requests to a server.
81077. All unit tests use the "check" framework. * Download from <https://libcheck.github.io/check/>
81078. **comment:** FIXME: check uses GCC specific variadic macros that are non-standard
label: code-design
81079. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and
associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge,
publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following
conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE
SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE *
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF

CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81080. A minimal timeout for all tests starting a server.

81081. Disable warnings for defining _CRT_SECURE_NO_ (here) and _CHECK_CHECK_STDINT_H (in check.h)

81082. A minimal timeout used for all tests with the "check" framework.

81083. TEST_CIVETWEB_CHECK_H

81084. **comment:** When using -Weverything, clang does not accept its own headers * in a release build configuration. Disable what is too much in * -Weverything.
label: code-design

81085. Unreferenced formal parameter. START_TEST has _i

81086. conditional expression is constant . asserts use while(0)

81087. Make the actual CORS request.

81088. XHR for Chrome/Firefox/Opera/Safari.

81089. Create the XHR object.

81090. Helper method to parse the title tag from the response.

81091. Response handlers.

81092. XDomainRequest for IE.

81093. CORS not supported.

81094. <http://www.html5rocks.com/en/tutorials/cors/>

81095. !/bin/sh

81096. Read in two pieces, to test continuation

81097. Copyright (c) 2004-2009 Sergey Lyubka Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Unit test for the civetweb web server. Tests embedded API.

81098. HTML forms can use GET or POST, and the encoding can be application/x-www-form-urlencoded or multipart/form-data. If no method is specified (like <form method="method">), GET should be the default method. If no encoding is specified, application/x-www-form-urlencoded should be the default. Submit buttons may overwrite action, method and enctype by using formaction, formmethod and formenctype. References: <http://www.w3.org/TR/html401/interact/forms.html> http://www.w3schools.com/html/html_forms.asp, http://www.w3schools.com/html/html_form_attributes.asp <http://www.w3.org/TR/html401/interact/forms.html#edef-enctype>

81099. www.w3.org/TR/html401/interact/forms.html www.w3schools.com/html/html_forms.asp, www.w3schools.com/html/html_form_attributes.asp www.w3.org/TR/html401/interact/forms.html#edef-enctype

81100. !/bin/sh

81101. !/bin/sh

81102. !/bin/sh

81103. !/bin/sh

81104. Determine what tests to run

81105. This unit test file uses the excellent Check unit testing library. * The API documentation is available here: * http://check.sourceforge.net/doc/check_html/index.html

81106. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81107. Write test logs to a file

81108. Run up the tests

81109. CK_NORMAL offers not enough diagnosis during setup phase

81110. Adapted from unit_test.c

81111. req3 is a complete and valid request

81112. Test helper function - adapted from unit_test.c

81113. "abcdcbdecdefdefgfgfghighjihkjlkjklmklmnlnomnopnqd"

81114. * * We include the source file so that we have access to the internal private * static functions

81115. Copyright (c) 2004-2013 Sergey Lyubka

81116. req12 is a valid request with body data

81117. invalid

81118. req1 minus 1 byte at the end is incomplete

81119. req10 is a valid request

81120. req1 is a valid request

81121. Multiline header are obsolete, so return an error * (<https://tools.ietf.org/html/rfc7230#section-3.2.4>).

81122. req5 is a complete and valid request (also somewhat malformed, * since it uses \n\n instead of \r\n\r\n)

81123. **comment:** len could be unused, if base64_decode is not tested because USE_LUA is * not defined81124. **label:** code-design

81124. This unit test file uses the excellent Check unit testing library. * The API documentation is available here: *

81124. http://check.sourceforge.net/doc/check_html/index.html

81125. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81126. req7 is invalid, but not yet complete

81127. a million "a" in blocks of 10

81128. Same is true for a leading space

81129. Used to debug test cases without using the check framework

81130. **comment:** TODO: adapt test for big endian81130. **label:** test

81131. req8 is a valid response

81132. **comment:** Pass small buffer, make sure alloc_printf allocates
label: code-design

81133. req2 is a complete, but invalid request

81134. a million "a"

81135. An empty string is neither a complete request nor a complete * response, so it must return 0

81136. Test alternative implementation

81137. is_valid_uri is superseeded by get_uri_type

81138. req11 is a complete but valid request

81139. length without body

81140. req9 is a valid response

81141. Valid request

81142. **comment:** TODO: check this for big endian
label: requirement

81143. **comment:** Can not test, if SHA1 is not included
label: test

81144. But a control character (like 0) makes it invalid

81145. req6 is incomplete

81146. req1 minus 1 byte at the start is complete but invalid

81147. Copyright (c) 2013-2015 the Civetweb developers

81148. test_parse_port_string requires WSASStartup for IPv6

81149. empty string

81150. "abc"

81151. part of "Thu, 01 Jan 1970 00:00:00 GMT"

81152. If the string is truncated, mg_snprintf calls mg_cry. * If DEBUG is defined, mg_cry calls DEBUG_TRACE. * In DEBUG_TRACE_FUNC, flockfile(stdout) is called. * For Windows, flockfile/funlockfile calls Enter-/ * LeaveCriticalSection(&global_log_file_lock). * So, we need to initialize global_log_file_lock:

81153. TEST_PRIVATE_H_

81154. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81155. This unit test file uses the excellent Check unit testing library. * The API documentation is available here: *
http://check.sourceforge.net/doc/check_html/index.html

81156. test_sdup

81157. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81158. This is required for "realpath". According to * <http://man7.org/linux/man-pages/man3/realpath.3.html> * defining _XOPEN_SOURCE 600 should be enough, but in * practice this does not work.

81159. main is already used in the test suite, * so this define will rename main in main.c

81160. * * We include the source file so that we have access to the internal private * static functions.

81161. TEST_PRIVATE_H_

81162. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81163. This is a redefine for "main" in main.c

81164. **comment:** TODO: mg_get_cookie and mg_get_var(2) should have the same behavior
label: code-design

81165. current version is 1.8

81166. zero size result buffer

81167. keys are found as first, middle and last key

81168. **comment:** ck_assert_int_eq(ret, -3); --> TODO: mg_get_cookie returns -3, mg_get_var -2. This should be unified.
label: code-design

81169. invalid result buffer

81170. key not found in string

81171. This unit test file uses the excellent Check unit testing library. * The API documentation is available here: *
http://check.sourceforge.net/doc/check_html/index.html

81172. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81173. space is valid

81174. a-z is valid

81175. check feature

81176. **comment:** TODO: this is the same situation as with mg_get_value
label: code-design

81177. longer value in the middle of a longer string - seccond occurance of key

81178. key with = but without value in the middle of a longer string
81179. A-Z is valid
81180. longer value in the middle of a longer string
81181. too small result buffer
81182. check structure of version string
81183. get system information
81184. hyphen is valid
81185. key without = and without value in the middle of a longer string
81186. mg_get_var call mg_get_var2 with last argument 0
81187. TEST_PUBLIC_H_
81188. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
81189. return 0 to accept every connection
81190. HTTP 1.1 GET request
81191. Read file before a .htpasswd file has been created
81192. This test should ensure the minimum server example in * docs/Embedding.md is still running.
81193. send some kilobyte of data
81194. TODO: fix path in CI test environment
81195. Can not send a websocket goodbye message here - * the connection is already closed
81196. Get data from callback using https://127.0.0.1
81197. Remove log files (they may exist from previous incomplete runs of * this test)
81198. TODO: ck_assert_str_eq(client_ri->request_method, "HTTP/1.0");
81199. Handle form: "POST x-www-form-urlencoded"
81200. Repeat test after .htpasswd is created
81201. Set options and start server
81202. Ask for something not existing - should get error404.htm
81203. Websocket test
81204. Elapsed time in ms - in most systems * only with second resolution
81205. www.github.com
81206. Stop the server
81207. Disconnect client 3
81208. Server start parameters for HTTPS
81209. HTTPS port - required
81210. Call a test client
81211. This unit test file uses the excellent Check unit testing library. * The API documentation is available here: * http://check.sourceforge.net/doc/check_html/index.html
81212. Wait for the websocket acknowledgement message
81213. defined(MG_USE_OPEN_FILE)
81214. set minimum SSL version to TLS 1.2 - recommended
81215. TODO: Check this problem on AppVeyor ck_assert_int_le(retry_fail_cnt, 2); ck_assert_int_ge(retry_ok_cnt, 1);
81216. Ask in a malformed way - should get error4xx.htm
81217. set "HTTPS only" header - recommended
81218. timing test
81219. Start server with bad options
81220. Try to copy the files required for AppVeyor
81221. Connect client 4
81222. not authorized
81223. **comment:** Stop the server and clean up
 label: code-design
81224. Handle form: "POST multipart/form-data" with chunked transfer encoding
81225. HTTP 1.1 GET request - must not work, since server is already stopped
81226. Client data
81227. return 1 to keep the connection open
81228. **comment:** Remove log files
 label: code-design
81229. e.g.: ck_assert_str_eq(client_ri->request_uri, "200");
81230. Return 1 means "already handled"
81231. strlen(websocket_goodbye_msg)
81232. Handle form: "POST multipart/form-data", chunked, store
81233. **comment:** Time estimation: Data size is 10 kB, with 1 kB/s speed limit. * The first block (1st kB) is transferred immediately, the second * block (2nd kB) one second later, the third block (3rd kB) two * seconds later, .. the last block (10th kB) nine seconds later. * The resolution of time measurement using the "time" C library * function is 1 second, so we should add +/- one second tolerance. * Thus, download of 10 kB with 1 kB/s should not be faster than * 8 seconds.
 label: code-design
81234. Free test data
81235. Response must be 505 HTTP Version not supported
81236. Error replies will close the connection, even if keep-alive is set.
81237. Get CGI generated data
81238. Try to access the file again. Expected: 401 error
81239. Response must be 400 Bad Request
81240. Try downloading several times
81241. full length string compare will reach limit in the implementation * of the check unit test framework
81242. Ask for something not existing - should get error4xx.htm
81243. Don't need to do this if mg_start failed
81244. Used to debug test cases without using the check framework
81245. Get directory listing
81246. Get data from callback
81247. Check error.log
81248. set some modern ciphers - recommended
81249. Get data from callback using mg_connect_client instead of mg_download
81250. Create an error4xx.htm file
81251. The check unit test library does not have build in memcmp functions.

81252. Start with default options
81253. It seems to be impossible to find out what the actual working * directory of the CI test environment is. Before breaking another * dozen of builds by trying blindly with different paths, just * create the file here
81254. File is in memory
81255. **comment:** TODO: add test for windows, check with POST
label: test
81256. *****
81257. Do not set this on Travis CI, since the build containers * contain older SSL libraries
81258. Build command for Linux: gcc test/public_server.c src/civetweb.c -I include/ -I test/ -l pthread -l dl -D LOCAL_TEST -D REPLACE_CHECK_FOR_LOCAL_DEBUGGING -D MAIN_PUBLIC_SERVER=main
81259. **comment:** TODO: the different paths * used in the different test * system is an unsolved * problem.
label: test
81260. Set to a number - but use a number above the limit
81261. Connect client 3
81262. chunks from 1 byte to strlen(boundary)-1
81263. File content check var
81264. Filename is required
81265. Response must be 200 OK
81266. Server var
81267. ck_assert_int_eq(i, (int)strlen(expected_cgi_result));
81268. **comment:** Remove invalid option
label: code-design
81269. Debug output for tests
81270. Handle form: "POST multipart/form-data", chunked, store, with files
81271. Get zipped static data - with Accept-Encoding
81272. assume there is enough space left in our typical 255 character string buffers
81273. Try to load non existing file
81274. Write long data (16 bit size header)
81275. Run the server for 15 seconds
81276. Get the current working directory
81277. Server context handle
81278. PUT to static file (will not work)
81279. We don't need the original anymore, the server has a private copy
81280. Get data from callback using https://[::1]
81281. Get data from callback using http://127.0.0.1
81282. strlen(websocket_welcome_msg)
81283. **comment:** TODO: A client API using a client certificate is missing
label: code-design
81284. stop the server without closing this connection
81285. Call the form handler
81286. Check the test dir
81287. First test with GET
81288. Handle form: "POST x-www-form-urlencoded", chunked, store
81289. HTTP 1.7 GET request - this HTTP version does not exist
81290. Initialize the library
81291. Start the server
81292. File is not in memory
81293. path to certificate file - required
81294. nonce is from including (auth_request + len) to excluding (str)
81295. Remove error files (in case they exist)
81296. Ask for something not existing - should get error.htm
81297. Not a valid HTTP response code, * but it should be written to the log and passed to * end_request.
81298. Add some handler
81299. Get redirect from callback using http://127.0.0.1
81300. CGI_ENVIRONMENT_SIZE
81301. Wait for the response
81302. Add large env field, so the server * must reallocate buffers.
81303. This time start the server with a valid configuration
81304. Create an error.htm file
81305. Retry DELETE with authorization of a user not authorized for DELETE
81306. Abort test with diagnostic message
81307. Ask for something not existing - should get default 404
81308. USE_WEBSOCKET
81309. Travis
81310. Read PUT response
81311. The Linux builds on Travis CI work fine with TLS1.2
81312. Send websocket welcome message
81313. HTTP request with multiline header. * Multiline header are obsolete with RFC 7230 section-3.2.4 * and must return "400 Bad Request"
81314. __MACH__
81315. Access to the file must work like before
81316. Get data from callback using mg_connect_client and absolute URI with a * sub-domain
81317. Check if there are at least 8 seconds
81318. **comment:** not required
label: requirement
81319. Create an error404.htm file
81320. WEBSOCKET SERVER
81321. This test should show a HTTPS server with enhanced * security settings. * * Articles: * <https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/> * * Scanners: * <https://securityheaders.io/> * <https://www.htbridge.com/ssl/> * <https://www.htbridge.com/websec/> * <https://www.ssllabs.com/ssltest/> * <https://www.qualys.com/forms/freescan/>
81322. test with unknown option
81323. Get zipped static data - will not work if Accept-Encoding is not set
81324. Check if CGI test executable exists
81325. **comment:** Return codes of the tested functions are evaluated. Checking all other * functions, used only to prepare the test environment seems redundant. * If they fail, the test fails anyway.
label: code-design
81326. Handle form: "POST multipart/form-data"
81327. HTTP 1.0 GET request
81328. Get data from callback using mg_connect_client and absolute URI

81329. For unknown reasons, there are sporadic hangs * for OSX if mark_point is called here
81330. Now connect a second client
81331. Create a default file in the document root
81332. Give the server some time to start in the test VM
81333. Won't get any message
81334. Then connect a first client
81335. Check access.log
81336. Closed
81337. Prepare test data
81338. Test with CGI test executable
81339. Now test form_store
81340. Un-initialize the library
81341. breakpoint here !
81342. Close the server
81343. **comment:** Copy the untruncated value, so string compare functions can be used.
 label: code-design
81344. method not allowed
81345. **comment:** TODO: request a file and keep alive * (will only work if NO_FILES is not set).
 label: code-design
81346. First store the length information in a text buffer.
81347. Create a .htpasswd file
81348. Server is running now
81349. **comment:** TODO: not yet available
 label: requirement
81350. Write long data (64 bit size header)
81351. Remove the user from the .htpasswd file again
81352. Wait for the websocket goodbye message
81353. sending megabytes to localhost does not always work in CI test * environments (depending on the network stack)
81354. send some strings that are almost boundaries
81355. Get static data
81356. Appveyor
81357. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
81358. Then send length information, chunk and terminating \r\n.
81359. Now remove the password file
81360. HTTP 1.0 GET request - server is not running
81361. Run the server for 5 seconds
81362. End test
81363. Handle form: "GET"
81364. WEBSOCKET CLIENT
81365. Get data from callback using http://[::1]
81366. **comment:** Test with bad num_thread option
 label: test
81367. Free data in ws_client4_conn
81368. end of options - required
81369. **comment:** Return 0 means "not yet handled"
 label: requirement
81370. Retry with authorization
81371. Not closed
81372. Send websocket goodbye message
81373. Handle form: "POST multipart/form-data" with chunked transfer * encoding, using a quoted boundary string
81374. connect client
81375. Wait for the websocket welcome message
81376. Remove all error files created by this test
81377. Error message detail may vary - it may not be empty and should contain * some information "connect" failed
81378. The Apple builds on Travis CI seem to have problems with TLS1.x * Allow SSLv3 and TLS
81379. Client var
81380. Nothing left to read
81381. Details of errmsg may vary, but it may not be empty
81382. Check the pem file
81383. POST to static file (will not work)
81384. initializer list here is rejected by CI test
81385. Close the client connection
81386. TEST_PUBLIC_H
81387. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.
81388. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF

CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81389. Copyright (c) 2015-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81390. TEST_SHARED_H_

81391. #exec "cd"

81392. #include "hello.txt"

81393. #include virtual="hello.txt"

81394. #include abspath="C:\Windows\system.ini"

81395. #exec "pwd"

81396. #include file=".hello.shtml"

81397. #include abspath="/etc/issue"

81398. #include file="hello.txt"

81399. Create a socket

81400. Connect to the server

81401. 3 --> 2, an absolute timer in the past (-123.456) will still * run once at start, and then with the period

81402. 3 --> 3, will not run at start

81403. Copyright (c) 2016-2017 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81404. 3 --> 1, an absolute timer in the past (-123.456) will still * run once at start, and then with the period

81405. If this test runs in a virtual environment, like the CI unit test * containers, there might be some timing deviations, so check the * counter with some tolerance.

81406. End timer thread

81407. 3 --> 2, because it is a single shot timer

81408. Sleep 1 second - timer will not run

81409. * * We include the source file so that we have access to the internal private * static functions

81410. Sleep 10 second - timers will run

81411. 3 --> 0, because it will run until c[1] = 0 and then stop

81412. Sleep 2 second - timers will not run

81413. 3 --> 2, will run at start, but no cyclic in 1 second

81414. Used to debug test cases without using the check framework

81415. Sleep 1 second - timer will run

81416. 3 --> 1, with 750 ms period, it will run once at start, * then once 750 ms later, but not 1500 ms later, since the * timer is already stopped then.

81417. return 0 here would be unreachable code

81418. Copyright (c) 2015 the Civetweb developers * * Permission is hereby granted, free of charge, to any person obtaining a copy * of this software and associated documentation files (the "Software"), to deal * in the Software without restriction, including without limitation the rights * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell * copies of the Software, and to permit persons to whom the Software is * furnished to do so, subject to the following conditions: * * The above copyright notice and this permission notice shall be included in * all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN * THE SOFTWARE.

81419. TEST_TIMER_H_

81420. !/usr/bin/env bash Create a tmp directory for the test to use

81421. measure fillseq with bunch of column families

81422. **comment:** measure overwrite performance

label: code-design

81423. measure readrandom with 6GB block cache

81424. **comment:** On the production build servers, set data and stat files/directories not in /tmp or else the tempdir cleaning scripts will make you very unhappy.

label: code-design

81425. Readwhilewriting

81426. measure readwhilewriting after load with filluniquerandom with 6GB block cache

81427. Seekrandomwhilewriting

81428. **comment:** measure memtable performance -- none of the data gets flushed to disk

label: code-design

81429. -eq 2];then

81430. send data to ods

81431. fill up the db for readrandom benchmark with filluniquerandom (1GB total size)

81432. running on devbox, just print out the values

81433. measure readrandom with 100MB block cache

81434. **comment:** measure overwrite performance with bunch of column families

label: code-design

81435. measure fillseq + fill up the DB for overwrite benchmark

81436. prepare a in-memory DB with 50M keys, total DB size is ~6G

81437. -eq 1];then

81438. measure readrandom with 6GB block cache and tailing iterator

81439. dummy test just to compact the data

81440. measure readrandom with 8k data in memtable

81441. !/usr/bin/env bash

81442. measure readrandom after load with filluniquerandom with 6GB block cache

81443. fill up the db for readrandom benchmark (1GB total size)

81444. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

81445. Have deleter which will erase item from cache, which will re-enter the cache at that point.

81446. release remaining 5 to keep valgrind happy
81447. cache is under capacity now since elements were released
81448. test1: increase capacity lets create a cache with capacity 5, then, insert 5 elements, then increase capacity to 10, returned capacity should be 10, usage=5
81449. the cache is over capacity since nothing could be evicted
81450. delete the key from the cache
81451. Add a bunch of light and heavy entries and then count the combined size of items still in the cache, which must be approximately the same as the total capacity.
81452. Insert item large than capacity to trigger eviction on empty cache.
81453. **comment:** Cleaning up all the handles
 label: code-design
81454. First release (i == 1) corresponds to Ref(), second release (i == 2) corresponds to Lookup(). Then, since all external refs are released, the below insertions should push out the cache entry.
81455. namespace rocksdb
81456. This tests that deleter is not called. When cache has free capacity it is not expected to immediately erase the released items.
81457. test3: init with flag being true.
81458. AS if the key have been inserted into cache but get evicted immediately.
81459. Frequently used entry must be kept around
81460. SUPPORT_CLOCK_CACHE
81461. test2: decrease capacity insert 5 more elements to cache, then release 5, then decrease capacity to 7, final capacity should be 7 and usage should be 7
81462. **comment:** cache is shared_ptr and will be automatically cleaned up.
 label: code-design
81463. namespace
81464. test1: set the flag to false. Insert more keys than capacity. See if they all go through.
81465. If i % 2 == 0, then the entry was unpinned before Lookup, so pinned usage increased
81466. the usage should be close to the capacity
81467. Check whether the entries inserted in the beginning are evicted. Ones without extra ref are evicted and those with are not.
81468. make sure the cache will be overloaded
81469. test insert without handle
81470. **comment:** release handles for pinned entries to prevent memory leaks
 label: code-design
81471. can no longer find in the cache
81472. Add entries. Unpin some of them after insertion. Then, pin some of them again. Check GetPinnedUsage().
81473. Guess what's in the cache now?
81474. This tests that deleter has been called
81475. Make sure eviction is triggered.
81476. Insert 5 entries, but not releasing.
81477. Insert n+1 entries, but not releasing.
81478. double cache size because the usage bit in block cache prevents 100 from being evicted in the first kCacheSize iterations
81479. a LRU Cache with n entries and one shard only
81480. release one handle
81481. test2: set the flag to true. Insert and check if it fails.
81482. Conversions between numeric keys/values and the types expected by Cache.
81483. Insert entries much more than Cache capacity
81484. element 0 is evicted and the rest is there This is consistent with the LRU policy since the element 0 was released first
81485. check that overloading the cache does not change the pinned usage
81486. insert a key and get two handles
81487. still can't find in cache
81488. make sure everything will be cached
81489. Low-pri entries will be inserted to head of low-pri pool.
81490. charge
81491. Low-pri entries can take high-pri pool capacity if available
81492. hash
81493. Low-pri entries will be inserted to head of low-pri pool after lookup.
81494. High-pri entries will be inserted to the head of the list after lookup.
81495. Allocate 2 cache entries to high-pri pool.
81496. deleter
81497. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
81498. namespace rocksdb
81499. High-pri entries can overflow to low-pri pool.
81500. handle
81501. **comment:** TODO: we also need to get the files of the latest commits. Get the most recently committed files.
 label: code-design
81502. **comment:** Find all gcno files to generate the coverage report
 label: test
81503. Fetch right version of gcov
81504. Generate the html report. If we cannot find lcov in this machine, we'll simply skip this step.
81505. **comment:** Parse the raw gcov report to more human readable form.
 label: code-design
81506. Unless otherwise specified, we'll not generate html report by default
81507. Exit on error.
81508. /usr/bin/env bash
81509. Write the output to both stdout and report file.
81510. write batch into TransactionDB
81511. put
81512. begin a transaction
81513. need a change to trigger a new backup
81514. Delete all keys we just ingested
81515. First run uses custom filter, second run uses bloom filter
81516. read from outside transaction, after commit
81517. Ok for uniqueness
81518. Reset the policy
81519. Custom merge operator
81520. Callback from rocksdb_writebatch_iterate()
81521. testing basic case with no iterate_upper_bound and no prefix_extractor
81522. testing iterate_upper_bound and forward iterator to make sure it stops at bound
81523. put outside a transaction
81524. bingo
81525. Create new empty database

81526. should stop here...
81527. Create new database
81528. normal not found
81529. ok
81530. Column families.
81531. close and destroy
81532. read from outside transaction, before commit
81533. iterate_upper_bound points beyond the last expected entry
81534. rollback
81535. Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
81536. !ROCKSDB_LITE
81537. If we do not compact here, then the lazy deletion of files (<https://reviews.facebook.net/D6123>) would leave around deleted files and the repair process will find those files and put them back into the database.
81538. open a TransactionDB
81539. Custom compaction filter
81540. snapshot
81541. Custom filter policy
81542. Simple sanity check that options setting db_paths work.
81543. should have filtered bar, but not foo
81544. reuse old transaction
81545. Must not find value when custom filter returns false
81546. wrong cf
81547. Simple sanity check that setting memtable rep works.
81548. Lite does not support C API
81549. start a new database from the checkpoint
81550. delete from outside transaction
81551. commit
81552. Can not use port/port.h macros as this is a c file
81553. Simple check cache usage
81554. Merge of a non-existing value
81555. delete
81556. iterate
81557. Force compaction
81558. Create database with vector memtable.
81559. Create database with hash skipList memtable.
81560. Ugh. Random seed of 0 used to produce no entropy. This code preserves the implementation that was in place when all of the magic values in this file were picked.
81561. [0, (1)] [0, (1), (2)] [(2)] [(1), 2, 3] [1]
81562. [0, (1)] [0, 1, (2)] [(2)] [1, 2, 3]
81563. Compare against saved keys
81564. disable flushing stale column families
81565. [0, (1)] [1]
81566. Write to log file D seqID 7 seqID 8
81567. [(1), 2, 3] [1, (0)], [(0)] [0]
81568. [0, 1]
81569. WaitForFlush() is not supported
81570. 1. check consistency 2. copy the logs from backup back to WAL dir. if the recovery happens again on the same log files, this should lead to incorrect results due to applying merge operator twice 3. check consistency
81571. delete obsolete files always
81572. Add three column families - one with no comparator and two with comparators specified
81573. GetProperty is not supported in lite
81574. RocksDBLite does not support GetDescriptor Verify the CF options of the returned CF handle.
81575. [0, (1)] [0, 1, (2)] [2]
81576. Write batch should be atomic.
81577. Since we didn't delete CF handle, RocksDB's contract guarantees that we're still able to read dropped CF
81578. 64KB
81579. GetLiveFilesMetaData is not supported
81580. base
81581. WaitForCompaction() is not supported in lite
81582. **comment:** SETUP column family "two" -- level style with 4 levels
 label: code-design
81583. Drop CF two
81584. 1MB should create ~10 files for each CF
81585. Not sure how easy it is to make this data driven. need to read back the WAL file and truncate last 10 entries
81586. Make sure Sanitize options sets arena_block_size to 1/8 of the write_buffer_size, rounded up to a multiple of 4k.
81587. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
81588. [0, (1)] [0, 1]
81589. iter == 0 -- no tailing iter == 2 -- tailing
81590. [(1), (2), 3] [(1), (0)] [(0)] [0, (1)] [1, (2)], [2]
81591. **comment:** Speed up threshold = min(4 * 2, 4 + (36 - 4)/4) = 8
 label: code-design
81592. Can't open without specifying default column family
81593. TEST functions are not supported in lite
81594. Sync points not supported in RocksDB Lite
81595. Hold super version.
81596. If it's the main thread hitting this sync-point, then it will be blocked until some other thread update the test_stage.
81597. 10 bytes for key, rest is value
81598. 10 bytes
81599. **comment:** [(1), (2), (3)] [(1), (0)] [(0)] [0, (1)] [1, (2)], [2, (3)] [3] delete obsolete logs --> [0, (1)] [1, (2)], [2, (3)] [3]
 label: code-design
81600. Add bunch more data to other CFs
81601. namespace
81602. off
81603. [(1), 2, 3] [(1), (0)] [(0)] [0, (1)] [1]
81604. Add more L0 files and force another manual compaction
81605. fill up the DB

81606. default argument means copy everything
81607. Tailing iterator not supported
81608. can't open dropped column family
81609. Create a database with four column families
81610. delete CF two
81611. This means a thread doing DropColumnFamily() is waiting for other thread to finish persisting options. In such case, we update the test_stage to unblock the main thread.
81612. **comment:** cleanup
label: code-design
81613. Return spread of files per level
81614. iter 0 -- drop CF, don't reopen iter 1 -- delete CF, reopen
81615. seqID 4 seqID 5
81616. **comment:** [(0), (1)] [(0), (1), (2)] [(2)] [(1), 2, 3] [1, (0)] [0] delete obsolete logs --> [(1), 2, 3] [1, (0)] [0]
label: code-design
81617. Each bracket is one log file. if number is in (), it means we don't need it anymore (it's been flushed) []
81618. SETUP column family "one" -- universal style
81619. Add a column family with no comparator specified
81620. test new iterators
81621. Log file A is not dropped after reopening because default column family's min log number is 0. It flushes to SST file X seqID 2 seqID 3 Current log file is file B now. While flushing, a new log file C is created and is set to current. Both CFs' min log number is set to file C so after flushing file B is deleted. Log file A still remains. Flushed to SST file Y.
81622. VERIFY compaction "one"
81623. **comment:** Speed up threshold = $\min(4 * 2, 4 + (12 - 4)/4) = 6$
label: code-design
81624. Before opening, there are four files: Log file A contains seqID 1 Log file C contains seqID 4, 5 SST file X contains seqID 1 SST file Y contains seqID 2, 3 Min log number: default CF: 0 CF one, two: C When opening the DB, all the seqID should be preserved.
81625. third flush. now, CF [two] should be detected as stale and flushed column family 1 should not be flushed since it's empty flush
81626. namespace rocksdb
81627. flush
81628. Flush the 0th column family to force a roll of the wal log
81629. no tailing
81630. Test that we can retrieve the comparator from a created CF
81631. 100KB
81632. Block flush jobs from running
81633. tailing
81634. Add a column family that doesn't support snapshot
81635. now we sleep again. this is just so we're certain that flush job finished
81636. Note that based on the test setting, this must not be the main thread.
81637. Disable on windows because SyncWAL requires env->IsSyncThreadSafe() to return true which is not so in unbuffered mode.
81638. !ROCKSDB_LITE
81639. GetSortedWalFiles is not supported
81640. setup options. all column families have max_write_buffer_number setup to 10 "default" -> 100KB memtable, start flushing immediately "one" -> 200KB memtable, start flushing with two immutable memtables "two" -> 1MB memtable, start flushing with three immutable memtables "three" -> 90KB memtable, start flushing with four immutable memtables
81641. not a multiple of 4k, round up 4k
81642. [0, (1)] [0, 1, 2]
81643. this test is placed here, because the infrastructure for Column Family test is being used to ensure a roll of wal files. Basic idea is to test that WAL truncation is being detected and not ignored
81644. WAIT for compactions
81645. Create an iterator holding the current super version.
81646. Cuckoo is not supported in lite
81647. Do n memtable flushes, each of which produces an sstable covering the range [small,large].
81648. copy the logs to backup
81649. Before opening, there are two logfiles: Log file A contains seqID 1 Log file D contains seqID 7, 8 Min log number: default CF: D CF one, two: D When opening the DB, log file D should be replayed using the seqID specified in the file.
81650. In this test, we generate enough files to trigger automatic compactions. The automatic compaction waits in NonTrivial:AfterRun We generate more files and then trigger an automatic compaction This will wait because the automatic compaction has files it needs. Once the conflict is hit, the automatic compaction starts and ends Then the manual will run and end.
81651. TRIGGER compaction "one"
81652. first iteration - dont reopen DB before dropping second iteration - reopen DB before dropping
81653. delete old files in backup_logs directory
81654. only 10 files in file cache
81655. assert consistency
81656. This test checks for automatic getting a conflict if there is a manual which has not yet been scheduled. The manual compaction waits in NotScheduled We generate more files and then trigger an automatic compaction This will wait because there is an unscheduled manual compaction. Once the conflict is hit, the manual compaction starts and ends Then another automatic will start and end.
81657. SETUP column family "one" -- level style
81658. this will trigger the flushes
81659. copy the logs from backup back to wal dir
81660. make sure all background compaction jobs can be scheduled
81661. [0]
81662. TEST functions used are not supported
81663. **comment:** ID 3 that was used for dropped column family "three" should not be reused
label: code-design
81664. Speedup threshold = $200 / 4 = 50$
81665. skipped as persisting options is not supported in ROCKSDB_LITE
81666. 3 files for default column families, 1 file for column family [two], zero files for column family [one], because it's empty
81667. anonymous namespace
81668. Flush jobs will close previous WAL files after finishing. By block flush jobs from running, we trigger a condition where the iterator destructor should close the WAL files.
81669. small write buffer size
81670. make sure that all files are deleted when we drop the column family
81671. Add more L0 files and force automatic compaction
81672. Add a column family that supports snapshot. Snapshot stays not supported.
81673. open only a subset of column families
81674. Generate log file A. seqID 1
81675. Log file A is not dropped after reopening because default column family's min log number is 0. It flushes to SST file X seqID 2 seqID 3 Current log file is file B now. While flushing, a new log file C is created and is set to current. Boths' min log number is set to file C in memory, so after flushing file B is deleted. At the

same time, the min log number of default CF is not written to manifest. Log file A still remains. Flushed to SST file Y.
81676. A flush will make `it` hold the last reference of its super version.
81677. recover the DB
81678. Deleting the iterator will clear its super version, triggering closing all files
81679. Allow both of flush and purge job to schedule.
81680. collect wal files
81681. TEST functions in DB are not supported in lite
81682. [0, (1)] [0, 1, (2)] [(2)] [2, 3]
81683. Return the value to associate with the specified key
81684. Start a thread that will drop the first column family and its comparator
81685. counts how many operations were performed
81686. Add some more entries
81687. **comment:** Make sure iterator created can still be used.
label: code-design
81688. this tests if max_column_family is correctly persisted with WriteSnapshot()
81689. ReadOnlyDB is not supported
81690. Required if inheriting from testing::WithParamInterface<>
81691. WaitForFlush() is not supported in lite
81692. Add another L0 file and force automatic compaction
81693. Take the last but one file, and truncate it
81694. TRIGGER compaction "two"
81695. Makes sure that obsolete log files get deleted
81696. Create an iterator holding the current super version, as well as the SST file just flushed.
81697. [0, (1)] [0, 1, (2)] [(2)] [2]
81698. WaitForFlush() is not supported Makes sure that obsolete log files get deleted
81699. VERIFY compaction "two"
81700. **comment:** GetSortedWalFiles is a flakey function -- it gets all the wal_dir children files and then later checks for their existence. if some of the log files doesn't exist anymore, it reports an error. it does all of this without DB mutex held, so if a background process deletes the log file while the function is being executed, it returns an error. We retry the function 10 times to avoid the error failing the test
label: test
81701. trigger compaction if there are >= 4 files
81702. Preserve file system state up to here to simulate a crash condition.
81703. seqID 4 seqID 5 seqID 6 Flushing all column families. This forces all CFs' min log to current. This is written to the manifest file. Log file C is cleared.
81704. namespace rocksdb
81705. Compact all L0 files using CompactFiles
81706. Add listener
81707. Small slowdown and stop trigger for experimental purpose.
81708. no assertion failure
81709. Another thread starts a compact files and creates an L0 compaction The background compaction then notices that there is an L0 compaction already in progress and doesn't do an L0 compaction Once the background compaction finishes, the compact files finishes
81710. IROCKSDB_LITE
81711. create couple files Background compaction starts and waits in BackgroundCallCompaction:0
81712. A class which remembers the name of each flushed file.
81713. Start compacting files.
81714. Add listener.
81715. verify all compaction input files are deleted
81716. **comment:** to trigger compaction more easily
label: code-design
81717. Always do full scans for obsolete files (needed to reproduce the issue).
81718. Create 5 files.
81719. Write one L0 file
81720. Disable RocksDB background compaction.
81721. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
81722. create couple files
81723. Make sure we can reopen the DB.
81724. In the meantime flush another file.
81725. c should invoke FullMerge due to kTypeValue at the beginning.
81726. Compaction should output just "a", "e" and "h" keys.
81727. keep skip to "f", aka keep skip to "g+"
81728. keep skip to "z"
81729. Same as ShuttingDownInFilter, but shutdown happens during filter call for a merge operand, not for a value.
81730. If there is a corruption after a single deletion, the corrupted key should be preserved.
81731. Last key the filter was called with.
81732. skip to "d+"
81733. namespace rocksdb
81734. Wait until the filter sees a key >= k and stalls at that key. If `exact`, asserts that the seen key is equal to k.
81735. Expects no merging attempts.
81736. Don't leave tombstones (kTypeDeletion) for filtered keys.
81737. Check that filter was never called again.
81738. b should invoke PartialMergeMulti with two operands.
81739. Unstall filter and wait for SeekToFirst() to return.
81740. Let key 1 through.
81741. See InitIterators() call below for the sequence of keys and their filtering decisions. Here we closely assert that compaction filter is called with the expected keys and only them, and with the right values.
81742. Compaction filter that gets stuck when it sees a particular key, then gets unstuck when told to. Always returns Decition::kRemove.
81743. Check that the compaction iterator did the correct sequence of calls on the underlying iterator.
81744. keep
81745. It is possible that the output of the compaction iterator is empty even if the input is not.
81746. Shutdown during compaction filter call for key 2.
81747. Filter will stall on key >= stall_at. Advance stall_at to unstall.
81748. See InitIterators() call below for why "c" is the only key for which FullMergeV2 should be called.
81749. a should invoke PartialMergeMulti with a single merge operand.
81750. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
81751. Destroy using last options
81752. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of

this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

81753. Send these L0 files to L1

81754. Whenever a compaction completes, this listener will try to verify whether the returned CompactionJobStats matches what we expect. The expected CompactionJobStats is added via AddExpectedStats().

81755. 4th Phase: perform L0 -> L1 compaction again, expect higher write amp When subcompactions are enabled, the number of output files increases by 1 because multiple threads are consuming the input and generating output files without coordinating to see if the output could fit into a smaller number of files like it does when it runs sequentially

81756. namespace rocksdb

81757. Insert some deletions for keys that don't exist that are both in and out of the key range

81758. time

81759. Do n memtable compactions, each of which produces an sstable covering the range [small,large].

81760. 3rd Phase: generate sparse L0 files (wider key-range, same num of keys)

81761. Stage 1: Generate several L0 files and then send them to L2 by using CompactRangeOptions and CompactRange(). These files will have a strict subset of the keys from the full key-range

81762. just enough setting to hold off auto-compaction.

81763. namespace

81764. A helper function which verifies whether two CompactionJobStats match. The verification of all compaction stats are done by ASSERT_EQ except for the total input / output bytes, which we use ASSERT_GE and ASSERT_LE with a reasonable bias --- 10% in uncompressed case and 20% when compression is used.

81765. 5th Phase: Do a full compaction, which involves in two sub-compactions. Here we expect to have 1 L0 files and 4 L1 files In the first sub-compaction, we expect L0 compaction.

81766. Here we treat one newly flushed file as an unit. For example, if a newly flushed file is 100k, and a compaction has 4 input units, then this compaction inputs 400k.

81767. 1st Phase: generate "num_L0_files" L0 files.

81768. Destroy it for not alternative WAL dir is used.

81769. Generates the expected CompactionJobStats for each compaction

81770. !ROCKSDB_LITE

81771. !defined(IOS_CROSS_COMPILE)

81772. 2nd Phase: perform L0 -> L1 compaction.

81773. Required if inheriting from testing::WithParamInterface<>

81774. compact two files into one in the last L0 -> L1 compaction

81775. This function behaves with the implicit understanding that two rounds of keys are inserted into the database, as per the behavior of the DeletionStatsTest.

81776. Once a compaction completed, this function will verify the returned CompactionJobInfo with the oldest CompactionJobInfo added earlier in "expected_stats_" which has not yet being used for verification.

81777. Note: key_base must be multiple of num_keys_per_L0_file

81778. Add an expected compaction stats, which will be used to verify the CompactionJobStats returned by the OnCompactionCompleted() callback.

81779. default cfd

81780. Whenever a compaction completes, this listener will try to verify whether the returned CompactionJobStats matches what we expect.

81781. interval needs to be >= 2 so that deletion entries can be inserted that are intended to not result in an actual key deletion by using an offset of 1 from another existing key

81782. Stage 4: Trigger compaction and verify the stats

81783. Return spread of files per level

81784. Verifies whether two CompactionJobStats match.

81785. Add a new record and flush so now there is a L0 file with a value too (not just deletions from the next step)

81786. make sure this is even

81787. Stage 3: Generate L0 files with some deletions so now there are files with the same key range in L0, L1, and L2

81788. An EventListener which helps verify the compaction results in test CompactionJobStatsTest.

81789. filter block index block

81790. Stage 2: Generate files including keys from the entire key range

81791. An EventListener which helps verify the compaction statistics in the test DeletionStatsTest.

81792. file size

81793. disallow trivial move

81794. 1/3 of the data will be updated.

81795. The following statement determines the expected smallest key based on whether it is a full compaction. A full compaction only happens when the number of flushes equals to the number of compaction input runs.

81796. !defined(IOS_CROSS_COMPILE)

81797. namespace rocksdb

81798. **comment:** This test documents the behavior where a corrupt key follows a deletion or a single deletion and the (single) deletion gets removed while the corrupt key gets written out. TODO(noetzli): We probably want a better way to treat corrupt keys.

label: code-design

81799. Tests three scenarios involving multiple single delete/put pairs: A: Put Snapshot SDel Put SDel -> Put Snapshot SDel B: Snapshot Put SDel Put SDel Snapshot -> Snapshot SDel Snapshot C: SDel Put SDel Snapshot Put -> Snapshot Put D: (Put) SDel Snapshot Put SDel -> (Put) SDel Snapshot SDel E: Put SDel Snapshot Put SDel -> Snapshot SDel F: Put SDel Put Sdel Snapshot -> removed G: Snapshot SDel Put SDel Put -> Snapshot Put SDel H: (Put) Put SDel Put Sdel Snapshot -> Removed I: (Put) Snapshot Put SDel Put SDel -> SDel J: Put Put SDel Put SDel SDel Snapshot Put Put SDel SDel Put -> Snapshot Put K: SDel SDel Put SDel Put SDel Put Snapshot SDel Put SDel SDel Put SDel -> Snapshot Put SDel L: SDel Put Del Put SDel Snapshot Del Put Del SDel Put SDel -> Snapshot SDel M: (Put) SDel Put Del Put SDel Snapshot Put Del SDel Del -> SDel Snapshot Del

81800. b does not appear because the operands are filtered

81801. ROCKSDB_LITE

81802. This is how the key will look like once it's written in bottommost file

81803. Because level 1 is not the last level, the sequence numbers of a and b cannot be set to 0

81804. Filtered

81805. Test multiple snapshots where the earliest snapshot is not a write-conflic-snapshot.

81806. Make "CURRENT" file that points to the new manifest file.

81807. **comment:** TODO(icanadi) Make it simpler once we mock out VersionSet

label: code-design

81808. namespace

81809. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

81810. returns expected result after compaction

81811. Filters merge operands with value 10.

81812. Test where all operands/merge results are filtered out.

81813. Level 0 files overlap

81814. If no file in L1 being compacted, L0->L1 compaction will be scheduled. being_compacted flag is cleared here.

81815. Tests if the files can be trivially moved in multi level universal compaction when allow_trivial_move option is set In this test as the input files doesn't overlaps, they should be trivially moved.

81816. num_levels - 2 is over target by 1100 + 200

81817. Level 1 size will be 10000 after merging with L0

81818. Universal and FIFO Compactions are not supported in ROCKSDB_LITE

81819. does not own FileMetaData

81820. A compaction should be triggered and pick file 2

81821. file_number
81822. Level 3 score slightly larger than 1
81823. Level 1 is over target by 200
81824. Level 4 exceeds target 1,000,000 by 900 after adding size from level 3 Size ratio L4/L3 is 9.9 After merge from L3, L4 size is 1000900
81825. Level 2 is less than target 10000 even added size of level 1
81826. Level 1 files overlap
81827. ROCKSDB_LITE
81828. num_levels - 3 is over target by 100 + 1000
81829. Tests if the files can be trivially moved in multi level universal compaction when allow_trivial_move option is set In this test as the input files overlaps, they cannot be trivially moved.
81830. Level 1 size will be 1400 after merging with L0
81831. This test exhibits the bug where we don't properly reset parent_index in PickCompaction()
81832. file_size
81833. Chain of overlapping user key ranges (forces ExpandWhileOverlapping() to expand multiple times)
81834. case 3.1: Higher levels (level 3) have overlap
81835. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
81836. case 3.3: Higher levels (level 5) have overlap, but it's only overlapping one key ("d")
81837. Locked file encountered when pulling in extra input-level files with same user keys. Verify we pick the next-best file from the same input level.
81838. output level should be the one above the bottom-most
81839. Locked file encountered when pulling in extra output-level files with same user keys. Expected to skip that compaction and pick the next-best choice.
81840. 6 L0 files, score 3.
81841. Level 2 score 1.8. File 7 is the largest. Should be picked
81842. File not overlapping
81843. Overlapping user keys
81844. Picking file 8 because overlapping ratio is the biggest.
81845. A compaction should be triggered and pick file 2 and 5. and it expands to file 1 and 3 too.
81846. No compaction should be scheduled, if L0 has higher priority than L1 but L0->L1 compaction is blocked by a file in L1 being compacted.
81847. L1 score more than 6.
81848. A compaction should be triggered and pick all files from level 1
81849. Level 3 exceeds target 100,000 of 1000
81850. largest
81851. score(L1) = 3.7 score(L2) = 1.85 There is no eligible file in L1 to compact since both candidates pull in file_number 5U, which overlaps with a file pending compaction (6U). The first eligible compaction is from L2->L3.
81852. Overlapping user keys on same level and output level
81853. Overlaps with file 28, 29, total size 521M
81854. namespace rocksdb
81855. verify whether compaction is needed based on the current size of L0 files.
81856. Merging to the second last level: $(5200 / 2100 + 1) * 1100$ Merging to the last level: $(50000 / 6300 + 1) * 1300$
81857. release the version storage
81858. Pick file 8 because it overlaps with 0 files on level 3.
81859. file 7 and 8 overlap with the same file, but file 8 is smaller so it will be picked. Overlaps with file 26, 27 Overlaps with file 27 Overlaps with file 28, but the file itself is larger. Should be picked.
81860. <- marked for compaction <- this one needs compacting
81861. Level 2 is less than target 10000 even added size of level 1 Size ratio of L2/L1 is 9600 / 1200 = 8
81862. case 3.2: Higher levels (level 5) have overlap
81863. estimated L1->L2 merge: $400 * (9100.0 / 1400.0 + 1.0)$
81864. Set Last level size 50000 num_levels - 1 target 5000 num_levels - 2 is base level with target 1000 (rounded up to max_bytes_for_level_base).
81865. verify the trigger given different number of L0 files.
81866. File 5 has to be included in the compaction
81867. Output level overlaps with the beginning and the end of the chain
81868. Picking file 7 because overlapping ratio is the biggest.
81869. grow the number of inputs in "level" without changing the number of "level+1" files we pick up Expand input level as much as possible no overlapping case
81870. case 1: Higher levels are empty
81871. Level 3 over the target, but since level 4 is empty, we assume it will be a trivial move.
81872. smallest_seq
81873. file_number 2U is largest and thus first choice. But it overlaps with file_number 1U which is being compacted. So instead we pick the next- biggest file, 3U, which is eligible for compaction.
81874. start a brand new version in each test.
81875. smallest
81876. This test checks ExpandWhileOverlapping() by having overlapping user keys ranges (with different sequence numbers) in the input files.
81877. If score in L1 is larger than L0, L1 compaction goes through despite there is pending L0 compaction.
81878. largest_seq
81879. Level 0 files don't overlap
81880. A compaction should be triggered and pick file 2 and 5. It can expand because adding file 1 and 3, the compaction size will exceed mutable_cf_options_.max_bytes_for_level_base.
81881. Level 1 score 1.2
81882. L1 total size 2GB, score 2.2, If one file being compacted, score 1.1.
81883. case 2: Higher levels have no overlap
81884. must return false when there's no files.
81885. input files to compaction process.
81886. **comment:** grow the number of inputs in "level" without changing the number of "level+1" files we pick up Expand input level as much as possible overlapping case
 label: code-design
81887. <- being compacted
81888. Overlaps with file 26, 27, total size 521M Overlaps with file 27, 28, total size 520M, the smalest overlapping
81889. Next
81890. Seek to random key
81891. Random walk and make sure iter and result_iter returns the same key and value
81892. Return the current option configuration.
81893. Destroy using last options
81894. put
81895. Measuring operations on DB (expect to be empty). source_strings are candidate keys
81896. Randomly generate source keys
81897. Seek to last
81898. Prev
81899. delete
81900. endptr

81901. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

81902. namespace

81903. Randomly generate 5 prefixes

81904. namespace rocksdb

81905. Seek to First

81906. Ugh. Random seed of 0 used to produce no entropy. This code preserves the implementation that was in place when all of the magic values in this file were picked.

81907. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

81908. Fill levels >= 1

81909. build 2 tables, flush at 5000

81910. if ((i % 100) == 0) fprintf(stderr, "@ %d of %d\n", i, n);

81911. Do it

81912. corrupt an index block of an entire file

81913. namespace rocksdb

81914. corrupt file size

81915. Force compactions by writing lots of values

81916. Write something. If sequence number was not recovered properly, it will be hidden by an earlier write.

81917. Pick file to corrupt

81918. Return the value to associate with the specified key

81919. Pick latest file

81920. WriteBatch tag for first record Somewhere in second block

81921. !ROCKSDB_LITE

81922. very big, we'll trigger flushes manually

81923. make unit test pass for now

81924. Do enough writing to force minor compaction

81925. Do not verify checksums. If we verify checksums then the db itself will raise errors because data is corrupted. Instead, we want the reads to be successful and this test will detect whether the appropriate corruptions have occurred.

81926. The 64 records in the first two log blocks are completely lost.

81927. if one write failed, every subsequent write must fail, too

81928. delete the file

81929. Return the ith key

81930. On Wndows OS Disk cache does not behave properly We do not call FlushBuffers on every Flush. If we do not close the log file prior to the corruption we end up with the first block not corrupted but only the second. However, under the debugger things work just fine but never pass when running normally For that reason people may want to run with unbuffered I/O. That option is not available for WAL though.

81931. Write must eventually fail because of corrupted table

81932. If LRU cache shard bit is smaller than 2 (or -1 which will automatically set it to 0), test SequenceNumberRecovery will fail, likely because of a bug in recovery code. Keep it 4 for now to make the test passes.

81933. Relative to end of file; make it absolute

81934. corrupts exactly one file at level 'level'. if no file found at level, asserts

81935. **comment:** one full file should be readable, since only one was corrupted the other file should be fully non-readable, since index was corrupted
label: code-design

81936. Create a big L0 file and check it compacts into multiple files in L1.

81937. namespace

81938. Write some keys using plain table.

81939. Write some keys using block based table.

81940. Generate one more file in level-0, and should trigger level-0 compaction

81941. namespace rocksdb

81942. Add some values to db.

81943. The following util methods are copied from plain_table_db_test.

81944. 100KB

81945. Write 28 values, each 10016 B ~ 10KB

81946. ROCKSDB_LITE

81947. Add more keys. Delete.

81948. Write 11 values, each 10016 B

81949. Try with empty DB first.

81950. Now add more keys and flush.

81951. Two SST files should be created, each containing 14 keys. Number of buckets will be 16. Total size ~156 KB.

81952. Return spread of files per level

81953. Write some keys using cuckoo table.

81954. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

81955. disallow trivial move

81956. Duplicate

81957. Update.

81958. Insert same key twice so that they go to different SST files. Then wait for compaction and check if the latest value is stored and old value removed.

81959. Now check keys in read only mode.

81960. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

81961. Does not exist, and create_if_missing == false: error

81962. Does exist, and error_if_exists == true: error

81963. Add a key

81964. Skip HashCuckooRep as it does not support single delete. FIFO and universal compaction do not apply to the test case. Skip MergePut because single delete does not get removed when it encounters a merge.

81965. **comment:** After a flush, "second", "third" and "fifth" should be removed
label: code-design

81966. CompactedDB

81967. Add more keys

81968. Block flush thread and disable compaction thread

81969. id1 should match id2 because identity was not regenerated

81970. second open should fail

81971. Compaction can still go through even if no thread can flush the mem table.

81972. Add more L0 files

81973. namespace rocksdb

81974. Insert can go through

81975. Full compaction

81976. generate one table with each type of checksum
81977. change when new checksum type added
81978. Unsupported case.
81979. Case 5: snapshot followed by a put followed by another Put Only the last put should remain.
81980. disable compaction
81981. Empty Key Set
81982. this will now also flush the last 2 writes
81983. skip HashCuckooRep as it does not support snapshot
81984. Case1: Delete followed by a put
81985. 10 bytes
81986. after we release the snapshot1, only two values left
81987. verify data with each type of checksum
81988. Fallback to read-only DB
81989. Case 3: Put followed by a delete
81990. This should trigger LogAndApply.
81991. MultiGet
81992. Empty Database, Empty Key Set
81993. Skip HashCuckooRep as it does not support single delete. FIFO and universal compaction do not apply to the test case. Skip MergePut because merges cannot be combined with single deletions.
81994. ROCKSDB_LITE
81995. After the current memtable is flushed, the DEL should have been removed
81996. Write two new keys
81997. Empty Database, Search for Keys
81998. Does not exist, and create_if_missing == true: OK
81999. It is possible to produce empty flushes when using single deletes. Tests whether empty flushes cause issues.
82000. Reopen and flush memtable.
82001. All other combinations are acceptable
82002. after we release the snapshot2, only one value should be left
82003. id1 should NOT match id3 because identity was regenerated
82004. Case 5: Put followed by snapshot followed by another Put Both puts should remain.
82005. four kv pairs * two bytes per value
82006. We have only one valid snapshot snapshot2. Since snapshot1 is not valid anymore, "first" should be removed by a compaction.
82007. 'foo' should be there because its put has WAL enabled.
82008. All entries (including duplicates) exist before any compaction or flush is triggered.
82009. Case 4: Put followed by another Put
82010. Try with both a short key and a long key
82011. Flush can still go through.
82012. **comment:** On Windows you can have either memory mapped file or a file with unbuffered access. So this asserts and does not make sense to run
 label: code-design
82013. 1 L0 file, use CompactedDB if max_open_files = -1
82014. clear database
82015. check if a new manifest file got inserted or not.
82016. Does exist, and error_if_exists == false: OK
82017. Case 2: Delete followed by another delete
82018. 05
82019. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82020. 11
82021. 12
82022. kAllTiers
82023. Should be able to write kTypeBlobIndex to memtables and SST files.
82024. Avoid values from being purged.
82025. Disable auto flushes.
82026. Normal iterator
82027. value_found
82028. 16
82029. namespace rocksdb
82030. Avoid blob values from being purged.
82031. SeekForPrev
82032. Next
82033. 02
82034. allow_blob
82035. Verify normal value
82036. fill data
82037. Iterator with blob support
82038. Get should be able to return blob index if is_blob_index is provided, otherwise return Status::NotSupported status.
82039. Prev
82040. Iterator should get blob value if allow_blob flag is set, otherwise return Status::NotSupported status.
82041. Verify blob index
82042. !ROCKSDB_LITE
82043. 10
82044. 04
82045. kTypeBlobIndex is a value type used by BlobDB only. The base rocksdb should accept the value type on write, and report not supported value for reads, unless caller request for it explicitly. The base rocksdb doesn't understand format of actual blob index (the value).
82046. 13
82047. 03
82048. 08
82049. 14
82050. Get should NOT return Status::NotSupported if blob index is updated with a normal value.
82051. 06
82052. 01
82053. 00
82054. 07
82055. 15
82056. 09
82057. Seek
82058. small write buffer

82059. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

82060. Make sure filter block is in cache.

82061. flush all data from memtable so that reads are from block cache

82062. read and cache data block

82063. Clear strict capacity limit flag. This time we shall hit compressed block cache.

82064. namespace rocksdb

82065. options.compression = kNoCompression;

82066. Release iterators and access cache again.

82067. compressed doesn't have any hits since blocks are not compressed on storage

82068. anonymous namespace

82069. adding data block

82070. cache evicted old index and block entries

82071. both block cache and compressed cache, but DB is not compressed also, make block cache sizes bigger, to trigger block cache hits

82072. high compression ratio

82073. set the cache capacity to the current usage

82074. index/filter blocks added to block cache right after table creation.

82075. A mock cache wraps LRU Cache, and record how many entries have been inserted for each priority.

82076. Make sure index block is in cache.

82077. Set a small enough block size so that each key-value get its own block.

82078. check that we triggered the appropriate code paths in the cache

82079. Load blocks into cache.

82080. SNAPPY

82081. Write 8MB (80 values, each 100K)

82082. Miss count should remain the same.

82083. ROCKSDB_LITE

82084. no block cache, only compressed cache

82085. Totally 3 files created up to now

82086. Make sure that when options.block_cache is set, after a new table is created its index/filter blocks are added to block cache.

82087. Data block should be inserted with low priority.

82088. Run this test three iterations. Iteration 1: only a uncompressed block cache Iteration 2: only a compressed block cache Iteration 3: both block cache and compressed cache Iteration 4: both block cache and compressed cache, but DB is not compressed

82089. default column family doesn't have block cache

82090. After disabling options.paranoid_file_checks. NO further block is added after generating a new file.

82091. Create a new SST file. This will further trigger a compaction and generate another file.

82092. Access data block.

82093. both compressed and uncompressed block cache

82094. Compressed block cache cannot be pinned.

82095. 200 bytes are enough to hold the first two blocks

82096. only uncompressed block cache

82097. only index/filter were added

82098. Test with strict capacity limit.

82099. Create a new table.

82100. Create a new table

82101. Set strict capacity limit flag. Now block will only load into compressed block cache.

82102. DB tests related to bloom filter.

82103. Full Filter Block

82104. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

82105. 11 RAND I/Os

82106. Restart with only prefix bloom is allowed.

82107. Lookup present keys. Should rarely read from either sstable.

82108. Now that we know the filter blocks exist in the last level files, see if filter caching is skipped for this optimization

82109. Needs insert some keys to make sure files are not filtered out by key ranges.

82110. Reopen with the same setting: only whole key is used

82111. sanity checks

82112. GROUP 1

82113. Try to create a DB with mixed files:

82114. In this case needs insert some keys to make sure files are not filtered out by key ranges.

82115. Generate randomly shuffled keys, so the updates are almost random.

82116. Check filter block ignored for files preloaded during DB::Open()

82117. namespace rocksdb

82118. public testing::WithParamInterface<bool> {

82119. ChangeCompactOptions() only changes compaction style, which does not trigger reset of table_factory

82120. **comment:** Lookup present keys. Should rarely read from small sstable.
label: code-design

82121. Part 2 (read path): rewrite last level with blooms, then verify they get cached only if !optimize_filters_for_hits

82122. Write DB with only full key filtering.

82123. Create with full filter

82124. In the current implementation partitioned filters depend on partitioned indexes

82125. **comment:** KeyMayExist function only checks data in block caches, which is not used by plain table format.
label: code-design

82126. Check db with partitioned full filter

82127. index and data block

82128. prefix will be x????

82129. Block Filter Block

82130. GROUP 0

82131. GROUP 2

82132. Reopen with both of whole key off and prefix extractor enabled. Still no bloom filter should be used.

82133. namespace

82134. Now we have two files: File 1: An older file with prefix bloom. File 2: A newer file with whole bloom filter.

82135. move to level 1

82136. Restart with both filters are allowed

82137. With partitioned filter we read one extra filter per level per each missed read.

82138. Populate multiple layers

82139. 1 Insert 2 K-V pairs into DB 2 Call Get() for both keys - expect memtable bloom hit stat to be 2 3 Call Get() for nonexistent key - expect memtable bloom miss stat to be 1 4 Call Flush() to create SST 5 Call Get() for both keys - expect SST bloom hit stat to be 2 6 Call Get() for nonexistent key - expect SST bloom miss stat to be 1 Test both: block and plain SST

82140. not supported in plain table

82141. end of while

82142. Check SST bloom stats

82143. assert that no new files were opened and no new blocks were read into block cache.

82144. Was not added to filter but rocksdb will try to read it from the filter

82145. Required if inheriting from testing::WithParamInterface<>

82146. Same scenario as in BloomStatsTest but using an iterator

82147. ROCKSDB_LITE

82148. File 1 will has it filtered out. File 2 will not, as prefix `foo` exists in the file.

82149. Check db with block_based filter

82150. Reopen with whole key filtering enabled and prefix extractor NULL. Bloom filter should be off for both of whole key and prefix bloom.

82151. Without block cache, we read an extra partition filter per each level*read and a partition index per each read

82152. A wrapped bloom over default FilterPolicy

82153. not in DB

82154. Check db with full filter

82155. setting > -1 makes it not preload all files

82156. Check if filter is useful

82157. **comment:** KeyMayExist can lead to a few false positives, but not false negatives. To make test deterministic, use a much larger number of bits per key-20 than bits in the key, so that false positives are eliminated
label: code-design

82158. Create with block based filter

82159. Add a large key to make the file contain wide range

82160. Check filter block ignored for file trivially-moved to bottom level

82161. No Level 0 file. Create one.

82162. Try to create a DB with mixed files.

82163. check memtable bloom stats

82164. Check filter block not cached for iterator

82165. Seek to key that was not in Domain

82166. Not in domain, wont be added to filter

82167. check SST bloom stats

82168. Check if they can be found

82169. column family index

82170. disallow trivial move

82171. Generate 11 sst files with the following prefix ranges. GROUP 0: [0,10] (level 1) GROUP 1: [1,2], [2,3], [3,4], [4,5], [5, 6] (level 0) GROUP 2: [0,6], [0,7], [0,8], [0,9], [0,10] (level 0) A seek with the previous API would do 11 random I/Os (to all the files). With the new API and a prefix filter enabled, we should only do 2 random I/O, to the 2 files containing the key.

82172. db configs

82173. Prevent auto compactions triggered by seeks

82174. Now we have three sorted run, L0, L5 and L6 with most files in L6 have no bloom filter. Most keys be checked bloom filters twice.

82175. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

82176. overwrite all the 100K keys once again.

82177. All the files are in the lowest level. Verify that all but the 100001st record has sequence number zero. The 100001st record is at the tip of this snapshot and cannot be zeroed out.

82178. Put some data.

82179. create a new database with the compaction filter in such a way that it deletes all keys

82180. Compaction filters should only be applied to records that are newer than the latest snapshot. However, if the compaction filter asks to ignore snapshots records newer than the snapshot will also be processed

82181. namespace rocksdb

82182. empty db

82183. push all files to the highest level L2. This means that all keys should pass at least once via the compaction filter

82184. write all the keys once again.

82185. push all files to lower levels

82186. Number of skips in tables: 2, 3, 3, 3.

82187. this will produce empty file (delete compaction filter)

82188. Make sure next file is much smaller so automatic compaction will not be triggered.

82189. Filter out keys with value is 2.

82190. Scan the entire database to ensure that nothing is left

82191. Compaction filter never applies to merge keys.

82192. The filter should delete 10 records.

82193. Write 100K+1 keys, these are written to a few files in L0. We do this so that the current snapshot points to the 100001 key. The compaction filter is not invoked on keys that are visible via a snapshot because we anyways cannot delete it.

82194. put some data

82195. Delete Filter Factory which ignores snapshots

82196. verify that all keys now have the new value that was set by the compaction process.

82197. The filter should delete 40 records.

82198. The sequence number of the remaining record is not zeroed out even though it is at the level Lmax because this record is at the tip

82199. In the same compaction, both of value type and merge type keys need to be deleted based on compaction filter, and there is a merge type for the key. For both keys, compaction filter results are ignored.

82200. ROCKSDB_LITE

82201. Scan the entire database as of the snapshot to ensure that nothing is left

82202. We have deleted 10 keys from 40 using the compaction filter Keys 6-9 before the snapshot and 100-105 after the snapshot

82203. This is a static filter used for filtering kvs during the compaction process.

82204. Key ranges in tables are [0, 38], [106, 149], [212, 260], [318, 371].

82205. re-write all data again

82206. Push all files to the highest level L2. Verify that the compaction is each level invokes the filter for all the keys in that level.

82207. In the same compaction, a value type needs to be deleted based on compaction filter, and there is a merge type for the key. compaction filter result is ignored.

82208. Force a manual compaction

82209. value key can be deleted based on compaction filter, leaving only merge keys.

82210. Write 100K keys, these are written to a few files in L0.

82211. Write several keys.

82212. Compaction filters should only be applied to records that are newer than the latest snapshot. This test inserts records and applies a delete filter.

82213. Skip x if floor(x/10) is even, use range skips. Requires that keys are zero-padded to length 10.

82214. Push all files to the highest level L2. This triggers the compaction filter to delete all keys, verify that at the end of the compaction process, nothing is left.

82215. snapshots
82216. push all files to lower levels. This should invoke the compaction filter for all 100000 keys.
82217. Verify total number of keys is correct after manual compaction.
82218. Release the snapshot and compact again -> now all records should be removed.
82219. Tests the edge case where compaction does not produce any output -- all entries are deleted. The compaction should create bunch of 'DeleteFile' entries in VersionEdit, but none of the 'AddFile's.
82220. skipped
82221. must have much smaller db size.
82222. filter out cases for table properties queries.
82223. Wait for compaction to finish
82224. Second non-trivial compaction is triggered
82225. **comment:** Create two more files for one column family, which triggers speed up condition, three compactions will be scheduled.
 label: code-design
82226. Since data is non-overlapping we expect compaction to initiate a trivial move
82227. 500KB
82228. 8 MB
82229. Verify all L0 files are still there.
82230. Multiple files in L0 No files in L1
82231. 2 files in L2
82232. b.smallestkey <= a.smallestkey <= b.largestkey
82233. If update stats on DB::Open is disable, we don't expect deletion entries taking effect.
82234. level1_file.name
82235. (1, 4, 1)
82236. Eventually the cancelled compaction will be rescheduled and executed.
82237. This test triggers move compaction and verifies that the file is not deleted when it's part of move compaction
82238. github issue #2249
82239. Another 110KB triggers a compaction to 400K file to fill up first path
82240. iter 1 -- delete_obsolete_files_period_micros == 0
82241. Create 1MB sst file
82242. block compactions
82243. Add 2 non-overlapping files
82244. Delete keys in range [1, 4]. These L0 files will be compacted with L1: - Tombstones for keys 2 and 4 can be dropped early. - Tombstones for keys 1 and 3 must be kept due to L2 files' key-ranges.
82245. Stage 1: generate a db with level compaction
82246. Reopen the DB with stats-update disabled
82247. level1_files
82248. file 5 [400 => 500)
82249. make sure all key-values are still there.
82250. stop the compaction thread until we simulate the file creation failure.
82251. Stage 2: reopen with universal compaction - should fail
82252. Trivial move the two non-overlapping files to level 6
82253. L0 file meta
82254. this range overlap with the next one
82255. One for compaction input, one for verifying compaction results.
82256. compaction options
82257. Compact all
82258. regression test for issue #2722: L0->L0 compaction can resurrect deleted keys from older L0 files if L1+ files' key-ranges do not include the key.
82259. Compaction will do L0=>L1 L1=>L2 L2=>L3 (3 trivial moves)
82260. Same ranges as above but overlapping
82261. Create new iterator for: (1) 1 for verifying flush results (2) 3 for compaction input files (3) 1 for verifying compaction results.
82262. First three 110KB files are not going to second path. After that, (100K, 200K)
82263. disallow trivial move
82264. File moved From L0 to L1 0 files in L0 1 file in L1
82265. Push data from level 0 to level 1 to force all data to be deleted Note that we don't delete level 0 files
82266. release snap so that first instance of key(3) can have seqId=0
82267. File with keys [100 => 199]
82268. Trigger a long memtable compaction and reopen the database during it Goes to 1st log file Fills memtable Triggers compaction Goes to new log file
82269. Compact just the new range
82270. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82271. create two files in l1 that we can compact
82272. index: 0 1 2 3 4 5 6 7 8 9 size: 1MB 1MB 1MB 1MB 1MB 2MB 1MB 1MB 1MB 1MB score: 1.5 1.3 1.5 2.0 inf Files 0-4 will be included in an L0->L1 compaction. L0->L0 will be triggered since the sync points guarantee compaction to base level is still blocked when files 5-9 trigger another compaction. Files 6-9 are the longest span of available files for which work-per-deleted-file decreases (see "score" row above).
82273. note CompactionOptions::output_file_size_limit is unset.
82274. trigger compaction when we have 2 files
82275. this time we're expecting significant drop in size.
82276. round 3 --- reopen db with auto_compaction on and see if deletion compensation still work.
82277. 1 files in L0
82278. Make sure the number of L0 files can trigger compaction.
82279. Only 1 file in L0
82280. this file should have been compacted away
82281. each value is 10K
82282. The auto compaction is scheduled but waited until here
82283. make l0 files' ranges overlap to avoid trivial move
82284. 10 KB
82285. (1, 4, 4)
82286. key 0 serves both to prevent trivial move and as the key we want to verify is not resurrected by L0->L0 compaction.
82287. the following only disable stats update in DB::Open() and should not affect the result of this test.
82288. Otherwise, we should see a significant drop in db size.
82289. file 3 [0 => 200]
82290. a.smallestkey < b.smallestkey <= a.largestkey
82291. **comment:** Disable as the test is flaky.
 label: test
82292. Verify number of compactions allowed will come back to 1.
82293. Expect compaction to fail here as one file will fail its creation.
82294. (1, 4, 7)

82295. 2 files in L2, 1 in L1
82296. fill up the DB
82297. Compaction will do L0=>L1 L1=>L2 L2=>L3 (3 trivial moves) then compacte the bottommost level L3=>L3 (non trivial move)
82298. 1 MB
82299. lookup iterator from table cache and no need to create a new one.
82300. Compaction will do L0=>L1 (trivial move) then move L1 files to L3
82301. Unblock all threads to unblock all compactions.
82302. move both files down to l1
82303. SYNC_POINT is not supported in released Windows mode.
82304. File with keys [0 => 99]
82305. this should execute L1->L2 (move)
82306. 110KB
82307. Create two 1MB sst files
82308. First non-trivial compaction is triggered
82309. Check level compaction with compact files
82310. Verify again after reopen.
82311. We expect that all the files were trivially moved from L0 to L1
82312. at least L0 and L1 L0 has the 2MB file (not compacted) and 4MB file (output of L0->L0)
82313. Preloading iterator issues one table cache lookup and creates a new table reader. One file is created for flush and one for compaction. Compaction inputs make no table cache look-up for data/range deletion iterators
82314. Create several column families. Make compaction triggers in all of them and see number of compactions scheduled to be less than allowed.
82315. trigger L0 compaction
82316. Verify level sizes
82317. Infinitely large
82318. second non-trivial compaction
82319. **comment:** Now all column families qualify compaction but only one should be scheduled, because no column family hits speed up condition.
 label: code-design
82320. Delete archival files.
82321. **comment:** Currently, the test relies on the number of calls to InputCompressionMatchesOutput() per compaction.
 label: code-design
82322. **comment:** This tests for a bug that could cause two level0 compactions running concurrently TODO(aekmekji): Make sure that the reason this fails when run with max_subcompactions > 1 is not a correctness issue but just inherent to running parallel L0-L1 compactions
 label: code-design
82323. index: 0 1 2 3 4 5 6 7 8 9 size: 1MB score: 1.25 1.33 1.5 2.0 inf Files 0-4 will be included in an L0->L1 compaction. L0->L0 will be triggered since the sync points guarantee compaction to base level is still blocked when files 5-9 trigger another compaction. All files 5-9 are included in the L0->L0 due to work-per-deleted file decreasing. Put a key-value in files 0-4. Delete that key in files 5-9. Verify the L0->L0 preserves the deletion such that the key remains deleted.
82324. this should execute both L0->L1 and L1->L2 (merge with previous file)
82325. round 1 --- insert key/value pairs.
82326. file 4 [300 => 400)
82327. All the TEST_P tests run once with sub_compactions disabled (i.e. options.max_subcompactions = 1) and once with it enabled
82328. (1, 4, 3)
82329. create first file and flush to l0
82330. nowait
82331. create second file and flush to l0
82332. file 2 [100 => 300]
82333. iterator is holding the file
82334. create 3 files in l0 so to trigger compaction
82335. Only verifying compaction outputs issues one table cache lookup for both data block and range deletion block).
82336. !defined(ROCKSDB_LITE) namespace rocksdb
82337. at least L0 and L1 L0 has a single output file from L0->L0
82338. Compaction range falls after files
82339. put key 1 and 3 in separate L1, L2 files. So key 0, 2, and 4+ fall outside these levels' key-ranges.
82340. Compaction will initiate a trivial move from L0 to L1
82341. (1, 4, 6)
82342. 2 files in L2, 1 in L0
82343. Reopening moves updates to level-0
82344. Compaction will do L0=>L1 L1=>L2 L2=>L3 (3 trivial moves) and will skip bottommost level compaction
82345. Fail the first file creation.
82346. third pass with universal compaction
82347. non overlapping ranges
82348. let compactions go
82349. Identifies all files between level "min_level" and "max_level" which has overlapping key range with "input_file_meta".
82350. (1, 4, 2)
82351. All key-values must exist after compaction fails.
82352. Since we do a normal stats update on db-open, there will be more random open files.
82353. As stats-update is disabled, we expect a very low number of random file open. Note that this number must be changed accordingly if we change the number of files needed to be opened in the DB::Open process.
82354. level1_file.size
82355. Many files 4 [300 => 4300)
82356. prevents trivial move
82357. file 1 [0 => 300]
82358. The auto compaction will wait until the manual compaction is registered before processing so that it will be cancelled.
82359. second pass with universal compaction
82360. file 1 [0 => 100]
82361. Large write buffer
82362. put extra key to trigger flush
82363. Make sure RocksDB will not get into corrupted state.
82364. First three 110KB files are going to level 0 After that, (100K, 200K)
82365. Verify manual compaction doesn't fill block cache
82366. This test verify UpdateAccumulatedStats is not on if options.skip_stats_update_on_db_open = true The test will need to be updated if the internal behavior changes.
82367. make sure all background compaction jobs can be scheduled
82368. Compaction range falls before files
82369. Stage 4: re-open in universal compaction style and do some db operations
82370. Another 110KB triggers a compaction to 400K file to fill up level 0
82371. Populate a different range
82372. Write 100KB (100 values, each 1K)

82373. Check that writes done during a memtable compaction are recovered if the database is shutdown during the memtable compaction.
82374. Create two more 1MB sst files
82375. options = CurrentOptions(options);
82376. Block all threads in thread pool.
82377. Verify L0 -> L1 compaction does fail.
82378. Repeat the reopen process, but this time we enable stats-update.
82379. insert relatively small amount of data to trigger auto compaction.
82380. 2 files in L6, 1 file in L5
82381. Make sure all overlapping files do not exist after compaction
82382. **comment:** Should speed up compaction when there are 4 files.
 label: code-design
82383. 2 files in L6
82384. 2 files in L0
82385. anonymous namespace
82386. 200KB
82387. b.smallestkey <= a.largestkey <= b.largestkey
82388. round 2 --- disable auto-compactions and issue deletions.
82389. Before two non-trivial compactions are installed, there are 3 files in L0
82390. (1, 4)
82391. file 2 [600 => 700]
82392. Purpose of dependencies: 4 -> 1: ensure the order of two non-trivial compactions 5 -> 2 and 5 -> 3: ensure we do a check before two non-trivial compactions are installed
82393. **comment:** First two levels have no compression, so that a trivial move between them will be allowed. Level 2 has Zlib compression so that a trivial move to level 3 will not be allowed
 label: code-design
82394. file 6 [500 => 600)
82395. (1, 4, 8)
82396. sentinel to prevent trivial move
82397. a.smallestkey <= b.largestkey < a.largestkey
82398. Required if inheriting from testing::WithParamInterface<>
82399. After two non-trivial compactions are installed, there is 1 file in L6, and 1 file in L1
82400. Write 8MB (80 values, each 100K)
82401. Generate four files in CF 0, which should trigger an auto compaction
82402. verify keys inserted in both level compaction style and universal compaction style
82403. Try deleting files in range which contain no keys
82404. Always gets compacted into 1 Level1 file, 0/1 Level 0 file
82405. preloading iterator issues one table cache lookup and create a new table reader.
82406. iter - 0 with 7 levels iter - 1 with 3 levels
82407. 1 file in L0 0 files in L1
82408. 100 KB
82409. File with keys [200 => 299]
82410. Reopening moves updates to L0
82411. Stage 3: compact into a single file and move the file to level 0
82412. this should execute L0->L1
82413. Compaction range overlaps files
82414. (1, 4, 5)
82415. Deletions can be dropped when compacted to non-last level if they fall outside the lower-level files' key-ranges.
82416. generate one more file in level-0, and should trigger level-0 compaction
82417. as auto_compaction is off, we shouldn't see too much reduce in db size.
82418. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82419. Initial base level is the last level
82420. Base level is not level 1
82421. Test compact range works
82422. namespace rocksdb
82423. Write about 40K more
82424. Verify DB
82425. A manual compaction will trigger the base level to become L2 Keep Writing data until base level changed 2->1. There will be L0->L2 compaction going on at the same time.
82426. Trigger a condition that the compaction changes base level and L0->Lbase happens at the same time. We try to make last levels' targets to be 40K, 160K, 640K, add triggers another compaction from 40K->160K.
82427. Put about 28K to L0
82428. Wait for 200 milliseconds before proceeding compactions to make sure two parallel ones are executed.
82429. Put about 7K to L0
82430. **comment:** Use InMemoryEnv, or it would be too slow.
 label: code-design
82431. Introduction of SyncPoint effectively disabled building and running this test in Release build. which is a pity, it is a good test
82432. Trigger parallel compaction, and the first one would change the base level. Hold compaction jobs to make sure
82433. All data should be in the last level.
82434. Issue manual compaction in one thread and still verify DB state in main thread.
82435. Insert extra about 28K to L0. After they are compacted to L4, base level should be changed to L3.
82436. Test specific cases in dynamic max bytes
82437. **comment:** Without it, valgrind may choose not to give another thread a chance to run before finishing the function, causing the test to be extremely slow.
 label: code-design
82438. Base level is still level 3.
82439. !defined(ROCKSDB_LITE)
82440. Write about 650K more. Each file is about 11KB, with 9KB of data.
82441. Make sure level 0 is not empty
82442. Force not expanding in compactions
82443. Compact against empty DB
82444. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
82445. std::cout << "Hit in " << filePath << "\n";
82446. ROCKSDB_LITE
82447. Open all files and look for the values we've put in there. They should not be found if encrypted, otherwise they should be found.
82448. namespace rocksdb
82449. namespace rocksdb

82450. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

82451. Verify setting an empty high-pri (flush) thread pool causes flushes to be scheduled in the low-pri (compaction) thread pool.

82452. ROCKSDB_LITE

82453. flush failed. ROCKSDB_LITE Flush job should release ref count to current version.

82454. If the issue is hit we will wait here forever.

82455. We had issue when two background threads trying to flush at the same time, only one of them get committed. The test verifies the issue is fixed.

82456. No inplace updates. All updates are puts with new seq number All 10 updates exist in the internal iterator

82457. Update key with values of smaller size

82458. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

82459. Callback function requests no actions from db

82460. Only 1 instance for that key.

82461. Update key with values of smaller varint size

82462. All 10 updates exist in the internal iterator

82463. Update key with values of larger size

82464. namespace rocksdb

82465. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

82466. Check that compaction attempts slept after errors TODO @krad: Figure out why ASSERT_EQ 5 keeps failing in certain compiler versions

82467. Test the following: (a) A random put fails in paranoid mode (simulate by sync fail) (b) All other puts have to fail, even if writes would succeed (c) All of that should happen ONLY if paranoid_checks = true

82468. Check background error counter bumped on flush failures.

82469. Try again with paranoid_checks=false

82470. the next put should NOT fail

82471. Check that CompactRange() returns failure if there is not enough space left on device

82472. Merging compaction (will fail)

82473. but we're still able to read

82474. Memtable compaction (will succeed)

82475. simulate error

82476. !(defined(NDEBUG) || !defined(OS_WIN)) ROCKSDB_LITE namespace rocksdb

82477. Recovery: should not lose data

82478. do the same thing with paranoid checks off

82479. Check that number of files does not grow when writes are dropped

82480. We iterate twice. In the second iteration, everything is the same except the log record never makes it to the MANIFEST file.

82481. Since paranoid_checks=true, writes should fail

82482. Should fail

82483. the next put should fail, too

82484. ROCKSDB_LITE

82485. First 1MB doesn't get range synced

82486. Following writes should fail as compaction failed.

82487. Insert foo=>bar mapping

82488. Test for the following problem: (a) Compaction produces file F (b) Log record containing F is written to MANIFEST file, but Sync() fails (c) GC deletes F (d) After reopening DB, reads fail since deleted F is named in log record

82489. foo=>bar is now in last level

82490. Background error count is 0 now.

82491. generate 5 tables

82492. Force out-of-space errors

82493. disallow trivial move

82494. Following writes should fail as flush failed.

82495. Since paranoid_checks=false, writes should succeed

82496. Test to make sure that the request will *not* fail as incomplete if num_internal_keys_skipped is *equal* to max_skippable_internal_keys threshold. (It will fail as incomplete only when the threshold is exceeded.)

82497. Test call back inserts two keys before "z" in mem table after MergeIterator::Prev() calls mem table iterator's Seek() and before calling Prev()

82498. Test that the num_internal_keys_skipped counter resets after a successful read.

82499. Test to check the SeekToLast() with iterate_upper_bound set to the first key

82500. Test for large number of skippable internal keys with a *non-default* max_sequential_skip_in_iterations.

82501. Test to check the SeekToLast() with the iterate_upper_bound set to the key that is deleted

82502. namespace rocksdb

82503. Test to check the SeekToLast() with iterate_upper_bound set (Deletion cases - Lot of internal keys after the upper_bound is deleted)

82504. Test call back inserts entries for update a key in the end of the mem table after MergeIterator::Prev() realized the mem table iterator is at its end and before an SeekToLast() is called.

82505. num_internal_keys_skipped counter resets here.

82506. **comment:** TODO(3.13): fix the issue of Seek() then Prev() which might not necessary return the biggest element smaller than the seek key.
label: code-design

82507. Test call back inserts entries for update a key in the end of the mem table after MergeIterator::Prev() realized the mem table iterator is at its end and before an SeekToLast() is called.

82508. Test if alternating puts and deletes of the same key are handled correctly.

82509. put, singledelete, merge

82510. **comment:** Test Prev() when one child iterator is at its end but more rows are added.
label: code-design

82511. Test that skipping separate keys is handled

82512. Test case to check SeekToLast with iterate_upper_bound set (same key put may times - SeekToLast should start with the maximum sequence id of the upper bound)

82513. data_[iter_] is not anymore the current element of the iterator. Increment it to reposition it to the right position.

82514. Test for large number of skippable internal keys with *default* max_sequential_skip_in_iterations.

82515. Fail the request as incomplete when num_internal_keys_skipped > max_skippable_internal_keys

82516. Test call back inserts entries for update a key before "z" in mem table after MergeIterator::Prev() calls mem table iterator's Seek() and before calling Prev()

82517. force seek

82518. internal_iter1_: a, f, g internal_iter2_: a, b, c, d, adding (z)

82519. should be called before operations with iterator

82520. Test call back inserts an entry for update a key before "z" in mem table after MergeIterator::Prev() calls mem table iterator's Seek() and before calling Prev()

82521. insert a key smaller than current key

82522. Test Prev() when one child iterator is at its end.

82523. Test call back inserts a key in the end of the mem table after MergeIterator::Prev() realized the mem table iterator is at its end and before an SeekToLast() is called.

82524. Test to check the SeekToLast() with iterate_upper_bound set
82525. Test to check the SeekToLast() with iterate_upper_bound not set
82526. Basic test case ... Make sure explicitly passing the default value works. Skipping internal keys is disabled by default, when the value is 0.
82527. Test Prev() when one child iterator is at its end but more rows are added and max_skipped is triggered.
82528. read data earlier than seqId 8
82529. Test Prev() when one child iterator has more rows inserted between Seek() and Prev() when changing directions.
82530. Test that the num_internal_keys_skipped counter resets after a successful read. Reverse direction
82531. Test to check the SeekToLast() with the iterate_upper_bound set (Deletion cases)
82532. Test to check the SeekToLast() iterate_upper_bound set to a key that is not Put yet
82533. Test to check the SeekToLast() with the iterate_upper_bound set (Checking the value of the key which has sequence ids greater than and less than the iterator's sequence id)
82534. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
82535. max iterators before reseek
82536. Separate values and merge operands in different file so that we make sure that we dont merge them while flushing but actually merge them in the read path
82537. Testing reverse iterator At this point, we have three versions of "a" and one version of "b". The reseek statistics is already at 1.
82538. Check that iterator did something like what we expect.
82539. Restore original position using Prev()
82540. Putting such deletes will force DBIter::Prev() to fallback to a Seek
82541. Insert a lot.
82542. **comment:** Check that the seek didn't do too much work. Checks are not tight, just make sure that everything is well below 100.
 label: code-design
82543. +2 all keys divisible by 2 in range [0 => 999]
82544. doesn't support SeekToLast
82545. Generate Random data
82546. Create a key that needs to be skipped for Seq too new
82547. base case with no bound
82548. **comment:** This test verifies block cache behaviors, which is not used by plain table format. Exclude kHashCuckoo as it does not support iteration currently
 label: code-design
82549. **comment:** TODO(gzh): merge operator does not support backward iteration yet
 label: code-design
82550. ROCKSDB_LITE
82551. Verify that all keys slices are valid (backward)
82552. Create iterator.
82553. 100K values
82554. Generate 4 sst files in L2
82555. Dont give merge operations for some keys
82556. 10 Mb
82557. +3 all keys divisible by 5 in range [0 => 999]
82558. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82559. +1 all keys in range [0 => 999]
82560. 20% of keys will be used to test seek() 20% of keys will be deleted 20% of keys will be added using Merge()
82561. skip as HashCuckooRep does not support snapshot
82562. Restore original position using Seek()
82563. read in the specified block via a regular get
82564. testing that iterate_upper_bound prevents iterating over deleted items if the bound has already reached
82565. namespace
82566. iterate_upper_bound points beyond the last expected entry
82567. Check that memtable wasn't flushed.
82568. Small number of merge operands to make sure that DBIter::Prev() dont fall back to Seek()
82569. The test needs to be changed if kPersistedTier is supported in iterator.
82570. Write to force compactions
82571. Percentage of keys that wont get merge operations
82572. Testing SeekToLast with iterate_upper_bound set
82573. Insert another version of b and assert that reseek is not invoked
82574. insert a total of three keys with same userkey and verify that reseek is still not invoked.
82575. the iteration should stop as soon as the bound key is reached even though the key is deleted hence internal_delete_skipped_count should be 0
82576. Put more values after the snapshot
82577. Key 10 bytes / Value 10 bytes
82578. Seek will bump ITER_BYTIES_READ
82579. Verify that all keys slices are valid
82580. the previous Prev call should have invoked reseek
82581. For half of the key range we will write multiple deletes first to force DBIter::Prev() to fall back to Seek()
82582. insert three keys with same userkey and verify that reseek is not invoked. For each of these test cases, verify that we can find the next key "b".
82583. Pick random keys to be used to test Seek()
82584. scan using non-blocking iterator. We should find it because it is in memtable.
82585. Generate 4 sst files in L0
82586. Test SeekForPrev to random keys
82587. Add some keys/values in memtables
82588. Delete 50% of the keys and update the other 50%
82589. namespace rocksdb
82590. **comment:** TODO: merge operator does not support backward iteration yet
 label: code-design
82591. testing iterate_upper_bound and forward iterator to make sure it stops at bound
82592. Percentage of keys that will be deleted
82593. Test Seek to random keys
82594. should stop here...
82595. First File containing 5 blocks of puts
82596. verify that we can find it via a non-blocking scan
82597. !ROCKSDB_LITE
82598. Use value size that will make sure that every block contain 1 key
82599. Insert a key, create a snapshot iterator, overwrite key lots of times, seek to a smaller key. Expect DBIter to fall back to a seek instead of going through all the overwrites linearly.
82600. Check that we can skip over a run of user keys by using reseek rather than sequential scan
82601. Verify key/value of current position
82602. Switch from forward to reverse

82603. Get iterator that will yield the current contents of the DB.
82604. Delete some random keys
82605. big enough to avoid flush
82606. Second file containing 9 blocks of merge operands
82607. Verify correctness.
82608. write one kv to the database.
82609. Enable prefix bloom for SST files
82610. Switch from reverse to forward
82611. write three entries with different keys using Merge()
82612. testing basic case with no iterate_upper_bound and no prefix_extractor
82613. prefix is the first letter of the key
82614. verify that a non-blocking iterator does not find any kvs. Neither does it do any IOs to storage.
82615. Do some Next() operations the restore the iterator to original position
82616. Insert.
82617. **comment:** TODO(3.13): fix the issue of Seek() + Prev() which might not necessarily return the biggest key which is smaller than the seek key.
 label: code-design
82618. Make sure iter stays at snapshot
82619. every block will contain one entry
82620. Test iterating all data forward
82621. insert two more versions of b. This makes a total of 4 versions of b and 4 versions of a.
82622. now testing with iterate_bound
82623. Seek iterator to a smaller key.
82624. testing with iterate_upper_bound and prefix_extractor Seek target and iterate_upper_bound are not same prefix This should be an error
82625. Insert data to true_data map and to DB
82626. Test iterating all data backward
82627. insert a total of four keys with same userkey and verify that reseek is invoked.
82628. flush memtable to storage. Now, the key should not be in the memtable neither in the block cache.
82629. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82630. Setup sync point dependency to reproduce the race condition of a log file moved to archived dir, in the middle of GetSortedWalFiles
82631. sync point is not included with DNDEBUG build
82632. Corrupt this log to create a gap
82633. this iter would miss "key4" if not fixed
82634. Introduction of SyncPoint effectively disabled building and running this test in Release build. which is a pity, it is a good test
82635. "key5" would be written in a new memtable and log
82636. Insert a new entry to a new log file
82637. trigger async flush, and log move. Well, log move will wait until the GetSortedWalFiles:1 to reproduce the race condition
82638. namespace
82639. Try to read past the gap, should be able to seek to key1025
82640. !defined(ROCKSDB_LITE)
82641. namespace rocksdb
82642. Try to read from the beginning. Should stop before the gap and read less than 1025 entries
82643. Verifies MemTableRepFactory is told the right column family id.
82644. namespace rocksdb
82645. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
82646. **comment:** workaround since there's no port::kMaxUint32 yet.
 label: code-design
82647. 10% value 10% overwrite 10% delete
82648. Skip HashCuckoo since it does not support merge operators
82649. namespace rocksdb
82650. Code executed before merge operation
82651. Overwrite random kNumPutAfter keys
82652. Do kTotalMerges merges
82653. Decrease cache capacity to force all unrefed blocks to be evicted
82654. Filse to write in L0 before compacting to lower level
82655. kNumPutBefore keys will have base values
82656. Keep last batch in memtable and stop
82657. Every key gets ~10 operands
82658. Evict all tables from cache before every merge operation
82659. Will trigger a merge when hitting max_successive_merges and the merge will fail. The delta will be inserted nevertheless.
82660. ROCKSDB_LITE
82661. 3 L0 files 1 L1 file 3 L2 files 1 L3 file 3 L4 Files
82662. Increase capacity again after doing the merge
82663. Code executed after merge operation
82664. every block will contain one entry
82665. wait for the key to be written
82666. Data should stay unmerged after the error.
82667. Delete random kNumDelete keys
82668. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
82669. Test merge operator functionality.
82670. 1000 keys every key have 30 operands, every operand is in a different file
82671. **comment:** After delete_obsolete_files_period_micros updated to 0, the next call to FindObsoleteFiles should make a full scan
 label: code-design
82672. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82673. Verify if DB can be reopen after setting options.
82674. RocksDB lite don't support dynamic options.
82675. test with level0_stop_writes_trigger
82676. ROCKSDB_LITE
82677. Background compaction executed.
82678. Need to insert two keys to avoid trivial move.
82679. default value
82680. examine the table cache (actual size should be 1014)
82681. Wait for stall condition recalculate.

82682. Block background compaction.
82683. GetOptions should be able to get latest option changed by SetOptions.
82684. Verify that candidate files set is empty when no full scan requested.
82685. namespace rocksdb
82686. Phase 1. Simply issue Put() and expect "cur-size-all-mem-tables" equals to "size-all-mem-tables"
82687. small write buffer
82688. compression ratio is -1.0 when no open files at level
82689. Restart the DB. Although number of files didn't reach options.level0_file_num_compaction_trigger, compaction should still be triggered because of the need-compaction hint.
82690. After reopening, no table reader is loaded, so no memory for table readers
82691. "192" is the size of the metadata of two empty skiplists, this would break if we change the default skiplist implementation
82692. Trigger automatic compactions.
82693. Small write buffer
82694. This ensures there no compaction happening when we call GetProperty().
82695. Clear internal stats
82696. Trigger compaction
82697. Block sync calls
82698. In the second round, add iterators.
82699. Come back to write to default column family
82700. Close before mock_env destructs.
82701. Set sizes to both background thread pool to be 1 and block them.
82702. ROCKSDB_LITE namespace rocksdb
82703. Verify the same set of properties through GetIntProperty
82704. anonymous namespace
82705. Reopen and issue iterating. See thee latency tracked
82706. Fill memtable
82707. Put common data ("key") at end to prevent delta encoding from compressing the key effectively
82708. no compression at L0, so ratio is less than one
82709. **comment:** TODO(techdept) : Disabled flaky test #12863555
 label: test
82710. Reopen and issue Get(). See thee latency tracked
82711. Phase 5. Reopen, and expect all these three counters to be the same again.
82712. **comment:** TODO(noetzli) kFlushesPerRound does not really correlate with how many flushes happen.
 label: code-design
82713. Use an iterator to hold current version
82714. Test rocksdb.num-live-versions
82715. **comment:** Data at L1 should be highly compressed thanks to Snappy and redundant data in values (ratio is 12.846 as of 4/19/2016).
 label: code-design
82716. Make sure that there is no flush between getting the two properties.
82717. **comment:** Issue flush and expect larger memory usage of table readers.
 label: code-design
82718. Phase 3. Delete iterators and expect "size-all-mem-tables" shrinks whenever we release an iterator.
82719. Gives larger bias here as index block size, filter block size, and data block size become much harder to estimate in this test.
82720. Expect all these three counters to be the same.
82721. ROCKSDB_LITE
82722. "rocksdb.estimate-oldest-key-time" only available to fifo compaction.
82723. default CF Create 4 tables in default column family
82724. Phase 0. The verify the initial value of all these properties are the same as we have no mem-tables.
82725. After reading a key, at least one table reader is loaded.
82726. CF 1 should show no histogram.
82727. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82728. If C++ gets a std::string_literal, this would be better to check at compile-time using static_assert.
82729. We iterate every key twice. Is it a bug?
82730. Expect the size shrinking
82731. put something and read it back , CF 1 should show histogram.
82732. Force flush to prevent flush from happening between getting the properties or after getting the properties and before the new round.
82733. Phase 2. Keep issuing Put() but also create new iterators. This time we expect "size-all-mem-tables" > "cur-size-all-mem-tables".
82734. Get() after flushes, See latency histogram tracked.
82735. Create 4 tables
82736. Wait for compaction to be done. This is important because otherwise RocksDB might schedule a compaction when reopening the database, failing assertion (A) as a result.
82737. Create 2 files
82738. Release sync calls
82739. Make them never flush
82740. When "max_open_files" is -1, we read all the files for "rocksdb.estimate-num-keys" computation, which is the ground truth. Otherwise, we sample 20 newest files to make an estimation. Formula: lastest_20_files_active_key_ratio * total_files
82741. in no iterator case, these two number should be the same.
82742. (A)
82743. Clear Level 0 so that when later flush a file with deletions, we don't trigger an organic compaction.
82744. options.max_open_files preloads table readers.
82745. Write 100KB (100 values, each 1K)
82746. When i == 1, compaction will output some files to L1, at which point L1 is not bottommost so range deletions cannot be compacted away. The new L1 files must be generated with non-overlapping key ranges even though multiple subcompactions see the same ranges deleted, else an assertion will fail. Only enable auto-compactions when we're ready; otherwise, the oversized L0 (relative to base_level) causes the compaction to run earlier.
82747. Write 1MB (256 values, each 4K)
82748. Regression test for bug where sentinel range deletions (i.e., ones with sequence number of zero) were included in output files. snapshot protects range tombstone from dropping due to becoming obsolete.
82749. input_level
82750. During compaction to bottommost level, verify range tombstones older than the oldest snapshot are removed, while others are preserved.
82751. we need to prevent trivial move using Puts so compaction will actually process the merge operands.
82752. Make sure a given key appears in each file so compaction won't be able to use trivial move, which would happen if the ranges were non-overlapping. Also, we need an extra element since flush is only triggered when the number of keys is one greater than SpecialSkipListFactory's limit. We choose a key outside the key-range used by the test to avoid conflict.
82753. put enough keys to fill up the first subcompaction, and later range-delete them so that the first subcompaction outputs no key-values. In that case it'll consider making an SST file dedicated to range deletions.
82754. background compaction may happen early for kFilesPerLevel'th file

82755. Delete merge operands from all but the last file
82756. Write half of the keys before the tombstone and half after the tombstone. Only covered keys (i.e., within the range and older than the tombstone) should be deleted.
82757. give files overlapping key-ranges to prevent trivial move
82758. This puts table caches in the state of being externally referenced only so they are destroyed immediately upon iterator unreferencing.
82759. regression test for exactly filled compaction output files. Previously another file would be generated containing all range deletions, which could invalidate the non-overlapping file boundary invariant.
82760. Want max_compaction_bytes to trigger the end of compaction output file, not target_file_size_base, so make the latter much bigger
82761. disallow_trivial_move
82762. range tombstone covers first half of the previous file
82763. 0+1+2+...+9
82764. PlainTableFactory and NumTableFilesAtLevel() are not supported in ROCKSDB_LITE
82765. namespace rocksdb
82766. output_level
82767. snapshot prevents key from being deleted during flush
82768. Now L1-L3 are full, when we compact L1->L2 we should see (1) subcompactions happen since input level > 0; (2) range deletions are not dropped since output level is not bottommost. If no file boundary assertion fails, that probably means universal compaction + subcompaction + range deletion are compatible.
82769. the above range tombstone can be dropped, so that one alone won't cause a dedicated file to be opened. We can make one protected by snapshot that must be considered. Make its range outside the first subcompaction's range to exercise the tricky part of the code.
82770. delete [95,105) in two files, [295,305) in next two
82771. end
82772. Regression test for #2752. Range delete tombstones between different snapshot stripes are not stored in order, so the first tombstone of each snapshot stripe should be checked as a smallest candidate.
82773. The RangeDelAggregator holds pointers into range deletion blocks created by table readers. This test ensures the aggregator can still access those blocks even if it outlives the table readers that created them. DBIter always keeps readers open for L0 files. So, in order to test aggregator outliving reader, we need to have deletions in L1 files, which are opened/closed on-demand during the scan. This is accomplished by setting kNumRanges > level0_stop_writes_trigger, which prevents deletions from all lingering in L0 (there is at most one range deletion per L0 file). The first L1 file will contain a range deletion since its begin key is 0. SeekToFirst() references that table's reader and adds its range tombstone to the aggregator. Upon advancing beyond that table's key-range via Next(), the table reader will be unreferenced by the iterator. Since we manually call Evict() on all readers before the full scan, this unreference causes the reader's refcount to drop to zero and thus be destroyed. When it is destroyed, we do not remove its range deletions from the aggregator. So, subsequent calls to Next() must be able to use these deletions to decide whether a key is covered. This will work as long as the aggregator properly references the range deletion block.
82774. 6+7+8+9
82775. make sure compaction treats files containing a split range deletion in the input level as an atomic unit. I.e., compacting any input-level file(s) containing a portion of the range deletion causes all other input-level files containing portions of that same range deletion to be included in the compaction.
82776. Ensures range deletion spanning multiple compaction output files that are cut by max_compaction_bytes will have non-overlapping key-ranges.
<https://github.com/facebook/rocksdb/issues/1778>
82777. 6+7+8+9 (earlier operands covered by tombstone)
82778. snapshot protects range tombstone from dropping due to becoming obsolete.
82779. insert range deletions [95,105) in two files, [295,305) in next two to prepare L1 for later manual compaction.
82780. Must be > 1 so the first L1 file can be closed before scan finishes
82781. exclusive
82782. 1+2+...+9
82783. first iteration verifies query correctness in memtable, second verifies query correctness for a single SST file
82784. background compaction may happen early for kNumFiles'th file
82785. num_entries_flush
82786. ROCKSDB_LITE
82787. each block holds two keys
82788. For L1+, iterators over files are created on-demand, so need seek
82789. !ROCKSDB_UBSAN_RUN
82790. put extra key to trigger final flush
82791. It spans the whole key-range, thus will be included in all output files
82792. Hold a snapshot to separate these two delete ranges.
82793. i == 0: CompactFiles i == 1: CompactRange i == 2: automatic compaction
82794. Write 12K (4 values, each 3K)
82795. iterations check unsupported in memtable, l0, and then l1
82796. prevents empty after compaction
82797. deletes [0, 5]
82798. obsolete after compaction
82799. gaps between ranges creates sentinels in our internal representation
82800. regression test for #2743. the range delete tombstones in memtable should be added even when Get() skips searching due to its prefix bloom filter 1MB 8KB
82801. Write a third before snapshot, a third between snapshot and tombstone, and a third after the tombstone. Keys older than snapshot or newer than the tombstone should be preserved.
82802. need a L1 file for subcompaction to be triggered
82803. put extra key to trigger flush
82804. Hold a snapshot so range deletions can't become obsolete during compaction to bottommost level (i.e., L1).
82805. SpecialSkipListFactory lets us specify maximum number of elements the memtable can hold. It switches the active memtable to immutable (flush is prevented by the above options) upon inserting an element that would overflow the memtable.
82806. Copyright (c) 2016-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
82807. begin
82808. NumTableFilesAtLevel() is not supported in ROCKSDB_LITE
82809. protected by snapshot
82810. output_path_id
82811. Iterates kNumFiles * kNumPerFile + 1 times since flushing the last file requires an extra entry.
82812. Keep clearing block cache's LRU so range deletion block can be freed as soon as its refcount drops to zero.
82813. Put a snapshot before the range tombstone, verify an iterator using that snapshot sees all inserted keys.
82814. extra entry to trigger SpecialSkipListFactory's flush
82815. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82816. If pending output guard does not work correctly, PurgeObsoleteFiles() will delete the file that Compaction is trying to create, causing this: error db/db_test.cc:975: IO error: /tmp/rocksdbtest-1552237650/db_test/000009.sst: No such file or directory
82817. Randomly sleep shortly
82818. This flush will cause bg_error_ and will fail
82819. used in the main loop assert
82820. Live SST files = 0 Total SST files = 0
82821. Compaction will do trivial move from L0 to L1
82822. Close DB and destroy it using DeleteScheduler
82823. Verify the total files size

82824. Hold BackgroundEmptyTrash
82825. Live SST files = 0 Total SST files = 5 (used in 2 version)
82826. clear error to ensure compaction callback is called
82827. Compaction will delete files from L0 in first db path and generate a new file in L1 in second db path
82828. trigger compaction when we have 2 files
82829. If the moved file is actually deleted (the move-safeguard in ~Version::Version() is not there), we get this failure: Corruption: Can't access /000009.sst
82830. namespace rocksdb
82831. 1 Create some SST files by inserting K-V pairs into DB 2 Close DB and change suffix from ".sst" to ".ldb" for every other SST file 3 Open DB and check if all key can be read
82832. **comment:** File just flushed is too big for L0 and L1 so gets moved to L2.
label: code-design
82833. Create a DB with 2 db_paths, and generate multiple files in the 2 db_paths using CompactRangeOptions, make sure that files that were deleted from first db_path were deleted using DeleteScheduler and files in the second path were not.
82834. This file should have been deleted during last compaction
82835. We have deleted the 4 sst files in the delete_scheduler
82836. this should execute both L0->L1 and L1->(move)->L2 compactions
82837. Generate 5 files in L0
82838. This test triggers move compaction and verifies that the file is not deleted when it's part of move compaction
82839. finish the flush!
82840. Live SST files = 5 Total SST files = 5
82841. No files were moved
82842. Open DB with infinite max open files - First iteration use 1 thread to open files - Second iteration use 5 threads to open files
82843. 2 MB 1 MB
82844. Compaction will move the 4 files in L0 to trash and create 1 L1 file
82845. Close db before mock_env destruct.
82846. Set wait until time to before current to force not to sleep.
82847. Generate a file containing 100 keys.
82848. Check the estimated db size vs the db limit just to make sure we dont run into an infinite loop ~60 bytes per key
82849. Verify that we are tracking all sst files in dbname_
82850. Reopening the DB will load all existing files
82851. low limit for size calculated using sst files
82852. 1 MB
82853. Turn timed wait into a simulated sleep
82854. hold current version
82855. Live SST files = 1 (compacted file) Total SST files = 6 (5 original files + compacted file)
82856. Delete all keys and compact, this will delete all live files
82857. block the flush
82858. Compact 5 files into 1 file in L0
82859. Live SST files = 0 Total SST files = 1 (compacted file)
82860. ROCKSDB_LITE
82861. Set the maximum allowed space usage to the current total size
82862. This reproduces a bug where we don't delete a file because when it was supposed to be deleted, it was blocked by pending_outputs Consider: 1. current file_number is 13 2. compaction (1) starts, blocks deletion of all files starting with 13 (pending outputs) 3. file 13 is created by compaction (2) 4. file 13 is consumed by compaction (3) and file 15 was created. Since file 13 has no references, it is put into VersionSet::obsolete_files_ 5. FindObsoleteFiles() gets file 13 from VersionSet::obsolete_files_. File 13 is deleted from obsolete_files_set. 6. PurgeObsoleteFiles() tries to delete file 13, but this file is blocked by pending outputs since compaction (1) is still running. It is not deleted and it is not present in obsolete_files_ anymore. Therefore, we never delete it.
82863. Create 4 files in L0
82864. **comment:** We don't propagate oldest-key-time table property on compaction and just write 0 as default value. This affect the exact table size, since we encode table properties as varint64. Force time to be 0 to work around it. Should remove the workaround after we propagate the property on compaction.
label: code-design
82865. Live SST files = 0 Total SST files = 6 (5 original files + compacted file)
82866. Every time we write to a table file, call FOF/POF with full DB scan. This will make sure our pending_outputs_ protection work correctly
82867. 110KB
82868. Verify that we track all the files again after the DB is closed and opened
82869. Create two 1MB sst files
82870. 512 Kb
82871. Insert 2.5MB data, which should trigger a flush because we exceed write_buffer_size. The flush will be blocked with block_first_time pending_file is protecting all the files created after
82872. Create 12 Files in L0 (then move then to L2)
82873. Create 12 files in L0
82874. 1 Mb / Sec
82875. force
82876. Live SST files = 5 Total SST files = 5 (used in 2 version)
82877. 10 Kbs / Sec
82878. We created 4 sst files in L0
82879. We flushed at least 25 files
82880. This test will set a maximum allowed space for the DB, then it will keep filling the DB until the limit is reached and bg_error_ is set. When bg_error_ is set we will verify that the DB size is greater than the limit.
82881. disallow trivial move
82882. Create 1MB sst file
82883. Compaction must have deleted some files
82884. Check that compressions occur and are counted when compression is turned on
82885. Check that compressions do not occur when turned off
82886. pick arbitrary ticker and histogram. On first iteration they're zero because db is unused. On second iteration they're zero due to Reset().
82887. The Put() makes some of the ticker/histogram stats nonzero until we Reset().
82888. compressible string
82889. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
82890. namespace rocksdb
82891. Query the largest range
82892. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82893. build a decent LSM
82894. fetch key from 1st and 2nd table, which will internally place that table to the table cache.
82895. namespace rocksdb
82896. run the query
82897. Indirect test

82898. Make sure that we've received properties for those and for those files only which fall within requested ranges
82899. Random returns numbers with zero included when we pass empty ranges TestGetPropertiesOfTablesInRange() derefs random memory in the empty ranges[0] so want to be greater than zero and even since the below loop requires that random_keys.size() to be even.
82900. namespace
82901. 2. Put two tables to table cache and
82902. Query the empty range
82903. ROCKSDB_LITE
82904. A helper function that ensures the table properties returned in `GetPropertiesOfAllTablesTest` is correct. This test assumes entries size is different for each of the tables.
82905. Since we deref zero element in the vector it can not be empty otherwise we pass an address to some random memory
82906. Create one table per CF, then verify it was created with the column family name property.
82907. Query the middle range
82908. smallestkey < limit && largestkey >= start
82909. 1. Read table properties directly from file
82910. create a bunch of ranges
82911. Create 4 tables
82912. Ensure that we have at least L0, L1 and L2
82913. 3. Put all tables to table cache
82914. Query a bunch of random ranges
82915. Fixed random seed
82916. flush all those keys to an immutable SST file
82917. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
82918. force a flush to make sure that no records are read from memtable
82919. skip "Otest"
82920. Now seek to the same key. The iterator should remain in the same position.
82921. should still be true after compaction
82922. namespace rocksdb
82923. Write rows with keys 00000, 00002, 00004 etc.
82924. This keeps track of the number of times NeedToSeekImmutable() was true.
82925. Sets iterate_upper_bound and verifies that ForwardIterator doesn't call Seek() on immutable iterators when target key is >= prev_key and all iterators, including the memtable iterator, are over the upper bound.
82926. Not valid since "21" is over the upper bound.
82927. Introduction of SyncPoint effectively disabled building and running this test in Release build. which is a pity, it is a good test
82928. Seek(0102) shouldn't find any records since 0202 has a different prefix
82929. 1st L0 file
82930. 2nd L0 file
82931. write a single record, read it using the iterator, then delete it
82932. !defined(ROCKSDB_LITE)
82933. make sure we can read all new records using the existing iterator
82934. we either see the entry or it's not in cache
82935. Seek to 13. This should not require any immutable seeks.
82936. Seek to 00001. We expect to find 00002.
82937. write many more records
82938. add a record and check that iter can see it
82939. level 1: [20, 25] [35, 40] level 2: [10 - 15] [45 - 50] level 3: [20, 30, 40] Previously there is a bug in tailing_iterator that if there is a gap in lower level, the key will be skipped if it is within the range between the largest key of index n file and the smallest key of index n+1 file if both file fit in that gap. In this example, 25 < key < 35 <https://github.com/facebook/rocksdb/issues/1372>
82940. Add another key to the memtable.
82941. Entry is already in cache, lookup will remove the element from lru
82942. Small write buffer
82943. ignore merge for now (*map_)[key.ToString()] = value.ToString();
82944. 3GB value 8MB key
82945. Test for check of prefix_extractor when hash index is used for block-based table
82946. Produce two L0 files with overlapping ranges.
82947. compact it three times
82948. **comment:** Test max_bytes_for_level_multiplier and max_bytes_for_level_base. Now, reduce both multiplier and level base, After filling enough data that can fit in L1 - L3, we should see L1 size reduces to 128KB from 256KB which was asserted previously. Same for L2.
 label: code-design
82949. Trigger reseek
82950. Start threads
82951. put some data
82952. find the new manifest file. assert that this manifest file is the same one as in the previous snapshot. But its size should be larger because we added an extra key after taking the previous snapshot.
82953. Generating 360KB in Level 2
82954. our clients require that GetLiveFiles returns files with "/" as first character!
82955. Unblock background threads
82956. Do a memtable compaction. Before bug-fix, the compaction would not detect the overlap with level-0 files and would incorrectly place the deletion in a deeper level.
82957. If thread tracking is not enabled, compaction count should be 0.
82958. 2nd round: flush and put a new value in memtable.
82959. Within (0.5, 1.5) of 4MB.
82960. Check if deletion worked.
82961. the second half of the test involves in random failure of file creation.
82962. Release sync calls
82963. Create 1 more file to trigger TTL compaction. The old files are dropped.
82964. Block background threads
82965. skip as HashCuckooRep does not support snapshot
82966. Now there is a file in L1.
82967. Only allow one compactin going through.
82968. table with xxhash checksum
82969. Fill levels 1 and 2 to disable the pushing of new memtables to levels > 0.
82970. only 5 files should survive
82971. Initialize state
82972. skip cuckoo hash as it does not support snapshot.
82973. ROCKSDB_USING_THREAD_STATUS
82974. iter == 0 -- leveled iter == 1 -- leveled, but throw in a flush between two levels compacting iter == 2 -- universal

82975. We need the 2nd write to trigger delay. This is because delay is estimated based on the last write size which is 0 for the first write.
82976. Now there is one L1 file (around 90KB) which exceeds 50KB base by 40KB L2 size is 360KB, so the estimated level fanout 4, estimated pending compaction is around 200KB triggering soft_pending_compaction_bytes_limit
82977. TEST_FlushMemTable() is not supported in ROCKSDB_LITE
82978. Block compaction again
82979. Delete archival files.
82980. Create 3 L0 files, making score of L0 to be 3.
82981. Start with 200MB/s
82982. Check that the writer thread counter is \geq the counter in the value
82983. Compaction range falls after files
82984. Github issue #595 Large write batch with column families
82985. Check that there are no extra characters.
82986. 20K 1.5MB
82987. Stop the threads and wait for them to finish
82988. **comment:** * This test is not reliable enough as it heavily depends on disk behavior. * Disable as it is flaky.
label: test
82989. only the non-default column family has non-matching comparator
82990. Check sizes across recovery by reopening a few times
82991. Compact more often
82992. each column family will have 5 thread, each thread generating 2 memtables. each column family should end up with 10 table files
82993. Now there is one L1 file (around 60KB) which exceeds 50KB base by 10KB Given level multiplier 10, estimated pending compaction is around 100KB doesn't trigger soft_pending_compaction_bytes_limit
82994. copy these files to a new snapshot directory
82995. Record the number of compactations at a time.
82996. Put
82997. Block sync calls
82998. **comment:** ApproximateOffsetOf() is not yet implemented in plain table format.
label: requirement
82999. **comment:** Test level0_stop_writes_trigger. Clean up memtable and L0. Block compaction threads. If continue to write and flush memtables. We should see put stop after 8 memtable flushes since level0_stop_writes_trigger = 8
label: code-design
83000. once the data has been flushed, we are able to get the data when kPersistedTier is used.
83001. now it's done
83002. returns false if the calling-Test should be skipped
83003. (Single) delete hidden by a put
83004. Compaction related options
83005. Compact away the placeholder files we created initially
83006. Test with TTL + Intra-L0 compactions.
83007. Fill one mem table
83008. inf
83009. Required if inheriting from testing::WithParamInterface<>
83010. Each file contains a different key which will be dropped later.
83011. Skip HashCuckooRep as it does not support single delete. FIFO and universal compaction do not apply to the test case. Skip MergePut because merges cannot be combined with single deletions.
83012. Insert 400KB. Some data will be compressed
83013. iter - 0 with 7 levels iter - 1 with 3 levels
83014. **comment:** After base level turn L4->L3, L3 becomes LZ4 and L4 becomes Zlib
label: code-design
83015. Test report_bg_io_stats
83016. Test disable_auto_compactions Compaction thread is unblocked but auto compaction is disabled. Write 4 L0 files and compaction should be triggered. If auto compaction is disabled, then TEST_WaitForCompact will be waiting for nothing. Number of L0 files do not change after the call.
83017. generate one more file in level-0, and should trigger level-0 compaction
83018. **comment:** Short sometimes to encourage collisions
label: code-design
83019. 10KB
83020. Decrease buffer size below current usage
83021. Test compression sanity check
83022. 500KB
83023. Block flush thread and disable compaction thread
83024. # rate limiting with 0.7 x threshold
83025. 1 hour
83026. These writes will be slowed down to 1KB/s
83027. writing 96 x 64KB \Rightarrow 6 * 1024KB (L1 + L2) = (1 + 4) * 1024KB
83028. 1st round: put but not flush
83029. After PinnableSlice destruction element is added back in LRU
83030. The first sync point is to make sure there's one flush job running when we perform VerifyOperationCount().
83031. 2MB 2MB
83032. Test max_write_buffer_number Block compaction thread, which will also block the flushes because max_background_flushes == 0, so flushes are getting executed by the compaction thread
83033. Only the new 10 files remain.
83034. Promote L0 level to L2.
83035. Insert 5 tasks to low priority queue and 5 tasks to high priority queue
83036. Check DB is not in read-only state.
83037. Destroy previous versions if they exist. Using the long way.
83038. Compact just the new range
83039. Slow down twice. One for memtable switch and one for flush finishes.
83040. Skip HashCuckooRep as it does not support single delete. FIFO and universal compaction do not apply to the test case. Skip MergePut because single delete does not get removed when it encounters a merge.
83041. Multiget
83042. The Put Phase.
83043. Skip HashCuckooRep as it does not support single delete. FIFO and universal compaction do not apply to the test case. Skip MergePut because single delete does not get removed when it encounters a merge.
83044. Generate and flush a file about 20KB.
83045. Step 4: Wait for compaction to finish
83046. repeat the test with differet number of high / low priority threads
83047. Multi-element batch
83048. Increase
83049. Create 10 more files. The old 5 files are dropped as their ttl expired.

83050. namespace
83051. Delete databases
83052. Make rocksdb busy
83053. Try up to 60 seconds.
83054. Block compaction
83055. Place a table at level last-1 to prevent merging with preceding mutation
83056. Don't trigger compact/slowdown/stop
83057. 110KB
83058. release file snapshot
83059. Occasionally sleep a while
83060. Github issue #596
83061. For DB instances that use the hash index + block-based table, the iterator will be invalid right when seeking a non-existent key, right than return a key that is close to it.
83062. look at the new live files after we added an 'extra' key and after we took the first snapshot.
83063. Let them run for a while
83064. Generating 360KB in Level 3
83065. 1MB 4MB
83066. the oldest wal should now be getting_flushed
83067. 100KB
83068. Multi-threaded test:
83069. With Intra-L0 compaction, out of 10 files, 6 files will be compacted to 1 (due to level0_file_num_compaction_trigger = 6). So total files = 1 + remaining 4 = 5.
83070. only 2 memtables will be alive, so logs_to_free needs to always be below 2
83071. **comment:** TODO(yhchiang): adding assert to verify each compaction stage.
label: test
83072. Reopen it without prefix extractor, make sure everything still works. RocksDB should just fall back to the binary index.
83073. first iteration -- auto compaction second iteration -- manual compaction
83074. Test that shows the fall back to size-based FIFO compaction if TTL-based deletion doesn't move the total size to be less than max_table_files_size.
83075. nonoverlapping with the file on level 0
83076. Write approximately 100MB of "B" values
83077. Group commit test:
83078. Test max_bytes_for_level_base Increase level base size to 256KB and write enough data that will fill L1 and L2. L1 size should be around 256KB while L2 size should be around 256KB x 4.
83079. they are either both ok or both not-found
83080. flush should happen here
83081. Put until writes are stopped, bounded by 256 puts. We should see stop at ~128KB
83082. DEL eliminated, but v1 remains because we aren't compacting that level (DEL can be eliminated because v2 hides v1).
83083. Test to check whether flushing preserves a single delete hidden behind a put.
83084. do not compress L0
83085. Half of the time directly use WriteBatch. Half of the time use WriteBatchWithIndex.
83086. This second sync point is to ensure the flush job will not be completed until we already perform VerifyOperationCount().
83087. Test with large TTL + Intra-L0 compactions. Files dropped based on size, as ttl doesn't kick in.
83088. Wait 100 milliseconds for they are scheduled.
83089. writing 20 x 64KB = 10 x 128KB (L1 + L2 + L3) = (1 + 2 + 4) * 128KB
83090. **comment:** make sure the thread is not done
label: requirement
83091. Read a value and verify that it matches the pattern written above and that writes to all column families were atomic (unique_id is the same)
83092. Change the number of threads in high / low priority pool.
83093. We must have at most one file per level except for level-0, which may have up to kL0_StopWritesTrigger files.
83094. About 200KB/s limited rate
83095. Since no flushes and compactions have run, the db should still be in the same state even after considerable time has passed.
83096. Do we own map_
83097. Inserting a new entry would create a new mem table, triggering slow down.
83098. Write values of the form <key, my id, counter, cf, unique_id>. into each of the CFs We add some padding for force compactions.
83099. **comment:** ApproximateOffsetOf() is not yet implemented in plain table format, which is used by Size(). skip HashCuckooRep as it does not support snapshot
label: requirement
83100. Compact all files.
83101. Write 120KB (12 values, each 10K)
83102. Save a snapshot from each DB this time that we'll use next time we compare things, to make sure the current state is preserved with the snapshot
83103. here we expect some of the Put fails.
83104. Check that we process level-0 files in correct order. The code below generates two level-0 files where the earlier one comes before the later one in the level-0 file list since the earlier one has a smaller "smallest" key.
83105. CURRENT, MANIFEST, OPTIONS, *.sst files (one for each CF)
83106. Try if CreateColumnFamily also fails
83107. 1MB
83108. Does not exist, and create_if_missing == true: OK
83109. **comment:** The existing memtable became eligible for flush when we reduced its capacity to 64KB. Two keys need to be added to trigger flush: first causes memtable to be marked full, second schedules the flush. Then we should have a 128KB L0 file, a 64KB L0 file, and a memtable with just one key.
label: code-design
83110. This makes sure at least one compaction is running.
83111. Fill memtable Trigger flush
83112. Not Supported directly
83113. Start from scratch and disable compaction/flush. Flush can only happen during compaction but trigger is pretty high
83114. Advance sequence number one more
83115. Compact all
83116. very small only two memtables allowed ==> only two log files
83117. Verify the total number of threads
83118. get a file snapshot
83119. **comment:** Cleaning up
label: code-design
83120. **comment:** Insert more than 80K. L4 should be base level. Neither L0 nor L4 should be compressed, so total data size should be more than 80K.
label: code-design
83121. This test may fail because of a legit case that multiple L0 files are trivial moved to L1.
83122. repeat the test with multiple column families
83123. No files in L1
83124. Create 3 L0 files, making score of L0 to be 3
83125. new wal should have been created
83126. (A)

83127. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

83128. all flushes should now do nothing because their CF is dropped

83129. Estimate the total sleep time fall into the rough range.

83130. Single deletes that encounter the matching put in a flush should get removed.

83131. 10 KB

83132. Check that it is not supported with max_open_files != -1.

83133. When base level is L4, L4 is LZ4.

83134. Enable auto compaction and perform the same test, # of L0 files should be reduced after compaction.

83135. Make files spanning the following ranges in level-0: files[0] 200 .. 900 files[1] 300 .. 500 Note that files are sorted by smallest key.

83136. make sure we can re-open it.

83137. Schedule a sleeping task.

83138. No reseek

83139. Decrease

83140. **comment:** Speed up the test
label: test

83141. This makes sure a compaction won't be scheduled until we have done with the above Put Phase.

83142. Wait to ensure the all threads has been registered

83143. Setup databases

83144. this has to start a flush. if flushes are blocked, this will try to create 3 memtables, and that will fail because max_write_buffer_number is 2

83145. The existing memtable inflated 64KB->128KB when we invoked SetOptions(). Write 192KB, we should have a 128KB L0 file and a memtable with 64KB data.

83146. namespace rocksdb

83147. Make sparse update

83148. Still expect finding the value as its delete has not yet being flushed.

83149. Check format

83150. It should be compacted to no more than 20 files.

83151. **comment:** The following condition prevents a race condition between flush jobs acquiring work and this thread filling up multiple memtables. Without this, the flush might produce less files than expected because multiple memtables are flushed into a single L0 file. This race condition affects assertion (A).
label: code-design

83152. Stop trigger = 8

83153. But everything before we simulate the failure-test should succeed.

83154. If rocksdb does not do the correct job, internal assert will fail here.

83155. no need to call this

83156. Size limit is still guaranteed.

83157. verify we have the latest successful update

83158. this checks that updates across column families happened atomically -- all unique ids are the same

83159. should not compact the level 0 file

83160. **comment:** Clean up L0
label: code-design

83161. Does exist, and error_if_exists == false: OK

83162. Compaction range falls before files

83163. Populate a different range

83164. this should fill up 2 memtables

83165. Test compaction trigger and target_file_size_base Reduce compaction trigger to 2, and reduce L1 file size to 32KB. Writing to 64KB L0 files should trigger a compaction. Since these 2 L0 files have the same key range, compaction merge them and should result in 2 32KB L1 files.

83166. Read directly from persisted data.

83167. pool size 2, total task 4. Queue size should be 2.

83168. Delay both flush and compaction

83169. If we compact: Put(A, v1) Snapshot SingleDelete(A) Put(A, v2) We do not want to end up with: Put(A, v1) Snapshot Put(A, v2) Because a subsequent SingleDelete(A) would delete the Put(A, v2) but not Put(A, v1), so Get(A) would return v1.

83170. record the number and the size of the latest manifest file

83171. Delete

83172. Zero if not including mem table

83173. # rate limiting with half of the raw_rate

83174. Make sure that Flushes can proceed in parallel with CompactRange()

83175. Step 3: read a bunch of times

83176. update the latest successful put

83177. reopen and reverify we have the latest successful update

83178. Periodically re-use the same key from the previous iter, so we have multiple entries in the write batch for the same key

83179. **comment:** Use a small number to ensure a large delay that is still effective when we do Put TODO(myabandeh): this is time dependent and could potentially make the test flaky
label: test

83180. Arrange for the following to happen: * sstable A in level 0 * nothing in level 1 * sstable B in level 2 Then do enough Get() calls to arrange for an automatic compaction of sstable A. A bug would cause the compaction to be marked as occurring at level 1 (instead of the correct level 0).

83181. these keys should be deleted in previous compaction

83182. now it should compact the level 0 file

83183. Just to make sure that we are in the same state even after sleeping.

83184. Check that FIFO-with-TTL is supported only with BlockBasedTableFactory.

83185. **comment:** Sleep for 2 hours -- which is much greater than TTL. Note: Couldn't use SleepForMicroseconds because it takes an int instead of uint64_t. Hence used addon_time_directly. env_->SleepForMicroseconds(2 * 60 * 60 * 1000 * 1000);
label: code-design

83186. Now there is one L1 file but doesn't trigger soft_rate_limit The L1 file size is around 30KB.

83187. 0 because GetApproximateSizes() does not account for memtable space

83188. Verify the number of low-priority threads

83189. Unblock

83190. Clear memtable and make new option effective

83191. Suppose there is: small amount of data with prefix A large amount of data with prefix B small amount of data with prefix C and that recent updates have made small changes to all three prefixes. Check that we do not do a compaction that merges all of B in one shot.

83192. Does exist, and error_if_exists == true: error

83193. create a DB with block prefix index

83194. Most intervals should've been drained (interval time is 100ms, elapsed is micros)

83195. Fails because L1 is non-empty.

83196. do not compress L0 and L1

83197. Fails because L0 has overlapping files.

83198. Test paranoid_file_checks already done in db_block_cache_test

83199. Test max_sequential_skip_in_iterations

83200. overwrite one key, this key should not appear in the snapshot

83201. First three 110KB files are not going to level 2 After that, (100K, 200K)
83202. Enable time profiling
83203. Write 3 files that have the same key range. Since level0_file_num_compaction_trigger is 3, compaction should be triggered. The compaction should result in one L1 file
83204. as data has not yet being flushed, we expect not found.
83205. Test to make sure that all files with expired ttl are deleted on next manual compaction.
83206. ROCKSDB_LITE
83207. Disable because not all platform can run it. It requires more than 9GB memory to run it, With single allocation of more than 3GB.
83208. sanity check
83209. expecting one single L0 to L1 compaction
83210. Create 3 L0 files, making score of L0 to be 3, higher than L0.
83211. **comment:** Step 2: clear level 1 if necessary.
 label: code-design
83212. Wake up sleep task to enable compaction to run and waits for it to go to sleep state again to make sure one compaction goes through.
83213. DEL kept: "last" file overlaps
83214. **comment:** TODO(sanjay): Test Get() works
 label: test
83215. Put values on second level (so that they will not be in the same compaction as the other operations.
83216. 150KB
83217. Make sure data in files in L3 is not compacted by removing all files in L4 and calculate number of rows
83218. Not very large
83219. Sleep for 2 hours -- which is much greater than TTL.
83220. Check that FIFO-with-TTL is not supported with max_open_files != -1.
83221. Flush the file. File size is around 30KB.
83222. shrink level base so L2 will hit soft limit easier.
83223. Assuming each files' metadata is at least 50 bytes/
83224. Now reduce level0_stop_writes_trigger to 6. Clear up memtables and L0. Block compaction thread again. Perform the put and memtable flushes until we see the stop after 6 memtable flushes.
83225. foo => v1 is now in last level
83226. Generate and flush a file about 10KB.
83227. compressible string
83228. Expect same result in multiget
83229. Test write_buffer_size
83230. We expect that all the files were trivially moved from L0 to L2
83231. Arrange to have multiple files in a non-level-0 level.
83232. iter 0 -- zlib iter 1 -- bzip2 iter 2 -- lz4 iter 3 -- lz4HC iter 4 -- xpress
83233. pool size 3, total task 4. Queue size should be 1.
83234. Step 1: First place sstables in levels 0 and 2
83235. Moves to level last-2
83236. Test to make sure that all files with expired ttl are deleted on next automatic compaction.
83237. NewMemEnv returns nullptr in ROCKSDB_LITE since class InMemoryEnv isn't defined.
83238. Key has not yet been written
83239. Does not exist, and create_if_missing == false: error
83240. assert that nothing makes it to disk yet.
83241. Allow some expansion from metadata
83242. Block flushes
83243. nowait
83244. Test soft_pending_compaction_bytes_limit, hard_pending_compaction_bytes_limit
83245. Introduction of SyncPoint effectively disabled building and running this test in Release build. which is a pity, it is a good test
83246. **comment:** Clean up memtable and L0
 label: code-design
83247. # no rate limiting
83248. non overlapping ranges
83249. Verify non of the column family info exists
83250. Block compactions
83251. Wait for compaction so that put won't stop
83252. Write ~96M data
83253. Increase buffer size
83254. Large write buffer
83255. 3rd round: delete and flush
83256. Verify the number of high-priority threads
83257. Check format Check format
83258. Merging last-1 w/ last, so we are the base level for "foo", so DEL is removed. (as is v1).
83259. seconds
83260. Spread the size range to more.
83261. not supported, we should fail the Open()
83262. 10KB 150KB
83263. repeat the test with disabling thread tracking.
83264. record the lognumber and the size of the latest manifest file
83265. XXX
83266. verify that data in the snapshot are correct
83267. all statuses have to be the same
83268. first_table_version 1 -- generate with table_version == 1, read with table_version == 2 first_table_version 2 -- generate with table_version == 2, read with table_version == 1
83269. For mode (1), test DestroyDB() to delete all the logs under DB dir. For mode (2), no info log file should have been put under DB dir.
83270. check how many threads are doing compaction using GetThreadList
83271. All into one file
83272. copy only valid MANIFEST data
83273. Windows fails this test. Will tune in the future and figure out approp number
83274. Write 8MB (80 values, each 100K)
83275. 20K
83276. It should be compacted to 10 files.
83277. table with crc checksum
83278. minimum write buffer size is enforced at 64KB
83279. **comment:** Compactions should not cause us to create a situation where a file overlaps too much data at the next level.
 label: code-design
83280. All files are compacted
83281. Compaction range overlaps files

83282. Small write buffer
83283. r124+
83284. arg
83285. The first dependency enforces Marker can be loaded before MarkedPoint. The second checks that thread 1's MarkedPoint should be disabled here. Execution order:
| Thread 1 | Thread 2 | || Marker || MarkedPoint || | Thread1First | ||| MarkedPoint |
83286. only index/filter were added
83287. column_family_name -> column_family_id map (provided to WALFilter)
83288. We expect the record with apply_option_for_record_index to be not found.
83289. Make sure that we wrote enough to check all 7 levels
83290. Processing option that is requested to be applied at the given index
83291. We want to make sure to call this callback only once
83292. default column-family, only post_flush keys are expected
83293. **comment:** GetUniqueIdFromFile is not implemented
label: requirement
83294. On Recovery we should only find the second batch applicable to default CF But both batches applicable to pikachu CF
83295. Create a test filter that would apply wal_processing_option at the first record
83296. Read keys/values randomly and verify that reported read amp error is less than 2%
83297. Get base cache values
83298. ROCKSDB_LITE
83299. Use Snappy except for bottommost level use ZLib
83300. Triggering flush in DB2.
83301. Infinite for full compaction.
83302. While the compaction is running, we will create 2 new files that can fit in L2, these 2 files will be moved to L2 and overlap with the running compaction and break the LSM consistency.
83303. Fill memtable Trigger flush
83304. Start a non-exclusive manual compaction in a bg thread
83305. Miss and hit count should remain the same, they're all pinned.
83306. Ensure that given keys don't exist
83307. Filter is passed as a const object for RocksDB to not modify the object, however we modify it for our own purpose here and hence cast the constness away.
83308. Ensure that expected keys exists and not expected keys don't exist after recovery
83309. Reopen database. If max_open_files is set as -1, table readers will be preloaded. This will trigger a BlockBasedTable::Open() and prefetch L0 index and filter.
Level 1's prefetching is disabled in DB::Open()
83310. Create 3 batches with two keys each
83311. If max_open_files is -1, we have pinned index and filter in Rep, so there will not be changes in index and filter misses or hits. If max_open_files is not -1, Get()
will open a TableReader and prefetch index and filter.
83312. Callback is only executed once
83313. pikachu column-family, all keys are expected
83314. Hold NotifyOnCompactionCompleted in the unlock mutex section
83315. Write 10 random files
83316. After reopen, cache miss are increased by one because we read (and only read) filter and index on L0
83317. /dev/shm dont support getting a unique file id, this mean that running this test on /dev/shm will fail because lru_cache will load the blocks again regardless of them
being already in the cache
83318. Do some more writes
83319. bytes read for user iterator shouldn't count against the rate limit.
83320. Reopen without filter, now reopen should succeed - previous attempt to open must not have altered the db.
83321. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root
directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of
this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83322. Compaction pressure should trigger since 6 files
83323. Wait for compaction to be triggered
83324. 10KiB
83325. Current record index, incremented with each record encountered.
83326. Figure out last level with files
83327. compaction input has 512KB data
83328. This ensures that there is no ref to block cache entries
83329. this is never hit Use a write buffer total size so that the soft limit is about 105000.
83330. Clear "DBImpl::BGWorkCompaction" SYNC_POINT since we want to hold it again at DBTest2::CompactionStall:::1
83331. column_family_name -> keys_found_in_wal map We store keys that are applicable to the column_family during recovery (i.e. aren't already flushed to SST
file(s)) for verification against the keys we expect.
83332. this should be read from L0
83333. This manual compaction conflict with the other manual compaction so it should wait until the first compaction finish
83334. Add a key to memtable
83335. **comment:** Error between reported read amp and real read amp should be less than 2%
label: code-design
83336. namespace
83337. Iter2: Read half the DB, Read odd keys Key(1), Key(3), Key(5), Key(7), Key(9), ...
83338. Index at which to start changing records
83339. get base cache values
83340. Memtable-only iterator (read_tier=kMemtableTier); data not flushed yet.
83341. Block compaction
83342. Ensure that expected keys exist
83343. verify that handles_[0] only has post_flush keys while handles_[1] has pre and post flush keys
83344. When compact from Ln -> Ln+1, cut a file if the file overlaps with more than three files in Ln+1.
83345. this should be read from L1 the file is opened, prefetching results in a cache filter miss the block is loaded and added to the cache, then the get results in a cache hit
for L1 When we have infinite max_files, there is still cache miss because we have reset the block cache
83346. Reopen database with option to use WAL filter
83347. Index at which to apply wal_processing_option_ At other indexes default wal_processing_option::kContinueProcessing is returned.
83348. Make sure to use any custom env that the test is configured with.
83349. Make sure we read every thing in the DB (which is smaller than our cache)
83350. Use Snappy everywhere
83351. takes roughly one second, split into 100 x 10ms intervals. Each interval permits 5.12KB, which is smaller than the block size, so this test exercises the code for
chunking reads.
83352. 4 Files in L0
83353. Trivial move 2 files to L1
83354. Without hitting the threshold, no flush should trigger.
83355. **comment:** Huge block cache to make it easier to calculate read amplification
label: code-design
83356. column_family_id -> log_number map (provided to WALFilter)

83357. Batch to insert keys in
83358. Run a manual compaction that will compact the 2 files in L2 into 1 file in L2
83359. small write buffer
83360. Before flushing point lookups
83361. unhandled case
83362. Trivial move 2 files to L2
83363. While the compaction is running, we will create 2 new files that can fit in L1, these 2 files will be moved to L1 and overlap with the running compaction and break the LSM consistency.
83364. flush all data from memtable so that reads are from block cache
83365. Test with all WAL processing options
83366. Include the explicit prefetch of the footer in direct I/O case.
83367. namespace rocksdb
83368. move this table to L1
83369. Number of keys to add in the new batch
83370. Compute which keys we expect to be found and which we expect not to be found after recovery.
83371. Create a test filter that would add extra keys
83372. high compression ratio
83373. If max_open_files is not -1, we do not preload table readers, so there is no change.
83374. Write given keys in given batches
83375. After flushing point lookups
83376. In case of corruption we can turn off paranoid_checks to reopen database
83377. Test that the stats GetMapProperty API reporting 1 file in L2
83378. !ROCKSDB_LITE
83379. we expect all records to be processed
83380. fairness
83381. no block cache
83382. Trigger another flush. This time "dobbynia". "pikachu" should not be flushed, although it was never flushed.
83383. should be slightly above 512KB due to non-data blocks read. Arbitrarily chose 1MB as the upper bound on the total bytes read.
83384. default column family doesn't have block cache
83385. In this case, cache miss will be increased by one in BlockBasedTable::Open() because this is not in DB::Open() code path so we will prefetch L1's index and filter.
Cache hit will also be increased by one because Get() will read index and filter from the block cache prefetched in previous Open() call.
83386. rate_bytes_per_sec
83387. Create some data and flush "default" and "nikitich" so that they are newer CFs created.
83388. LZ4_VERSION_NUMBER >= 10400
83389. this is the real limit
83390. Insert to DB2
83391. We expect records beyond apply_option_for_record_index to be not found.
83392. Destroy old statistics obj that the blocks in lru_cache are pointing to
83393. compressed
83394. Output files to L1 are cut to three pieces, according to options.max_compaction_bytes
83395. create new table at L0
83396. make sure all background compaction jobs can be scheduled
83397. reset block cache
83398. 1M
83399. IOS_SOLARIS
83400. Trigger a flush on cf2
83401. Create a new table.
83402. In infinite max files case, there's a cache miss in executing Get() because index and filter are not prefetched before.
83403. The total soft write buffer size is about 105000
83404. Avoid undeterministic value by malloc_usable_size(); Force arena block size to 1
83405. uncompressed
83406. Iter1: Read half the DB, Read even keys Key(0), Key(2), Key(4), Key(6), Key(8), ...
83407. Use different compression algorithms for different levels but always use Zlib for bottommost level
83408. Run a manual compaction that will compact the 2 files in L1 into 1 file in L2
83409. Read amp is on average 100% since we read all what we loaded in memory
83410. refill_period_us
83411. Hit the write buffer limit again. "default" will have been flushed.
83412. Add three more small files that overlap with the previous file
83413. No flush should trigger
83414. Flush default column-family
83415. Ensure that all keys exist before change_records_from_index_. And after that index only single key exists as our filter adds only single key for each batch
83416. this should be read from L0 so cache values don't change
83417. Triggering to flush another CF in DB1
83418. Use the statistics object that we just created
83419. Make sure DB can be reopen with reduced number of levels, given no file is on levels higher than the new num_levels.
83420. empty vector
83421. index/filter blocks added to block cache right after table creation.
83422. Also test that the stats GetMapProperty API reporting the same result
83423. **comment:** Disable delta encoding to make it easier to calculate read amplification
label: code-design
83424. reopen database again to make sure previous log(s) are not used (even if they were skipped) reopn database with option to use WAL filter
83425. This snapshot will have sequence number 0 what is expected behaviour.
83426. Number of keys added to new batch
83427. Write 8MB (80 values, each 100K)
83428. this should be read from L1
83429. this is never hit
83430. Get() is issued after the first Put(), so it should see either "v1" or "v2".
83431. If the current record is applicable for column_family_id (i.e. isn't flushed to SST file(s) for column_family_id) add it to the cf_wal_keys_map for verification.
83432. This ensures that db does not ref anything in the block cache, so EraseUnRefEntries could clear them up.
83433. nothing should be returned using memtable-only iterator after flushing.
83434. Another 6 L0 files to trigger compaction again
83435. Wait for another compaction to be triggered
83436. Trigger a flush. Flushing "nikitich".
83437. First iteration: compress without preset dictionary Second iteration: compress with preset dictionary To make sure the compression dictionary was actually used, we verify the compressed size is smaller in the second iteration. Also in the second iteration, verify the data we get out is the same data we put in.
83438. Return the current option configuration.
83439. Destroy using last options
83440. This options optimize 2PC commit path

83441. Switch between different WAL settings
83442. Return a string that contains all key,value pairs in order, formatted like "(k1->v1)(k2->v2)".
83443. Prevent pushing of new sstables into deeper levels by adding tables that cover a specified range to all levels.
83444. Special Env used to delay background operations
83445. **comment:** Verify Iterator::Prev() Use a new iterator to make sure its status is clean.
 label: code-design
83446. Used to test InplaceUpdate
83447. namespace rocksdb
83448. Do n memtable compactions, each of which produces an sstable covering the range [small,large].
83449. Verify Iterator::Next()
83450. These options are not supported in ROCKSDB_LITE
83451. Verify ForwardIterator::Next()
83452. **comment:** Note: operator= is an unsafe approach here since it destructs shared_ptr in the same order of their creation, in contrast to destructors which destructs them in the opposite order of creation. One particular problem is that the cache destructor might invoke callback functions that use Option members such as statistics. To work around this problem, we manually call destructor of table_facotry which eventually clears the block cache.
 label: code-design
83453. Destroy it for not alternative WAL dir is used.
83454. If previous value is nullptr or delta is > than previous value, sets newValue with delta If previous value is not empty, updates previous value with 'b' string of previous value size - 1.
83455. Verify Iterator::Seek()
83456. default argument means copy everything
83457. Tailing iterator
83458. ROCKSDB_LITE
83459. **comment:** TODO(3.13) -- test more options
 label: test
83460. mmap reads should be orthogonal to WalDir setting, so we piggyback to this option config to test mmap reads as well
83461. 50 bytes
83462. **comment:** this redundant copy is to minimize code change w/o having lint error.
 label: code-design
83463. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83464. Verify ForwardIterator::Seek()
83465. default cfd
83466. this will generate non-overlapping files since it keeps increasing key_idx
83467. Switch between different compaction styles.
83468. Check reverse iteration results are the reverse of forward results
83469. Switch between different filter policy Jump from kDefault to kFilter to kFullFilter
83470. Return spread of files per level
83471. keep it the same as kTypeValue for testing kMergePut
83472. Normal Iterator
83473. snapshots
83474. Switch to a fresh database with the next option configuration to test. Return false if there are no more configurations to test.
83475. checks sequence number for updates
83476. Utility method to test InplaceUpdate
83477. Return the current option configuration.
83478. Sequence of option configurations to try
83479. Number of WAL files that are still open for write.
83480. **comment:** A hacky skip list mem table that triggers flush after number of entries.
 label: code-design
83481. **comment:** The factory for the hacky skip list mem table that triggers flush after number of entries exceeds a threshold.
 label: code-design
83482. Drop writes on the floor
83483. Return a string that contains all key,value pairs in order, formatted like "(k1->v1)(k2->v2)".
83484. Prevent pushing of new sstables into deeper levels by adding tables that cover a specified range to all levels.
83485. Check preallocation size preallocation size is never passed to base file.
83486. Special Env used to delay background operations
83487. namespace rocksdb
83488. Used to test InplaceUpdate
83489. Utility method to test InplaceUpdate
83490. Returns true iff an entry that compares equal to key is in the list.
83491. Skip some options, as they may not be applicable to a specific test. To add more skip constants, use values 4, 8, 16, etc.
83492. Simulate no-space errors while this pointer is non-nullptr.
83493. Do n memtable compactions, each of which produces an sstable covering the range [small,large].
83494. Slow down every log write, in micro-seconds.
83495. Force write to log files to fail while this pointer is non-nullptr
83496. Return a high memory usage when number of entries exceeds the threshold to trigger a flush.
83497. namespace anon
83498. These will be used only if filter_policy is set
83499. **comment:** Used as a bit mask of individual enums in which to skip an XF test point
 label: test
83500. If previous value is nullptr or delta is > than previous value, sets newValue with delta If previous value is not empty, updates previous value with 'b' string of previous value size - 1.
83501. A test merge operator mimics put but also fails if one of merge operands is "corrupted".
83502. ROCKSDB_LITE
83503. !(defined(NDEBUG) || !defined(OS_WIN))
83504. Lock to protect rnd_
83505. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83506. this will generate non-overlapping files since it keeps increasing key_idx
83507. Force write to manifest files to fail while this pointer is non-nullptr
83508. Switch between different compaction styles.
83509. Force sync of manifest files to fail while this pointer is non-nullptr
83510. Switch between different filter policy Jump from kDefault to kFilter to kFullFilter
83511. Return spread of files per level
83512. sstable Sync() calls are blocked while this pointer is non-nullptr.
83513. Drop writes on the floor while this pointer is non-nullptr.

83514. 10s

83515. 1s

83516. After number of inserts exceeds `num_entries_flush` in a mem table, trigger flush.

83517. Switch to a fresh database with the next option configuration to test. Return false if there are no more configurations to test.

83518. Simulate non-writable file system while this pointer is non-nullptr

83519. SyncPoint is not supported in Released Windows Mode.

83520. Insert key into the list. REQUIRES: nothing that compares equal to key is currently in the list.

83521. Switch between different WAL-related options.

83522. Write 3 files while 1st compaction is held These 3 files have different sizes to avoid compacting based on size_ratio

83523. Tests universal compaction with trivial move enabled

83524. Hold the 1st compaction from finishing

83525. All files at level 0 will be compacted into a single one.

83526. Make sure bloom filter is used at least once.

83527. Compact all to level 0

83528. (1, 4, 8) -> (5, 8)

83529. Write 120KB (12 values, each 10K)

83530. expect ok and verify the compacted files no longer exist.

83531. Compact all files into 1 file and put it in L4

83532. 1MB

83533. Suppose each file flushed from mem table has size 1. Now we compact (level0_file_num_compaction_trigger+1)=4 files and should have a big file of size 4.

83534. block compaction from happening

83535. Write 8 files while 1st compaction is held These 8 files have different sizes to avoid compacting based on size_ratio

83536. Pick the first and the last file, expect everything is compacted into one single file.

83537. Before compaction, we have 4 files at level 0, with size 4, 2.4, 1, 1. After compaction, we should have 3 files, with size 4, 2.4, 2.

83538. Trigger compaction if size amplification exceeds 110%

83539. Stage 1: Generate a set of files at level 0, but don't trigger level-0 compaction.

83540. When we start for the compaction up to (2 4 8), the latest compressed is not compressed.

83541. Disable size amplification compaction

83542. Stop SyncPoint and destroy the DB and reopen it again

83543. Make sure bloom filter is used for all but the last L0 file when looking up a non-existent key that's in the range of all L0 files.

83544. First three 110KB files are not going to second path. After that, (100K, 200K)

83545. Wait for the 1st background compaction process to start

83546. Insert more keys

83547. Another 110KB triggers a compaction to 400K file to second path

83548. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

83549. **comment:** Stage 2: Now we have one file at level 0, with size 4. We also have some data in mem table. Let's continue generating new files at level 0, but don't trigger level-0 compaction. First, clean up memtable before inserting new data. This will generate a level-0 file, with size around 0.4 (according to previously written data amount).

label: code-design

83550. Write 100KB file. And immediately it should be compacted to one file.

83551. no_wait

83552. 1KB

83553. (1, 2, 8) -> (3, 8)

83554. each value is 10K

83555. When we start for the compaction up to (2 4 8), the latest compressed is compressed given the size ratio to compress.

83556. namespace

83557. This time we will trigger a compaction because of size ratio and another compaction because of number of files that are not compacted greater than 7

83558. !defined(ROCKSDB_LITE)

83559. Hold the 1st and 2nd compaction from finishing

83560. Make sure we don't trigger a problem if the trigger condition is given to be 0, which is invalid.

83561. Wait for the 2nd background compaction process to start

83562. **comment:** Regression test for extra compactions scheduled. Once enough compactions have been scheduled to bring the score below one, we should stop scheduling more; otherwise, other CFs/DBs may be delayed unnecessarily.

label: code-design

83563. Before compaction, we have 4 files at level 0, with size 4, 0.4, 1, 1. After compaction, we should have 2 files, with size 4, 2.4.

83564. Config universal compaction to always compact to one single sorted run.

83565. Generate 3 overlapping files

83566. wait for the full compaction to be picked before adding files intended for the second one.

83567. Write 110KB (11 values, each 10K)

83568. Delete archival files.

83569. Generate two files in Level 0. Both files are approx the same size.

83570. (1, 1, 8) -> (2, 8)

83571. 105KB

83572. Reopen and check.

83573. namespace rocksdb

83574. Test that checks trivial move in universal compaction

83575. Compaction to non-L0 has happened.

83576. 100KB

83577. There should only be one picked compaction as the score drops below one after the first one is picked.

83578. Big size

83579. Fail when compacting to an invalid path ID

83580. Write 100KB

83581. the full (bottom-pri) compaction waits until a partial (low-pri) compaction has started to verify they can run in parallel.

83582. expect fail since universal compaction only allow L0 output

83583. The same queries will not trigger bloom filter

83584. Stage 5: Now we have 4 files at level 0, with size 4, 2.4, 2, 1. Let's generate a new file of size 1.

83585. (1, 2, 4) -> (3, 4)

83586. Stage 3: Now we have 2 files at level 0, with size 4 and 2.4. Continue generating new files at level 0.

83587. Need to restart it once to remove higher level records in manifest.

83588. use no_wait above because that one waits for flush and compaction. We don't want to wait for compaction because the full compaction is intentionally blocked while more files are flushed.

83589. Compacting the first four files was enough to bring the score below one so there's no need to schedule any more compactions.

83590. 100KB 32KB

83591. Query set of non existing keys

83592. Final reopen

83593. The third compaction (2 4) is compressed since this time it is (1 1 3.2) and 3.2/5.2 doesn't reach ratio.

83594. Level-0 compaction is triggered, but no file will be picked up.
83595. The second compaction (4) is compressed
83596. 114KB
83597. 105KB 4KB 32KB trigger compaction if there are ≥ 4 files
83598. Generate one more file at level-0, which should trigger level-0 compaction.
83599. Write 100KB (100 values, each 1K)
83600. Flush whatever is remaining in memtable. This is typically small, which should not trigger size ratio based compaction but will instead trigger size amplification.
83601. for single-level universal, everything's bottom level so nothing should be executed in bottom-pri thread pool.
83602. 32KB trigger compaction if there are ≥ 4 files
83603. Stage 4: Now we have 3 files at level 0, with size 4, 2.4, 2. Let's generate a new file of size 1.
83604. **comment:** TODO(kailiu) The tests on UniversalCompaction has some issues: 1. A lot of magic numbers ("11" or "12"). 2. Made assumption on the memtable flush conditions, which may change from time to time.
label: code-design
83605. 200KB
83606. The first compaction (2) is compressed.
83607. Full compaction to DB path 0
83608. Unblock compaction and wait it for happening.
83609. Write 7 files to trigger compaction
83610. (1, 4)
83611. Stage 3: Now we have 3 files at level 0, with size 4, 0.4, 2. Generate one more file at level-0, which should trigger level-0 compaction.
83612. (1, 3, 8) \rightarrow (4, 8)
83613. Verify that size amplification did occur
83614. (1,1,4) \rightarrow (2, 4)
83615. First compaction should output to bottom level. Second should output to L0 since older L0 files pending compaction prevent it from being placed lower.
83616. 105KB 4KB 32KB
83617. allow_ingest_behind increases number of levels while sanitizing.
83618. (1, 3, 4) \rightarrow (8)
83619. Before compaction, we have 4 files at level 0, with size 4, 0.4, 1, 1. After compaction, we should have 3 files, with size 4, 0.4, 2.
83620. Delay every compaction so multiple compactations will happen.
83621. Stage 2: reopen with universal compaction, num_levels=4
83622. Stage 3: Revert it back to one level and revert to num_levels=1.
83623. Stage 1: open a DB with universal compaction, num_levels=1
83624. (1, 8)
83625. again both values should be present.
83626. 4 memtable are not flushed, 1 sst file
83627. Reopen, insert and flush.
83628. First 4 keys goes to separate SSTs + 1 more SST for 2 smaller keys
83629. Reopen twice and validate.
83630. Make sure that if we re-open with a small write buffer size that we flush table files in the middle of a large log file.
83631. Verify clean slate behavior
83632. Flush() triggers deletion of obsolete tracked files
83633. Check, that records for 'default', 'dobrynia' and 'pikachu' from first, second and third WALs went to the same SST. So, there is 6 SSTs: three for 'nikitich', one for 'default', one for 'dobrynia', one for 'pikachu'
83634. seq id 4
83635. verify
83636. Simulate a crash.
83637. Verify behavior
83638. fill with new date
83639. Manually corrupt the specified WAL
83640. New WAL file
83641. ROCKSDB_LITE
83642. Create corrupted WAL
83643. assert that we successfully recovered only from logs, even though we destroyed the DB
83644. Test scope: - We expect to open data store under all circumstances - We expect only data upto the point where the first error was encountered
83645. Destroy DB before destruct fault_env.
83646. Read back all the keys we wrote and return the number of keys found
83647. corrupt the wal
83648. Reopen DB. Check if WAL logs flushed.
83649. Corruption offset position
83650. len=%
83651. 1 SST for big key + 1 SST for small one
83652. Starting number for the WAL file name like 00010.log
83653. Write to new WAL file
83654. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83655. Create WAL files with values filled in
83656. Corrupt the WAL
83657. Insert some data without flush
83658. Prior to the fix, we may incorrectly recover "v5" with sequence id = 3.
83659. Since we will reopen DB with smaller write_buffer_size, each key will go to new SST file
83660. There are still 4 memtable not flushed, and 2 sst tables
83661. Append some more data.
83662. **comment:** On windows long is 32-bit
label: code-design
83663. Github issue 1339. Prior the fix we read sequence id from the first log to a local variable, then keep increase the variable as we replay logs, ignoring actual sequence id of the records. This is incorrect if some writes come with WAL disabled.
83664. Reopen DB. WAL logs should not be flushed this time.
83665. WAL file
83666. fill up the DB
83667. Test scope: - We expect to open the data store under all scenarios - We expect to have recovered records past the corruption zone
83668. Both value's should be present.
83669. manual flush and insert again
83670. make sure we can re-open it.
83671. Number of WAL files to generate
83672. Test scope: We don't expect the data store to be opened if there is any corruption (leading, middle or trailing -- incomplete writes or corruption)
83673. Moving this to SyncWAL before the actual fsync TEST_SYNC_POINT("DBWALTest::SyncWALNotWaitWrite:1");
83674. Keys to be written per WAL file

83675. Corruption offset
83676. In this test we are trying to do the following: 1. Create a DB with corrupted WAL log; 2. Open with avoid_flush_during_recovery = true; 3. Append more data without flushing, which creates new WAL log. 4. Open again. See if it can correctly handle previous corruption.
83677. Corrupt the wal
83678. seq id 1
83679. Recreate and fill the store with some data
83680. Recovery will fail if DB directory doesn't exist.
83681. Fill data for testing
83682. wal files
83683. Test with flush after recovery.
83684. No inserts => default is empty
83685. Create DB with multiple column families and multiple log files.
83686. namespace rocksdb
83687. Overwrite data with 'a' from offset for length len
83688. wal=
83689. this should ignore the log files, recovery should not happen again if the recovery happens, the same merge operator would be called twice, leading to incorrect results
83690. seq id 5
83691. Test without flush after recovery.
83692. Moving this to SyncWAL after actual fsync TEST_SYNC_POINT("DBWALTest::SyncWALNotWaitWrite:2");
83693. Make 'nikitich' memtable to be flushed
83694. Test scope: - We expect to open the data store when there is incomplete trailing writes at the end of any of the logs - We do not expect to open the data store for corruption
83695. Insert again and reopen
83696. Now the original WAL is in log_files[0] and should be marked for recycling. Verify full purge cannot remove this file.
83697. In https://reviews.facebook.net/D20661 we change recovery behavior: previously for each log file each column family memtable was flushed, even it wasn't empty.
Now it's changed: we try to create the smallest number of table files by merging updates from multiple logs
83698. Save data for comparison.
83699. Skip the test if DB won't open.
83700. force
83701. Test WAL recovery for the various modes available
83702. copy the logs to backup
83703. delete old files in backup_logs directory
83704. Memtable for 'nikitich' has flushed, new WAL file has opened 4 memtable still not flushed
83705. copy the logs from backup back to wal dir
83706. off=
83707. make it flush
83708. Reopen. Verify data.
83709. Make sure 'dobrynia' was flushed: check sst files amount
83710. Probe data for invariants
83711. 1 SST for all keys
83712. recover the DB
83713. For github issue #1303
83714. Fill up 'nikitich' one more time
83715. Verifies WAL files that were present during recovery, but not flushed due to avoid_flush_during_recovery, will be considered for deletion at a later stage. We check at least one such file is deleted during Flush().
83716. **comment:** Check at least the first WAL was cleaned up during the recovery.
label: code-design
83717. Force flush with allow_2pc.
83718. Size of the value
83719. !(defined(NDEBUG) || !defined(OS_WIN))
83720. Make 'dobrynia' to be flushed and new WAL file to be created
83721. test checksum failure or parsing
83722. Offset of corruption
83723. Windows disk cache behaves differently. When we truncate the original content is still in the cache due to the original handle is still open. Generally, in Windows, one prohibits shared access to files and it is not needed for WAL but we allow it to induce corruption at various tests.
83724. **comment:** Test for regression of WAL cleanup missing files that don't contain data for every column family.
label: code-design
83725. **comment:** Corruption style
label: code-design
83726. Insert more data.
83727. In https://reviews.facebook.net/D20661 we change recovery behavior: previously for each log file each column family memtable was flushed, even it was empty.
Now it's changed: we try to create the smallest number of table files by merging updates from multiple logs
83728. Verify
83729. we won't be needing this file no more
83730. Test variations of WriteImpl.
83731. Check each sequence is used once and only once.
83732. Sequence number should be return through input write batch.
83733. The sequence isn't consumed by someone else.
83734. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
83735. namespace rocksdb
83736. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83737. When user keys are same
83738. When limit user key is prefix of start user key
83739. Size going up, memory allocation is triggered
83740. When start user key is prefix of limit user key
83741. namespace rocksdb
83742. When user keys are misordered
83743. When user keys are different, but correctly ordered
83744. namespace rocksdb
83745. **comment:** clean up all the files that might have been there before
label: code-design
83746. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

83747. 2 ssts, 1 manifest
83748. Make sure all background purges are executed
83749. 1 sst after iterator deletion
83750. 3 sst after compaction with live iterator
83751. ~DBImpl should wait until all BGWorkPurge are finished
83752. this time, we keep an iterator alive
83753. Take the last log file which is expected to be alive and try to delete it Should not succeed because live logs are not allowed to be deleted
83754. Call Flush to bring about a new working log file and add more keys Call Flush again to flush out memtable and move alive log to archived log and try to delete the archived log file
83755. Make sure no purges are executed foreground
83756. !ROCKSDB_LITE
83757. intermediate level files cannot be deleted.
83758. This test is to reproduce a bug that read invalid ReadOption in iterator cleanup function
83759. COntrolled setup. Levels 1 and 2 should both have 50K files. This is a little fragile as it depends on the current compaction heuristics.
83760. 5 sst files after 2 compactions with 2 live iterators
83761. always do full purge
83762. Used to test log files Used to test log files
83763. An empty job to guard all jobs are processed
83764. there should be only one (empty) log file because CreateTwoLevels() flushes the memtables to disk
83765. 1 sst after compaction
83766. We keep an iterator alive
83767. **comment:** ReadOptions is deleted, but iterator cleanup function should not be affected
 label: code-design
83768. Lowest level file deletion should succeed.
83769. file3.sst (110 => 124) .. overlap with file2.sst
83770. Write some keys through normal write path
83771. File dont overwrite any keys, No seqno needed
83772. we only use TestPutOperator in this test
83773. overlaps with memtable, so flush is triggered (thus file count increases by two at this step).
83774. namespace rocksdb
83775. copy file
83776. file2.sst (100 => 299)
83777. File overwrite some keys, a seqno will be assigned
83778. move file
83779. fadvise disabled
83780. Add file using file path
83781. We will need a seqno for the file regardless if the file overwrite keys in the DB or not because we have a snapshot
83782. A global seqno will be assigned anyway because of the snapshot
83783. fadvise enabled
83784. Current file size should be 0 after sst_file_writer init and before open a file.
83785. ROCKSDB_LITE
83786. overlaps with nothing, so places at bottom level and skips incrementing seqnum.
83787. overlaps with L0 file but not memtable, so flush is skipped
83788. prevent range deletions from being dropped due to becoming obsolete.
83789. This file have overlapping values with the existing data
83790. snapshot unneeded now that both range deletions are persisted
83791. No snapshot anymore, no need to assign a seqno
83792. sst_file_writer already finished, cannot add this value
83793. Current file size should be non-zero after success write.
83794. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
83795. file1.sst (0 => 99)
83796. range del [0, 50) in L0 file, [50, 100) in memtable
83797. check that we have change the old key
83798. Start ingesting and extrnal file in the background
83799. This file overlap with files in L4, we will ingest it in L3
83800. We intentionally add every file twice, and assert that it was added only once and the other add failed
83801. Verify data in files
83802. L3 L2 L1 L0
83803. L3 L2
83804. SST CF unknown
83805. Flush / Compact the DB
83806. file5.sst (200 => 299)
83807. Add file while holding a snapshot will fail
83808. file4.sst (30 => 39) This file values overlap with file1 values
83809. This thread will try to insert into the memtable but since we have 4 L0 files this thread will be blocked and hold the writer thread
83810. This file has overlapping values with the existing data
83811. Start a thread that will ingest a new file
83812. SST CF match
83813. files[0].sst (0 => 99) files[1].sst (100 => 199) ... file[8].sst (800 => 899)
83814. this time ingest should fail as the file doesn't fit to the bottom level
83815. Use DB::AddFile to insert range
83816. Make sure DelayWrite is called first
83817. file5.sst (400 => 499)
83818. Thread 0 -> Load {f0,f1} Thread 1 -> Load {f0,f1} Thread 2 -> Load {f2,f3} Thread 3 -> Load {f2,f3} Thread 4 -> Load {f4,f5} Thread 5 -> Load {f4,f5} ...
83819. Now let compaction start
83820. Generate the file containing the range
83821. We can add the file after releasing the snapshot
83822. Generate file names
83823. Can't ingest behind since allow_ingest_behind isn't set to true
83824. Write range [5 => 10] to L0
83825. CF was not dropped, ok to Ingest
83826. Flush and read from L0
83827. These 2 files dont overlap with each other
83828. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
83829. file3.sst (300 => 399)
83830. file #1 in L0

83831. This test reproduce a bug that can happen in some cases if the DB started purging obsolete files when we are adding an external sst file. This situation may result in deleting the file while it's being added.

83832. Universal picker should go at second from the bottom level

83833. This range overlap with data already exist in DB

83834. Current file size should be 0 after sst_file_writer init and before open a file.

83835. Use DB::Put to insert range (insert into memtable)

83836. Wait for IngestExternalFile() to start and aquire mutex

83837. This file dont overlap with anything in the DB, will go to L4

83838. Now we can add the file

83839. file #2 in L0

83840. Generate a file id if not provided

83841. This file does not overlap with any file or with the running compaction

83842. This file dont overlap with anything in the DB and fit in L4 as well

83843. Insert 100 -> 200 using IngestExternalFile

83844. Update true_data map to include the new inserted data

83845. Output of the compaction will go to L3

83846. Insert 30 -> 50 using AddFile

83847. SST CF dont match

83848. Add file using file path

83849. atomic counter of currently running bg threads

83850. Write num_files files in parallel

83851. Bulk load 10 files every file contain 1000 keys

83852. This file overlap with the memtable, so it will flush it and add it self to L0

83853. File 7 overlaps with file 4 (L3)

83854. Sort data if asked to do so

83855. file2.sst (100 => 199)

83856. Delete keys in range (200 => 299)

83857. While writing the MANIFEST start a thread that will ask for compaction

83858. cf1.sst

83859. Verify values of file5 in DB

83860. Current file size should be non-zero after success write.

83861. file1.sst (0 => 500)

83862. Generate files with these key ranges {14 -> 0} {24 -> 10} {34 -> 20} {44 -> 30} ..

83863. This file overlaps with the output of the compaction (going to L3) so the file will be added to L0 since L3 is the base level

83864. Insert 80 -> 130 using AddFile

83865. bottom level should be empty

83866. 25% of writes go through memtable

83867. Ingest into cf1

83868. We need to verify that no compactions can run while AddFile is ingesting the files into the levels it find suitable. So we will wait for 2 seconds to give a chance for compactions to run during this period, and then make sure that no compactions where able to run

83869. This file does not overlap with the current running compacting

83870. Make sure the values are correct before and after flush/compaction

83871. Insert 100 -> 200 into the memtable

83872. namespace rocksdb

83873. File 5 overlaps with file 2 (L3 / base level)

83874. file2.sst (100 => 299)

83875. Compacting the DB will remove the tombstones

83876. No need for flush, this file keys fit between the memtable keys

83877. File 1 will go to level L2 (since it overlap with file 0 in L3)

83878. Files dont overlap and dont overlap with DB key range

83879. Overwrite what we added using external files

83880. cf_unknown.sst

83881. list 1 has internal key range conflict

83882. check that we still can open the DB, as num_levels should be sanitized to 3

83883. Flush 4 files containing the same keys

83884. Hold AddFile from finishing writing the MANIFEST

83885. fit in L3 but will overlap with compaction so will be added to L2 but a compaction will trivially move it to L3 and break LSM consistency

83886. IROCKSDB_LITE

83887. `DBImpl::AddFile::Start` will wait until we be here

83888. We have 2 overlapping files in L0

83889. Wait for BackgroundCompaction() to be called

83890. Cannot add this key because it's not after last added key

83891. No Flush needed, but need a global seqno, Ingest in L0

83892. Overwrite values of keys divisible by 100

83893. default_cf.sst

83894. Verify the correctness of the data

83895. Read values from memtable

83896. Destruct SstFileWriter with a failing Finish

83897. file3.sst (195 => 199) This file values overlap with file2 values

83898. Delete keys in range (400 => 499)

83899. Insert 10 -> 40 using AddFile

83900. These 2nd and 3rd files overlap with each other

83901. sst_file_writer already finished, cannot add this value

83902. Current file size should be non-zero after success finish.

83903. file1.sst (0 => 99)

83904. This list of files have key ranges are overlapping with each other

83905. sometimes we use copy, sometimes link .. the result should be the same

83906. Insert 0 -> 20 using AddFile

83907. Add files using file path list

83908. We deleted range (200 => 299) but cannot add file5 because of the range tombstones

83909. This file will flush the memtable

83910. Make sure values are correct before and after flush/compaction

83911. No Flush needed, No global seqno needed, Ingest in L1

83912. Wait for max 5 seconds, if we did not finish all bg threads then we hit the deadlock bug

83913. Wait for AddFile to start picking levels and writing MANIFEST

83914. This file overlaps with file 0 (L3), file 1 (L2) and the output of compaction going to L1

83915. Ingest into cf2

83916. Cannot ingest a file into a dropped CF

83917. This file list has overlapping values with the existing data
83918. Insert the generated file
83919. Add file will fail when holding snapshot and use the default skip_snapshot_check to false
83920. Insert keys using normal path and take a snapshot
83921. No need for flush
83922. Bulk load num_files files in parallel
83923. Key range of file5 (400 => 499) dont overlap with any keys in DB
83924. File 6 overlaps with file 2 (L3 / base level) and file 5 (L0)
83925. Destuct SstFileWriter without calling Finish()
83926. Overwrite values of keys divisible by 5
83927. Ingest into default cf
83928. Overwrite all keys using IngestExternalFile
83929. Cannot create an empty sst file
83930. file3.sst (195 => 299) This file values overlap with file2 values
83931. File 0 will go to last level (L3)
83932. Add file will success when set skip_snapshot_check to true even db holding snapshot
83933. For 20% of ranges we use DB::Put, for 80% we use DB::AddFile
83934. We deleted range (400 => 499) but cannot add file5 because of the range tombstones
83935. Hold compaction from finishing
83936. These 2 files dont overlap with each other but overlap with keys in DB
83937. Return the current option configuration.
83938. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright 2014 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83939. This test uses a custom Env to keep track of the state of a filesystem as of the last "sync". It then checks for data loss errors by purposely dropping file data (or entire files) not protected by a "sync".
83940. When need to make sure data is persistent, sync WAL
83941. Return the ith key
83942. rnd cannot be null for kResetDropRandomUnsyncedData
83943. 100KB
83944. Block the job queue to prevent flush job from running.
83945. Setting a separate data path won't pass the test as we don't sync it after creating new files,
83946. When need to make sure data is persistent, call DB::CompactRange()
83947. random transfer
83948. Return the value to associate with the specified key
83949. namespace rocksdb
83950. Previous log file is not fsynced if sync is forced after log rolling.
83951. No new files created so we expect all values since no files will be dropped.
83952. Case 4: mixed
83953. Case 1: no overlap, files are on the left of next level files
83954. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83955. namespace rocksdb
83956. level 2 0 1 2 3 4 level 3
83957. level 1, 0
83958. level 2, 1
83959. level 2, [2 - 4], no overlap
83960. level 2
83961. Case 2: no overlap, files are on the right of next level files
83962. Case 3: empty L2
83963. level 1, 1
83964. level 1, 2
83965. level 2, 0
83966. level 3
83967. level 1
83968. Case 0: Empty
83969. namespace rocksdb
83970. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
83971. Successful parses
83972. Errors
83973. range tombstone end key
83974. a key is visible only if: 1. it's the last one written (j == insertions - 1) 2. there's a snapshot pointing at it
83975. **comment:** TODO(icanadi) Remove this once we mock out VersionSet
label: requirement
83976. range tombstone seqnum 10000
83977. Make "CURRENT" file that points to the new manifest file.
83978. **comment:** TODO(icanadi) Mock out everything else: 1. VersionSet 2. Memtable
label: requirement
83979. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
83980. namespace rocksdb
83981. Test data: seqno [1, 2 ... 8998, 8999, 9000, 9001, 9002 ... 9999] key [1001, 1002 ... 9998, 9999, 0, 1, 2 ... 999] range-delete "9995" -> "9999" at seqno 10000
83982. Write 8 files in L0
83983. suppress the first error and disable write-dropping such that a retry can succeed.
83984. BG compaction is disabled. Number of L0 files will simply keeps increasing in this test.
83985. trigger flush so compaction is triggered again; this time it succeeds
83986. Small write buffer
83987. db
83988. the usual TEST_WaitForFlushMemTable() doesn't work for failed flushes, so forge a custom one for the failed flush case.
83989. namespace rocksdb
83990. verify whether the previously created file matches the flushed file.
83991. make sure call-back functions are called in the right order
83992. Do a trivial move from L0 -> L1
83993. ROCKSDB_USING_THREAD_STATUS
83994. ROCKSDB_LITE

83995. This simple Listener can only handle one flush at a time.
83996. keep writing until writes are forced to stop.
83997. Verify the id of the current thread that created this table file matches the id of any active flush or compaction thread.
83998. Write 3 non-overlapping files in L0
83999. Write 4 files in L0
84000. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84001. remember the info for later checking the FlushJobInfo.
84002. third iteration triggers the second memtable's flush
84003. Least significant size byte is stored in header[4].
84004. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84005. Drop all payload as well as a header byte
84006. Remove the LAST block, including header.
84007. Write records that span two blocks
84008. Return a skewed potentially long string
84009. If an error occurs during Read() in UnmarkEOF(), the records contained in the buffer should be returned on subsequent calls of ReadRecord() until no more full records are left, whereafter ReadRecord() should return false to indicate that it cannot read any further.
84010. test is only valid for recycled logs
84011. checksum
84012. Returns OK iff recorded error message contains "msg"
84013. initial_offset
84014. Two sizable records in first block
84015. Construct a string of the specified length made out of the supplied partial string.
84016. Compute crc of type/len/data
84017. Make a trailer that is exactly the same length as an empty record.
84018. Span three blocks
84019. namespace log namespace rocksdb
84020. Truncated last record is ignored, not treated as an error
84021. Type is stored in header[6]
84022. Consider two fragmented records: first(R1) last(R1) first(R2) last(R2) where the middle two fragments disappear. We do not want first(R1),last(R2) to get joined and returned as a valid record.
84023. Tests of all the error paths in log_reader.cc follow:
84024. Cause a bad record length in the LAST block.
84025. Wipe the middle block
84026. Construct a string from a number
84027. Record metadata for testing initial offset functionality
84028. By using scratch we ensure that caller has control over the lifetime of result.data()
84029. Make sure reads at eof work
84030. compact database
84031. universal compaction
84032. create first key range
84033. close database
84034. create second key range
84035. Get rid of any state from an old run.
84036. level compaction
84037. delete second key range
84038. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Test for issue 178: a manual compaction causes deleted data to reappear.
84039. commenting out the line below causes the example to work correctly
84040. Open database. Disable compression since it affects the creation of layers and the code below is trying to test against a very specific scenario.
84041. count the keys
84042. anonymous namespace
84043. Create a mock VersionSet
84044. Add a third memtable to push the first memtable out of the history
84045. Create an empty MemTableList and validate basic functions.
84046. Create a test db if not yet created
84047. Attempt to 'flush' to clear request for flush
84048. namespace rocksdb
84049. Verify keys are no longer in MemTableList
84050. Note: now to_flush contains tables[0,1,2,4]. to_flush2 contains tables[3]. Current implementation will only commit memtables in the order they were created. So InstallMemtableFlushResults will install the first 3 tables in to_flush and stop when it encounters a table not yet flushed.
84051. Calls MemTableList::InstallMemtableFlushResults() and sets up all structures needed to call this function.
84052. **comment:** nothing in progress of being flushed
 label: requirement
84053. Request a flush again
84054. column_family_id
84055. Create some MemTables
84056. Add another table
84057. No flush pending since the list is empty.
84058. Create a mock Logger
84059. Request a flush even though there is nothing to flush
84060. Request a flush again. Should be nothing to flush
84061. Verify that key2 from the first memtable is no longer in the history
84062. Verify that the second memtable's keys are in the history
84063. This will actually install 2 tables. The 1 we told it to flush, and also tables[4] which has been waiting for tables[3] to commit.
84064. Create a MemTable
84065. Flush second memtable
84066. Verify keys are present in history
84067. Should pick 4 of 5 since 1 table has been picked in to_flush2
84068. We now have the minimum to flush regardless of whether FlushRequested()
84069. **comment:** Even though we have less than the minimum to flush, a flush is pending since we had previously requested a flush and never called PickMemtablesToFlush() to clear the flush.
 label: code-design
84070. Create MemTableList
84071. Pick tables to flush

84072. Add another tables
84073. Fetch the newly written keys
84074. Flush the 4 memtables that were picked in to_flush
84075. Add 2 tables
84076. We now have the minimum to flush regardless of whether FlushRequested() was called.
84077. Add second memtable to list
84078. Create dummy mutex.
84079. Write some keys to this memtable.
84080. Add memtable to list
84081. Nothing to flush
84082. MemTable found out that this key is *not* found (at this sequence#)
84083. Flush the 1 memtable that was picked in to_flush2
84084. Revert flush
84085. **comment:** Cleanup
 label: code-design
84086. Rollback first pick of tables
84087. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84088. snapshots
84089. Flush this memtable from the list. (It will then be a part of the memtable history).
84090. Create another memtable and write some keys to it
84091. Refcount should be 0 after calling InstallMemtableFlushResults. Verify this, by Ref'ing then UnRef'ing:
84092. Fetch keys via MemTableList
84093. Pick tables to flush again
84094. Create mock default ColumnFamilyData
84095. Make sure that merge operands are not filtered out if there's a snapshot pointing at them
84096. Merging with a value results in a successful merge.
84097. The merge helper stops upon encountering a corrupt key
84098. we have one operand that will survive because it's a delete
84099. Filtered
84100. make sure that we're passing user keys into the filter
84101. namespace rocksdb
84102. A single operand can not be merged.
84103. <- iter_after merge
84104. Merging with a deletion turns the deletion into a value
84105. sequence number is 29 here, because the first merge operand got filtered out
84106. when all merge operands are filtered out, we leave the iterator pointing to the Put/Delete that survived
84107. Filtered Filtered next user key
84108. range_del_agg
84109. filtered out all
84110. Make sure that merge operands are filtered at the beginning
84111. Filtered Filtered
84112. The compaction filter is called on every merge operand
84113. Corrupt key <- iter_after merge
84114. MergeHelper preserves the operand stack for merge operators that cannot do a partial merge.
84115. Merging stops before a snapshot.
84116. If MergeHelper encounters a new key on the last level, we know that the key has no more history and it can merge keys.
84117. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84118. Implement 'add' directly with the new Merge operation
84119. in this case, FullMerge should be called instead.
84120. Imagine we are maintaining a set of uint64 counters. Each counter has a distinct name. And we would like to support four high level operations: set, add, get and remove
 This is a quick implementation without a Merge operation.
84121. just treat the internal rep of int64 as the string
84122. Apply to memtable and count the number of merges
84123. for merge
84124. mapped to a rocksdb Delete
84125. namespace
84126. DBWithTTL is not supported in ROCKSDB_LITE
84127. Run test on TTL database !ROCKSDB_LITE
84128. public interface of Counters. All four functions return false if the underlying level db operation failed.
84129. **comment:** TODO: Make this test like a general rocksdb unit-test
 label: test
84130. !ROCKSDB_LITE
84131. mapped to a rocksdb Get
84132. Check for any errors found during the scan
84133. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84134. 1+...+49 = ?
84135. Test case 1: partial merge should be called when the number of merge operands exceeds the threshold.
84136. Create the batch
84137. **comment:** Disable unused variable warning.
 label: code-design
84138. mapped to a rocksdb Merge operation
84139. default value is 0 if non-existent
84140. Test case 2: partial merge should not be called when a put is found.
84141. deserialization
84142. **comment:** Min merge is hard-coded to 2.
 label: code-design
84143. mapped to a leveldb Put
84144. if count >= min_merge, then partial merge should be called once.
84145. convenience functions for testing
84146. return default value if not found;
84147. Get the value
84148. Temporary remove this test { std::cout << "Test merge-operator not set after reopen (recovery case)\n"; { auto db = OpenDb(dbname); MergeBasedCounters counters(db, 0); counters.add("test-key", 1); counters.add("test-key", 1); counters.add("test-key", 1); } DB* reopen_db; ASSERT_TRUE(DB::Open(Options(), dbname, &reopen_db).isValidArgument()); }
84149. 'add' is implemented as get -> modify -> set An alternative is a single merge operation, see MergeBasedCounters

84150. 1+2 = 3
84151. Verify whether the current Options Files are the latest ones.
84152. !(defined NDEBUG) || !defined(OS_WIN)
84153. namespace
84154. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84155. namespace rocksdb
84156. Make sure we always keep the latest option files.
84157. !ROCKSDB_LITE
84158. profile the timer cost by itself!
84159. make perf_context_test export ROCKSDB_TESTS=PerfContextTest.SeekKeyComparison For one memtable: ./perf_context_test --write_buffer_size=500000 --total_keys=10000 For two memtables: ./perf_context_test --write_buffer_size=250000 --total_keys=10000 Specify --random_key=1 to shuffle the key before insertion Results show that, for sequential insertion, worst-case Seek Key comparison is close to the total number of keys (linear), when there is only one memtable. When there are two memtables, even the avg Seek Key comparison starts to become linear to the input size.
84160. reset
84161. Issuing a flush in the middle.
84162. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84163. ROCKSDB_LITE
84164. increment the counter only when it's a DB Mutex
84165. Path to the database on file system
84166. In read-only mode Get(), no super version operation is needed
84167. Start this test with a fresh DB
84168. Return the current option configuration.
84169. Destroy using last options
84170. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84171. column_family_id
84172. Return spread of files per level
84173. Neither key nor value should exist.
84174. Write 120KB (10 values, each 12K)
84175. Key doesn't exist any more but prefix exists.
84176. 100KB
84177. namespace
84178. generate one more file in level-0, and should trigger level-0 compaction
84179. namespace rocksdb
84180. Test Bloom Filter
84181. Set only one bucket to force bucket conflict. Test index interval for the same prefix to be 1, 2 and 4
84182. !ROCKSDB_LITE
84183. Compare needs to be aware of the possibility of a and/or b is prefix only
84184. 2. Insert an entry for the same prefix as the last entry in the bucket.
84185. 4. Insert an entry with a larger prefix
84186. Verify seeking past the prefix won't return a result.
84187. GFLAGS
84188. Path to the database on file system
84189. end namespace rocksdb
84190. both a and b are prefix
84191. Insert keys with common prefix and one key with different
84192. skip some options
84193. insert x random prefix, each with y continuous element.
84194. 1. Insert one row.
84195. note, both a and b could be prefix only
84196. 5. Insert an entry with a smaller prefix
84197. namespace
84198. !ROCKSDB_LITE
84199. 6. Insert to the beginning and the end of the first prefix
84200. Show results in prefix
84201. Only for SkipListFactory
84202. one of them is prefix
84203. test non-existing keys
84204. return a slice backed by test_key
84205. 3. Insert an entry for the same prefix as the head of the bucket.
84206. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84207. test seek existing keys
84208. both a and b are whole key
84209. Only for SkipListFactory test SeekToLast() with iterate_upper_bound_in prefix_seek_mode
84210. Only for SkipListFactory test the case iter1 iter2 | prefix | suffix | | prefix | suffix | | 1 | 1 | 1 | 2 | | 1 | 3 | | 1 | 4 | | 2 | 1 | | 3 | 3 | | 2 | 2 | | 3 | 4 | after seek(15), iter1 will be at 21 and iter2 will be 33. Then if call Prev() in prefix mode where SeekForPrev(21) gets called, iter2 should turn to invalid state because of bloom filter.
84211. Test same result regardless of which order the range deletions are added.
84212. Copyright (c) 2016-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84213. Note the Cover* tests also test cases where tombstones are inserted under a larger one when VerifyRangeDels() runs them in reverse
84214. snapshots
84215. namespace rocksdb
84216. anonymous namespace
84217. Leave one unflushed so we can verify RepairDB's flush logic
84218. Examine table properties to verify RepairDB() used the right options when converting WAL->SST
84219. Need to get path before Close() deletes db_, but delete it after Close() to ensure Close() doesn't re-create the manifest.
84220. ROCKSDB_LITE namespace rocksdb
84221. Exactly one of the key-value pairs should be in the DB now.
84222. Need to get path before Close() deletes db_, but overwrite it after Close() to ensure Close() didn't change the manifest.
84223. Corrupt one of the SST files but preserve the manifest that refers to it, then verify the DB is still usable for the intact SST.
84224. Verify repair logic uses correct ColumnFamilyOptions when repairing a database with different options for column families.
84225. Replace the manifest with one that is only aware of the first SST file.
84226. Add a couple SST files, delete the manifest, and verify RepairDB() saves the day.
84227. unknown_cf_opts

84228. ROCKSDB_LITE

84229. make sure that all WALs are converted to SSTables.

84230. This test case invokes repair while some data is unflushed, then verifies that data is in the db.

84231. In this case, the manifest is valid but does not reference all of the SST files. Expect a full recovery.

84232. Leave one unflushed so we can verify WAL entries are properly associated with column families.

84233. Delete one of the SST files but preserve the manifest that refers to it, then verify the DB is still usable for the intact SST.

84234. Need to get path before Close() deletes db_, but delete it after Close() to ensure Close() didn't change the manifest.

84235. RepairDB() records the comparator in the manifest, and DB::Open would fail if a different comparator were used.

84236. Copyright (c) 2016-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84237. Verify repair logic associates SST files with their original column families.

84238. Also check comparator when it's provided via "unknown" CF options

84239. Manifest is in an invalid format. Expect a full recovery.

84240. -- Step 2: Read properties

84241. not sanitize

84242. PlainTable is not supported in Lite

84243. namespace rocksdb

84244. make sure the entries will be inserted with order.

84245. Test properties collectors with internal keys or regular keys for block based table

84246. Collects keys that starts with "A" in a table.

84247. namespace

84248. !ROCKSDB_LITE

84249. PlainTable is not supported in Lite test plain table

84250. **comment:** HACK: Set options.info_log to avoid writing log in SanitizeOptions().

label: code-design

84251. starts with 'A' starts with 'A' starts with 'A'

84252. Collects keys that starts with "A" in a table. Backward compatible mode It is also used to test internal key table property collector

84253. deletes + single-deletes

84254. Utilities test functions

84255. with sanitization, even regular properties collector will be able to handle internal keys.

84256. sanitize

84257. simply assume all user keys are not empty.

84258. -- Step 1: build table

84259. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84260. just a place holder

84261. namespace rocksdb

84262. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84263. minus 2X for the number of deletion entries because: 1x for deletion entry does not count as a data entry. 1x for each deletion entry will actually remove one data entry.

84264. Call back function to add extra customized builds.

84265. namespace rocksdb

84266. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

84267. namespace rocksdb

84268. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

84269. Level 0 is not ordered Ignored because of [5,6] in l1 Ignored because of [2,3] in l2 Ignored because of [2,3] in l2

84270. Test whether the overlaps are detected as expected Perfect overlap with last level Partial overlap with last level Partial overlap with last level Contains range of last level Inside range of last level Inside range of last level

84271. add to file_level_

84272. **comment:** NOT thread safe

label: requirement

84273. **comment:** TODO(icanadi) mock out VersionSet TODO(icanadi) move other WalManager-specific tests from db_test here

label: test

84274. did a read TODO(icanadi) move SpecialEnv outside of db_test, so we can reuse it here ASSERT_EQ(env_->sequential_read_counter_.Read(), 1);

84275. TEST : Create WalManager with a ttl and no size limit. Create some archived log files and call PurgeObsoleteWALFiles(). Assert that files are not deleted Reopen db with small ttl. Assert that all archived logs was removed.

84276. !ROCKSDB_LITE

84277. Create a zero record WAL file.

84278. Check that an empty iterator is returned

84279. number

84280. no new reads since the value is cached TODO(icanadi) move SpecialEnv outside of db_test, so we can reuse it here ASSERT_EQ(env_->sequential_read_counter_.Read(), 1);

84281. namespace

84282. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84283. namespace rocksdb

84284. TEST : Create WalManager with huge size limit and no ttl. Create some archived files and call PurgeObsoleteWALFiles(). Count the archived log files that survived. Assert that all of them did. Change size limit. Re-open WalManager. Assert that archive is not greater than wal_size_limit_mb after PurgeObsoleteWALFiles() Set ttl and time_to_check_ to small values. Re-open db. Assert that there are no archived logs left.

84285. TODO(icanadi) move SpecialEnv outside of db_test, so we can reuse it here. Waiting for lei to finish with db_test env_->count_sequential_reads_ = true; sequential_read_counter_ sanity test ASSERT_EQ(env_->sequential_read_counter_.Read(), 0);

84286. column_family_id

84287. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

84288. Try a write where the key is one slice but the value is two

84289. One where the key is composite but the value is a single slice

84290. It requires more than 30GB of memory to run the test. With single memory allocation of more than 30GB. Not all platform can run it. Also it runs a long time. So disable it.

84291. Insert key and value of 3GB and push total batch size to 12GB.

84292. The test requires more than 18GB memory to run it, with single memory allocation of more than 12GB. Not all the platform can run it. So disable it.

84293. namespace anonymous

84294. The header size is 12 bytes. The two Puts take 8 bytes which gives total of $12 + 8 * 2 = 28$ bytes.

84295. wal only
84296. namespace rocksdb
84297. IROCKSDB_LITE
84298. Wait here until all verification in this sync-point callback finish for all writers.
84299. Writers that have linked to the queue
84300. Make sure db is a DBImpl
84301. namespace rocksdb
84302. Wait for the last joined writer to link to the queue. In this way the writers link to the queue one by one. This allows us to confidently detect the first writer who increases threads_linked as the leader.
84303. Verification once writers call JoinBatchGroup.
84304. Test a simple Write
84305. Test WriteWithCallback
84306. check my state
84307. check for keys
84308. In each scenario we'll launch multiple threads to write. The size of each array equals to number of threads, and each boolean in it denote whether callback of corresponding thread should succeed or fail.
84309. who am i
84310. !ROCKSDB_LITE
84311. insert some keys
84312. grab unique key
84313. leaders gotta lead
84314. Writers that have called JoinBatchGroup.
84315. Test WriteWithCallback for a callback that fails
84316. Writers that pass WriteThread::JoinBatchGroup::Wait sync-point.
84317. no batching so everyone should be a leader
84318. (meta test) the first WriteOP should indeed be the first and the last should be the last (all others can be out of order)
84319. do all the writes
84320. Each write thread create a random write batch and write to DB with a write callback.
84321. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84322. loser has to lose
84323. This is more than max rate. Max delayed rate will be used.
84324. also set max delayed rate
84325. 1720 left
84326. 4170 left
84327. 1720 balance + 20000 credit = 20170 left Use 8000, 12170 left
84328. sleep credit 200 One refill: 10240 filed, sleep credit generates 2000. 8000 used 7240 + 10240 + 2000 - 8000 = 11480 left
84329. namespace rocksdb
84330. sleep credit 2000
84331. 1000 used, 8240 left
84332. sleep credit 300 One refill, credit 4480 balance + 3000 credit + 10240 refill Use 8000, 9720 left
84333. One refill: 10240 bytes allowed, 1000 used, 9240 left
84334. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84335. sleep debt 624 Out of bound sleep, still 10480 left
84336. sleep debt 0
84337. sleep credit 100 1000 used, 7240 left
84338. sleep credit 100 6000 used, 4480 left.
84339. Need a refill
84340. sleep debt 824 1000 used, 10480 left.
84341. sleep debt 1000
84342. Rate reset after changing the token.
84343. Read "foo".
84344. empty folder returns empty vector
84345. if dir is a file, returns IOError
84346. Returns a vector of 0 or 1 Env*, depending whether an Env is registered for TEST_ENV_URI. The purpose of returning an empty vector (instead of nullptr) is that gtest ValuesIn() will skip running tests when given an empty collection.
84347. Normalizes trivial differences across Envs such that these test cases can run on all Envs.
84348. Check that the file exists.
84349. necessary for default POSIX env
84350. Read "d".
84351. namespace rocksdb
84352. Read "hello".
84353. anonymous namespace
84354. Check that the directory is empty.
84355. non-exist directory returns IOError
84356. Write to the file.
84357. Try to skip past end of file.
84358. don't know whether it's file or directory, try both. The tests must only create files or empty directories, so one must succeed, else the directory's corrupted.
84359. Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84360. Removes . and .. from directory listing
84361. Check that opening non-existent file fails.
84362. Random reads.
84363. Check that renaming works.
84364. Read "world".
84365. Too high offset.
84366. ROCKSDB_LITE
84367. Check that deleting works.
84368. Read sequentially.
84369. fail because file already exists
84370. Check that renaming overwriting works
84371. These are no-ops, but we test they return success.
84372. Check for expected size.
84373. Create a file.
84374. folder with contents returns relative path to test dir
84375. Try reading past EOF.
84376. Create file.

84377. To ensure that Env::GetUniqueId-related tests work correctly, the files should be stored in regular storage like "hard disk" or "flash device", and not on a tmpfs file system (like /dev/shm and /tmp on some systems). Otherwise we cannot get the correct id. This function serves as the replacement for test::TmpDir(), which may be customized to be on a file system that doesn't work with GetUniqueId().

84378. Write a page worth of 'b' right after the first sector

84379. Shrink to 1 thread

84380. If \$TEST_IOCTL_FRIENDLY_TMPDIR/rocksdb.XXXXXX fits, use \$TEST_IOCTL_FRIENDLY_TMPDIR; subtract 2 for the "%s", and add 1 for the trailing NUL byte.

84381. The last task finishes. Task 0 running, 2 waiting.

84382. Create the files

84383. Test that all WritableFileWrapper forwards all calls to WritableFile.

84384. Delete the files

84385. call IncBackgroundThreadsIfNeeded to two pools. One increasing and the other decreasing

84386. Write to file

84387. Enqueue 5 more tasks. Thread pool size now is 4. Task 0, 3, 4, 5 running; 6, 7 waiting.

84388. Increase to 5 threads. Task 0 and 2 running.

84389. **comment:** Clean up
label: code-design

84390. Schedule 3 tasks. 0 running; Task 1, 2 waiting.

84391. Only works in linux platforms

84392. Small write should preallocate one block

84393. use \$TEST_IOCTL_FRIENDLY_TMPDIR value

84394. only works in linux platforms

84395. Get Unique ID again after waiting some time.

84396. ignore failure

84397. Wake up the last thread

84398. Try fallocate in a file to see whether the target file system supports it. Skip the test if fallocate is not supported.

84399. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

84400. verify that preallocated blocks were deallocated on file close Because the FS might give us more blocks, we add a full page to the size and expect the number of blocks to be less or equal to that.

84401. Write an entire preallocation block, make sure we increased by two.

84402. !defined(ROCKSDB_LITE) && !defined(OS_WIN)

84403. Remove it with a different tag

84404. Wake up task 0, 3 and 4. Task 5, 6, 7 running.

84405. 48 bytes for extra information + bytes allocated

84406. Data structures to signal tasks to run.

84407. Wait until the thread pools are empty.

84408. namespace

84409. Pointer to shared slot Order# for the execution of this callback

84410. Hold jobs to schedule;

84411. Set number of thread to 1 first.

84412. !defined(ROCKSDB_LITE)

84413. Change number of threads a couple of times while there is no sufficient tasks.

84414. No writes should mean no preallocation

84415. Wake up thread 8

84416. ROCKSDB_FALLOCATE_PRESENT

84417. Wake up thread 9.

84418. Enqueue thread 8 and 9. Task 5 running; one of 8, 9 might be running.

84419. **comment:** Determine whether we can use the FS_IOC_GETVERSION ioctl on a file in directory DIR. Create a temporary file therein, try to apply the ioctl (save that result), cleanup and return the result. Return true if it is supported, and false if anything fails. Note that this function "knows" that dir has just been created and is empty, so we create a simply-named test file: "f".
label: code-design

84420. Shrink back to 1 thread. Still task 0, 1 running, 2 waiting

84421. Remove it from the queue with the right tag

84422. not TRAVIS OS_LINUX || OS_WIN

84423. mkdtemp failed: diagnose it, but don't give up.

84424. Delete the file

84425. namespace rocksdb

84426. Write a page worth of 'a'

84427. Diagnose ioctl-related failure only if this is the directory specified via that envvar.

84428. Returns true if any of the strings in ss are the prefix of another string.

84429. Trigger jobs to run.

84430. Schedule in different order than start time

84431. IROCKSDB_LITE

84432. Travis doesn't support fallocate or getting unique ID from files for whatever reason.

84433. When we have n == -1 there is not a terminating zero expected

84434. verify that blocks are preallocated Note here that we don't check the exact number of blocks preallocated -- we only require that number of allocated blocks is at least what we expect. It looks like some FS give us more blocks than we asked for. That's fine. It might be worth investigating further.

84435. Write five more blocks at once, ensure we're where we need to be.

84436. Only works in linux and WIN platforms

84437. wait for all jobs to finish

84438. Test that the two ways to get children file attributes (in bulk or individually) behave consistently.

84439. Unblock background thread

84440. Generate random data

84441. Random Read

84442. Write to mirror string

84443. Increase to 2 threads. Task 0, 1 running; 2 waiting

84444. Pick random offset for read

84445. Wake up threads 7. Task 5 running

84446. Schedule another task

84447. make sure we don't have more than pool_size_jobs running.

84448. Shrink back to 1 thread. Still task 5, 6, 7 running

84449. Check whether a bunch of concurrently existing files have unique IDs.

84450. Check that after file is deleted we don't get same ID again in a new file.

84451. Pick random offset for write

84452. Block the low priority queue

84453. Wait a short while for the jobs to be dispatched.

84454. allocate 100 MB
84455. Get Unique ID
84456. Reopen the file every 500 iters
84457. The filesystem containing the file does not support fallocate
84458. Verify the above
84459. Check IDs are the same.
84460. "[DEBUG]"
84461. Get Unique ID again
84462. schedule same number of jobs in each pool
84463. Wake up task 6. Task 5, 7 running
84464. Close file and reopen it
84465. close the file, should deallocate the blocks
84466. Increase to 4 threads. Task 5, 8, 9 running.
84467. Collect and check whether the IDs are unique.
84468. The file now has 1 sector worth of a followed by a page worth of b
84469. **comment:** clean up
 label: code-design
84470. Sequential Read
84471. Sync + corrupt => no change
84472. Corrupted
84473. Add new data and corrupt it
84474. Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file.
 See the AUTHORS file for names of contributors.
84475. this will be true unless test runs for 2 seconds
84476. namespace rocksdb
84477. * Class: org_rocksdb_RocksDBExceptionTest * Method: raiseException * Signature: ()V
84478. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84479. * Class: org_rocksdb_RocksDBExceptionTest * Method: raiseExceptionWithStatusCodeSubCode * Signature: ()V
84480. * Class: org_rocksdb_RocksDBExceptionTest * Method: raiseExceptionWithStatusCodeState * Signature: ()V
84481. * Class: org_rocksdb_RocksDBExceptionTest * Method: raiseExceptionWithStatusCode * Signature: ()V
84482. * Class: org_rocksdb_RocksDBExceptionTest * Method: raiseExceptionNoMsgWithStatusCodeSubCode * Signature: ()V
84483. * Class: org_rocksdb_RocksDBExceptionTest * Method: raiseExceptionNoMsgWithStatusCode * Signature: ()V
84484. column_family_id
84485. * Class: org_rocksdb_WriteBatchTestInternalHelper * Method: setSequence * Signature: (JJ)V
84486. **comment:** todo: Currently the following code is directly copied from db/write_bench_test.cc. It could be implemented in java once all the necessary components can be accessed via jni api.
 label: code-design
84487. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). This file implements the "bridge" between Java and C++ and enables calling c++ rocksdb::WriteBatch methods testing from Java side.
84488. * Class: org_rocksdb_WriteBatchTest * Method: getContents * Signature: (J)[B
84489. exception thrown: ArrayIndexOutOfBoundsException
84490. exception thrown: OutOfMemoryError
84491. * Class: org_rocksdb_WriteBatchTestInternalHelper * Method: append * Signature: (JJ)V
84492. * Class: org_rocksdb_WriteBatchTestInternalHelper * Method: sequence * Signature: (J)V
84493. re-open db and read from start to end integer keys should be in ascending order as defined by SimpleIntComparator
84494. ** Abstract tests for both Comparator and DirectComparator
84495. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84496. ** Get a comparator which will expect Integer keys * and determine an ascending order ** @return An integer ascending order key comparator
84497. ** Compares integer keys * so that they are in ascending order ** @param a 4-bytes representing an integer key * @param b 4-bytes representing an integer key
 ** @return negative if a < b; 0 if a == b, positive otherwise
84498. does key already exist (avoid duplicates) generate a different key
84499. ** Test which stores random keys into a column family * in the database * using an @see getAscendingIntKeyComparator * it then checks that these keys are read back in * ascending order ** @param db_path A path where we can store database * files temporarily * * @throws java.io.IOException if IO error happens.
84500. store 10,000 random integer keys
84501. protect against int key calculation overflow
84502. does key already exist (avoid duplicates)
84503. generate a different key
84504. ** Test which stores random keys into the database * using an @see getAscendingIntKeyComparator * it then checks that these keys are read back in * ascending order ** @param db_path A path where we can store database * files temporarily * * @throws java.io.IOException if IO error happens.
84505. Verify that backups exist
84506. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84507. restore the backup
84508. restore db from latest backup
84509. Fill database with some test values
84510. Create four backups
84511. close the database
84512. Values must have suffix V2 because of restoring latest backup.
84513. Create two backups
84514. The latest backup must remain.
84515. ** Verify backups. ** @param be {@link BackupEngine} instance. * @param expectedNumberOfBackups numerical value * @throws RocksDBException
 thrown if an error occurs within the native * part of the library.
84516. Open empty database.
84517. restore db from first backup
84518. Open database again.
84519. The second backup must remain.
84520. ** Fill database with some test values. ** @param db {@link RocksDB} instance. * @throws RocksDBException thrown if an error occurs within the native * part of the library.
84521. Delete everything except the latest backup
84522. Delete the first backup
84523. negative will be mapped to 0
84524. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84525. no-op

84526. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84527. no op

84528. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84529. Size has to be positive

84530. setup sample properties

84531. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84532. iterate over new_cf key/value pairs

84533. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84534. found value which fits in outValue

84535. found value which fits partially

84536. check if key is merged

84537. Test listColumnFamilies

84538. Test open database with column family names

84539. not found value

84540. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84541. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84542. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84543. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84544. UseAdaptiveMutex test

84545. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84546. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84547. Will be abc3 because the next one after abc999 is abc3

84548. Seek for abc

84549. Will be invalid because abc is after abc1

84550. test the round-tripability of keys written and read with the Comparator

84551. Iterate over keys using a iterator

84552. Get last one

84553. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84554. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84555. setup sample properties

84556. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84557. no-op

84558. Test with parameter initialization

84559. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84560. test the round-tripability of keys written and read with the DirectComparator

84561. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84562. clear first slice

84563. get first char in otherslice

84564. remove prefix

84565. make sure we don't double-free

84566. no-op

84567. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84568. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84569. new Bloom filter

84570. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84571. Identify the last line of the header

84572. As InfoLogLevel is set to FATAL_LEVEL, here we expect the log content to be empty.

84573. ** Read LOG file contents into String. ** @return LOG file contents as String. * @throws IOException if file is not found.

84574. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84575. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84576. KeyMayExist in CF1 must return false

84577. Test with column family

84578. Test without column family

84579. Test without column family but with readOptions

84580. Test with column family and readOptions

84581. no op

84582. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84583. Set custom logger to options

84584. messages shall be received due to previous actions.

84585. change log level to debug level

84586. Create new logger with max log level passed by options

84587. **comment:** there should be more than zero received log messages in debug level.
label: code-design

84588. there should be zero messages using fatal level as log level.

84589. **comment:** there should be zero messages using warn level as log level.
label: code-design

84590. Test HashSkipListMemTableConfig

84591. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84592. writing aa under key

84593. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84594. Test also with createColumnFamily

84595. test replace one with another merge operator instance

84596. merge bb under key

84597. writing xx under cfkey2

84598. test param init

84599. merge yy under cfkey2

84600. **comment:** test reuse
 label: code-design

84601. Writing aa under key

84602. Writing bb under key

84603. no-op

84604. Test Optimize for statements

84605. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84606. Set a table factory and check the names

84607. Free instances

84608. Initialize a dbOptions object from cf options and db options

84609. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84610. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84611. Size has to be positive

84612. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84613. Test with parameter initialization

84614. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84615. * * Determine if OS is 32-Bit/64-Bit * * @return boolean value indicating if operating system is 64 Bit.

84616. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84617. * * Helper class to get the appropriate Random class instance dependent * on the current platform architecture (32bit vs 64bit)

84618. * * Factory to get a platform specific Random instance * * @return {@link java.util.Random} instance

84619. * * Random32Bit is a class which overrides {@code nextLong} to * provide random numbers which fit in size_t. This workaround * is necessary because there is no unsigned_int < Java 8

84620. * * Utility class constructor

84621. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84622. test that put fails in readonly mode

84623. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84624. no-op

84625. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84626. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84627. open database

84628. fill database with key/value pairs

84629. found value which fits in outValue

84630. Because we only compacted those keys we issued in this round, there shouldn't be any L1 -> L2 compaction. So we expect zero L2 files here.

84631. add non existing key

84632. test same call with readOptions

84633. test same method with ReadOptions

84634. Making sure there isn't any L0 files.

84635. not found value

84636. Making sure there are some L1 files. Here we only use != 0 instead of a specific number as we don't want the test make any assumption on how compaction works.

84637. Disable auto L0 -> L1 compaction

84638. Make sure we do create one more L0 files.

84639. merge key1 with another value portion

84640. Compact all L0 files we just created

84641. To disable auto compaction

84642. merge on non existent key shall insert the value

84643. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84644. found value which fits partially

84645. remove existing key

84646. a slightly bigger write buffer than L0 file so that we can ensure manual flush always go before background flush happens.

84647. default rocksenv will always return zero for compaction pool no matter what was set via setBackgroundThreads

84648. default rocksenv will always return zero for flush pool no matter what was set via setBackgroundThreads

84649. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84650. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84651. verify otherDb

84652. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84653. read key/value pairs after flush using MemEnv

84654. shall throw an exception because db dir does not exist.

84655. write key/value pairs using MemEnv to db and to otherDb.

84656. db - keys

84657. Check iterator access

84658. write key/value pairs using MemEnv

84659. values

84660. After reopen the values shall still be in the mem env. as long as the env is not freed.

84661. read key/value pairs using MemEnv
84662. verify key/value pairs after flush using MemEnv
84663. reached end of database
84664. flush
84665. otherDb - keys
84666. verify db
84667. ** Resource to trigger garbage collection after each test * run. ** @deprecated Will be removed with the implementation of * {@link RocksObject#finalize()}
84668. make sure we don't double-free
84669. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84670. setting null to snapshot in ReadOptions leads to no Snapshot being used.
84671. read with previously created snapshot will read previous version of key value pair
84672. results must be the same with new Snapshot instance using the same native pointer
84673. iterate using a snapshot
84674. snapshot was created before newkey
84675. retrieve key value pair created before the snapshot was made
84676. set snapshot in ReadOptions
84677. iterate over current state of db
84678. Get new Snapshot of database set snapshot in ReadOptions
84679. add new key/value pair
84680. iterate using a snapshot on default column family
84681. Retrieve snapshot from read options
84682. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84683. release Snapshot
84684. iterate over current state of column family
84685. using no snapshot the latest db entries will be taken into account
84686. update key value pair to newvalue
84687. retrieve key value pair
84688. Get new Snapshot of database
84689. read for newkey using the snapshot must be null
84690. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84691. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84692. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84693. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84694. insert 5 writes into a cf
84695. the latest sequence number is 5 because 5 puts were written beforehand
84696. Get updates since the beginning
84697. the latest sequence number is 10 because (5 + 5) puts were written beforehand
84698. The first sequence number is 1
84699. reopen
84700. no-op
84701. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84702. ** Convert an int to 4 bytes ** @param v The int ** @return A byte array containing 4 bytes
84703. ** Simple type conversion methods * for use in tests
84704. ** Convert first 4 bytes of a byte array to an int ** @param data The byte array ** @return An integer
84705. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84706. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84707. ** Returns a copy of the current events list ** @return a list of the events which have happened upto now
84708. compare the results to the test data
84709. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84710. ** A simple WriteBatch Handler which adds a record * of each event that it receives to a list
84711. attempt to read test data back from the WriteBatch by iterating with a handler
84712. setup test data
84713. load test data to the write batch
84714. ** Enumeration of Write Batch * event actions
84715. without previous corresponding setSavePoint
84716. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84717. ** This class mimics the db/write_batch_test.cc * in the c++ rocksdb library. * <p/> * Not ported yet: * <p/> * Continue(); * PutGatherSlices();
84718. ** Package-private class which provides java api to access * c++ WriteBatchInternal.
84719. submit the callables
84720. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84721. put data to the write batch and make sure we can read it.
84722. add zero byte value
84723. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84724. update k2 in the write batch and check the value
84725. add put records
84726. remove k1 and make sure we can read back the write
84727. reverse iterative access
84728. add a deletion record
84729. forward iterative access
84730. reinsert k1 and make sure we see the new value
84731. without previous corresponding setSavePoint
84732. Arrays.equals(key, iterator.key()) ensures an exact match in Rocks, instead of a nearest match
84733. direct access - seek to key offsets

84734. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84735. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84736. * * Custom Junit Runner to print also Test classes * * and executed methods to command prompt.

84737. * * Listener which overrides default functionality * to print class and method to system out.

84738. * * Main method to execute tests * * @param args Test classes as String names

84739. * * RocksJUnitListener constructor * * @param system JUnitSystem

84740. * * Is `a` a prefix of `b` * * @return The length of the matching prefix, or 0 if it is not a prefix

84741. put

84742. * * Open the database using the Java ReverseBytewiseComparator * and test the results against another Java ReverseBytewiseComparator

84743. Seek to First

84744. Next

84745. Seek to random key

84746. Random walk and make sure iter and result_iter returns the same key and value

84747. Prev

84748. * * Open the database using the C++ BytewiseComparatorImpl * and test the results against our Java DirectBytewiseComparator

84749. * * Open the database using the C++ BytewiseComparatorImpl * and test the results against our Java BytewiseComparator

84750. * * Open the database using a C++ Comparator

84751. Seek to last

84752. note that calling value on a non-valid iterator from the Java API results in a SIGSEGV

84753. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84754. delete

84755. * * Open the database using the Java BytewiseComparator * and test the results against another Java BytewiseComparator

84756. * * Open the database using a Java Comparator

84757. * * Open the database using the Java DirectBytewiseComparator * and test the results against another Java DirectBytewiseComparator

84758. * * Open the database using the C++ ReverseBytewiseComparatorImpl * and test the results against our Java ReverseBytewiseComparator

84759. * * This is a direct port of various C++ * tests from db/comparator_db_test.cc * and some code to adapt it to RocksJava

84760. UNIX

84761. Linux

84762. AIX

84763. Copyright (c) 2014, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84764. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

84765. Check keys exist.

84766. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

84767. Advance to next key in the valid key space

84768. Our test skip list stores 8-byte unsigned integers

84769. Seek to middle

84770. Backward iteration test

84771. Simple iterator tests

84772. namespace rocksdb

84773. **comment:** Simple test that does single-threaded testing of the ConcurrentTest scaffolding.
label: test

84774. Remember the initial committed state of the skiplist.

84775. Seek to end

84776. Compare against model iterator

84777. InlineSkipList is not protected by mu_. We just use a single writer thread to modify it.

84778. Note that generation 0 is never inserted, so it is ok if <*,0,*> is missing.

84779. Iterate over the list, make sure keys appears in order and no extra keys exist.

84780. REQUIRES: No concurrent calls to WriteStep or ConcurrentWriteStep

84781. Validate the list is well-formed.

84782. Seek to beginning

84783. REQUIRES: No concurrent calls for the same k

84784. Per-key generation

84785. Verify that everything in [pos,current) was not present in initial_state.

84786. Current state of the test

84787. We sometimes pass K to seek to the end of the skiplist

84788. **comment:** We want to make sure that with a single writer and multiple concurrent readers (with no synchronization other than when a reader's iterator is created), the reader always observes all the data that was present in the skip list when the iterator was constructor. Because insertions are happening concurrently, we may also observe new values that were inserted since the iterator was constructed, but we should never miss any values that were present at iterator construction time. We generate multi-part keys: <key,gen,hash> where: key is in range [0..K-1] gen is a generation number for key hash is hash(key,gen) The insertion code picks a random key, sets gen to be 1 + the last generation number inserted for that key, and sets hash to Hash(key,gen). At the beginning of a read, we snapshot the last inserted generation number for each key. We then iterate, including random calls to Next() and Seek(). For every key we encounter, we check that it is either expected given the initial snapshot or has been concurrently added since the iterator started.
label: code-design

84789. Forward iteration test

84790. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

84791. SkipList is not protected by mu_. We just use a single writer thread to modify it.

84792. Advance to next key in the valid key space

84793. Seek to middle

84794. Backward iteration test

84795. Simple iterator tests

84796. namespace rocksdb

84797. **comment:** Simple test that does single-threaded testing of the ConcurrentTest scaffolding.
label: test

84798. Remember the initial committed state of the skiplist.

84799. Seek to end

84800. Compare against model iterator

84801. Note that generation 0 is never inserted, so it is ok if <*,0,*> is missing.

84802. Seek to beginning

84803. Per-key generation

84804. REQUIRES: External synchronization
84805. Verify that everything in [pos,current) was not present in initial_state.
84806. Current state of the test
84807. We sometimes pass K to seek to the end of the skip list
84808. **comment:** We want to make sure that with a single writer and multiple concurrent readers (with no synchronization other than when a reader's iterator is created), the reader always observes all the data that was present in the skip list when the iterator was constructed. Because insertions are happening concurrently, we may also observe new values that were inserted since the iterator was constructed, but we should never miss any values that were present at iterator construction time. We generate multi-part keys: <key,gen,hash> where: key is in range [0..K-1] gen is a generation number for key hash is hash(key,gen) The insertion code picks a random key, sets gen to be 1 + the last generation number inserted for that key, and sets hash to Hash(key,gen). At the beginning of a read, we snapshot the last inserted generation number for each key. We then iterate, including random calls to Next() and Seek(). For every key we encounter, we check that it is either expected given the initial snapshot or has been concurrently added since the iterator started.
label: code-design
84809. Forward iteration test
84810. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84811. Allocate another 2MB
84812. 90% of the hard limit will hit the condition
84813. A write buffer manager of size 256MB
84814. 1GB cache
84815. Still need flush as the hard limit hits
84816. Allocate another 20MB
84817. Free 2MB will not cause any change in cache cost
84818. ROCKSDB_LITE namespace rocksdb
84819. 15 MB total, 8MB mutable.
84820. A write buffer manager of size 50MB
84821. Allocate another 30MB
84822. Allocate 1.5MB will allocate 2MB
84823. Every free will release 1MB if still not hit 3/4
84824. Scheduling for freeing will release the condition
84825. Destroy write buffer manager should free everything
84826. 9MB total, 8MB mutable.
84827. Free 20MB will release 1MB from cache
84828. A write buffer manager of size 10MB
84829. 11MB total, 6MB mutable. hard limit still hit
84830. 11MB total, 4MB mutable. hard limit stills but won't flush because more than half data is already being flushed.
84831. $99 * 250 / 100$ $95 * 250 / 100$ $50 * 250 / 100$ $(1 + 250) / 2$
84832. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84833. dropping oldest window with value 2, remaining 3 ~ 5
84834. $99 * 110 / 100$ $95 * 110 / 100$ $50 * 110 / 100$ $(1 + 110) / 2$
84835. dropping oldest window with value 1, remaining 2 ~ 4
84836. namespace rocksdb
84837. fill up to bucket [70, 110)
84838. namespace rocksdb
84839. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84840. namespace rocksdb
84841. Sanity check to make sure that contents and order of TickersNameMap match Tickers enum
84842. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
84843. If the test fails, likely a new option is added to DBOptions but it cannot be set through GetDBOptionsFromString(), or the test is not updated accordingly. After adding an option, we need to make sure it is settable by GetDBOptionsFromString() and add the option to the input string passed to DBOptionsFromString() in this test. If it is a complicated type, you also need to add the field to kDBOptionsBlacklist, and maybe add customized verification for it.
84844. namespace rocksdb
84845. Items in the form of <offset, size>. Need to be in ascending order and not overlapping. Need to updated if new pointer-option is added.
84846. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84847. GFLAGS
84848. Need to update the option string if a new option is added.
84849. This option is not setable:
84850. **comment:** It based on the behavior of compiler that padding bytes are not changed when copying the struct. It's prone to failure when compiler behavior changes. We verify there is unset bytes to detect the case.
label: code-design
84851. Following options are not settable through GetColumnFamilyOptionsFromString():
84852. !__clang__ OS_LINUX || OS_WIN !ROCKSDB_LITE
84853. In this test, we catch a new option of BlockBasedTableOptions that is not settable through GetBlockBasedTableOptionsFromString(). We count padding bytes of the option struct, and assert it to be the same as unset bytes of an option struct initialized by GetBlockBasedTableOptionsFromString().
84854. Construct the base option passed into GetBlockBasedTableOptionsFromString().
84855. If the test fails, likely a new option is added to BlockBasedTableOptions but it cannot be set through GetBlockBasedTableOptionsFromString(), or the test is not updated accordingly. After adding an option, we need to make sure it is settable by GetBlockBasedTableOptionsFromString() and add the option to the input string passed to the GetBlockBasedTableOptionsFromString() in this test. If it is a complicated type, you also need to add the field to kBbtoBlacklist, and maybe add customized verification for it.
84856. Verify options are settable from options strings. We take the approach that depends on compiler behavior that copy constructor won't touch implicit padding bytes, so that the test is fragile. As a result, we only run the tests to verify new fields in options are settable through string on limited platforms as it depends on behavior of compilers.
84857. options in the blacklist need to appear in the same order as in ColumnFamilyOptions.
84858. If the test fails, likely a new option is added to ColumnFamilyOptions but it cannot be set through GetColumnFamilyOptionsFromString(), or the test is not updated accordingly. After adding an option, we need to make sure it is settable by GetColumnFamilyOptionsFromString() and add the option to the input string passed to GetColumnFamilyOptionsFromString() in this test. If it is a complicated type, you also need to add the field to kColumnFamilyOptionsBlacklist, and maybe add customized verification for it.
84859. Count padding bytes by setting all bytes in the memory to a special char, copy a well constructed struct to this memory and see how many special bytes left.
84860. Deprecatd option which is not initialized. Need to set it to avoid Valgrind error
84861. persist the change of prefix_extractor
84862. table_factory
84863. only digits and dots are allowed
84864. Missing option name

84865. Phase 1: randomly assign base_opt custom type options
84866. Invalid chars after closing curly brace
84867. change the compaction filter
84868. GetColumnFamilyOptionsFromString is not supported in ROCKSDB_LITE
84869. Random Construct a string
84870. Further verify pointer-typed options
84871. Wrong name "max_write_buffer_number_"
84872. default for VerifyDBOptions
84873. compaction_filter_factory
84874. Empty key
84875. unrecognized filter policy name
84876. unrecognized checksum type
84877. Phase 3: Set new_options from the derived string and expect new_options == base_options
84878. new_opt_mat
84879. Error Paring value
84880. merge_operator
84881. change the compaction filter factory
84882. Multi-level nested options
84883. ROCKSDB_LITE
84884. use same prefix extractor but with different parameter
84885. Phase 2: obtain a string from base_opt
84886. change the merge operator
84887. Units (m)
84888. Nested plain table options Empty
84889. unrecognized index type
84890. Empty value
84891. change the name of merge operator back-and-forth
84892. if true, '\' together with str[i + 1] is not a valid escape.
84893. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84894. However this is valid!
84895. This two transformations should be canceled and should output the original input.
84896. Value with '='
84897. GetPlainTableOptionsFromString is not supported
84898. Units (k)
84899. Phase 3: Set new_opt from the derived string and expect new_opt == base_opt
84900. Overwrriten option
84901. namespace
84902. input_strings_escaped
84903. Regular options
84904. !ROCKSDB_LITE namespace rocksdb
84905. Phase 2: obtain a string from base_option
84906. set compaction_filter to nullptr and expect non-ok status
84907. Make sure block-based table factory options was deserialized correctly
84908. if there're two consecutive '\', skip the second one.
84909. must have at least one digit before each dot must have at least one digit after each dot
84910. set the value back and expect ok status
84911. Units (g)
84912. from CheckOptionsCompatibility
84913. Invalid block based table option
84914. change the name of the compaction filter factory back-and-forth
84915. new_opt_map
84916. Add a random character
84917. Change prefix extractor from non-nullptr to nullptr
84918. Wrong key/value pair
84919. can only contains at most one dot. options_file_version must be at least one
84920. GetOptionsFromMap is not supported in ROCKSDB_LITE
84921. GetBlockBasedTableOptionsFromString is not supported
84922. don't overwrite block based table options
84923. !ROCKSDB_LITE
84924. Last one
84925. change the name and expect non-ok status
84926. Phase 1: Make big change in base_options
84927. change the name and expect non-ok status
84928. Non-empty
84929. make sure default values are overwritten by something else
84930. change the name back and expect ok status
84931. Okay to change prefix_extractor form nullptr to non-nullptr
84932. default for VerifyCFOptions
84933. memtable factory
84934. Empty nested options
84935. unrecognized filter policy config
84936. change the table factory
84937. expect pass only in kSanityLevelLooselyCompatible
84938. expect pass as it's safe to change prefix_extractor from non-null to null
84939. With random spaces
84940. prefix_extractor
84941. GFLAGS
84942. persist the change
84943. StringToMap is not supported in ROCKSDB_LITE
84944. **comment:** repeat the test with FixedPrefixTransform
label: test
84945. set table_factory to nullptr and expect non-ok status
84946. default ColumnFamilyOptions
84947. test by setting memtable_factory to nullptr
84948. test by setting table_factory to nullptr
84949. compaction_filter

84950. ignore_unknown_options
84951. Unexpected chars after closing curly brace
84952. GetMemTableRepFactoryFromString is not supported
84953. **comment:** Mismatch curly braces
 label: code-design
84954. since we already handle those two consecutive '\s in the next if-then branch, any '\' appear at the end of an escaped string in such case is not valid.
84955. Units (t)
84956. test by setting compaction_filter to nullptr
84957. from CheckOptionsCompatibility
84958. Nested block based table options Empty
84959. GetOptionsFromString is not supported in RocksDB Lite
84960. Regular nested options
84961. set memtable_factory to nullptr and expect non-ok status
84962. unknow options should fail parsing without ignore_unknown_options = true
84963. Make sure segfault is not hit by semi-random strings
84964. unknown option
84965. unrecognized EncodingType
84966. **comment:** Garbage inside curly braces
 label: code-design
84967. Replace random position to space
84968. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under the BSD-style license found in the LICENSE file in the root directory of this source tree. An additional grant of patent rights can be found in the PATENTS file in the same directory. Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84969. Global operators to be replaced by a linker when this file is a part of the build
84970. Second filter
84971. Test for block based filter block use new interface in FilterPolicy to create filter builder/reader
84972. For testing: emit an array with one hash value per key
84973. Last filter
84974. Check third filter (empty)
84975. Check first filter
84976. Check last filter
84977. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2012 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
84978. namespace rocksdb
84979. Third filter is empty
84980. Check second filter
84981. First filter
84982. Return true if any byte in this range was Marked
84983. Generate random key value pairs. The generated key will be sorted. You can tune the parameters to generated different kinds of test key/value pairs for different scenario.
84984. read contents of block sequentially
84985. read block contents randomly
84986. generate different prefix
84987. create block reader
84988. In this test case, no two key share same prefix.
84989. search in block for this key
84990. find a random key in the lookaside array
84991. namespace rocksdb
84992. compare with lookaside array
84993. Seek existent keys
84994. A slow and accurate version of BlockReadAmpBitmap that simply store all the marked ranges in a set.
84995. last key id
84996. step
84997. Generate keys with same prefix. first key id last key id step padding size,
84998. Add only half of the keys
84999. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85000. Error in read amplification will be less than 2% if we are reading randomly
85001. restart interval
85002. Error in read amplification will be less than 1% if we are reading sequentially
85003. generating keys that shares the prefix
85004. read serialized contents of the block
85005. read kv from block
85006. Read the block sequentially using Seek()
85007. first key id
85008. Seek non-existent keys. For hash index, if no key with a given prefix is not found, iterator will simply be set as invalid; whereas the binary search based iterator will return the one that is closest.
85009. padding size (8 bytes randomly generated suffix)
85010. add a bunch of records to a block
85011. 1 byte 32 bytes 61 bytes 64 bytes 0.5 KB 1 KB 4 KB 10 KB 50 KB 1 MB 4 MB 10 MB
85012. Read the block randomly
85013. for each prefix, there will be 5 keys starts with it.
85014. return the block contents
85015. block test
85016. Generate some random entries
85017. 100 bytes values
85018. Read the block sequentially using Next()
85019. res = 5;
85020. namespace rocksdb
85021. res = 2; res = 2 * 3;
85022. res = 5; res = 5 * 7;
85023. **comment:** the first Cleanup is on stack and the rest on heap, so test all the combinations of them
 label: code-design
85024. **comment:** Putting the PinnableSlice tests here due to similarity to Cleanable tests
 label: test
85025. ~Cleanable

85026. res = 5 * 7;
85027. **comment:** Test the Reset does cleanup
 label: code-design
85028. res = 2;
85029. res = 2; res = 2 * 5 * 7;
85030. **comment:** the first Cleanup is on stack and the rest on heap, so test with both cases
 label: code-design
85031. res = 2 * 3;
85032. Test Clenable is usable after Reset
85033. res = 2; res = 2 * 3; res = 2 * 3 * 5 * 7;
85034. res = 2; res = 2 * 3; res = 2 * 3 * 5;
85035. **comment:** Use this to keep track of the cleanups that were actually performed
 label: code-design
85036. **comment:** ~Cleanable cleanups must have be delegated to value
 label: code-design
85037. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85038. column_family_id
85039. i is not one of the expected locations. Empty bucket.
85040. Assert Table Properties.
85041. Have two hash functions. Insert elements with overlapping hashes. Finally insert an element with hash value somewhere in the middle so that it displaces all the elements after that.
85042. **comment:** Need to have a temporary variable here as VS compiler does not currently support operator= with initializer_list as a parameter
 label: code-design
85043. ROCKSDB_LITE
85044. **comment:** Check unused bucket.
 label: code-design
85045. Check that all keys wereReader found.
85046. Check contents of the bucket.
85047. Have two hash functions. Insert elements with overlapping hashes. Finally try inserting an element with hash value somewhere in the middle and it should fail because the no. of elements to displace is too high.
85048. namespace
85049. namespace rocksdb
85050. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85051. Read file
85052. Give disjoint hash values.
85053. namespace.
85054. Give disjoint hash values, in reverse order.
85055. Methods, variables related to Hash functions.
85056. **comment:** Test read when key is unused key.
 label: code-design
85057. Last level file.
85058. Check reader now.
85059. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85060. Create last level file as we are interested in measuring performance of last level file only.
85061. namespace rocksdb
85062. GFLAGS.
85063. column_family_id
85064. Search for a key with colliding hash values.
85065. Value is just a part of key.
85066. Assume only the fast path is triggered
85067. namespace
85068. Assume no merge/deletion
85069. Search for a key with an independent hash value.
85070. ROCKSDB_LITE
85071. Test with collision. Make all hash values collide.
85072. Add hash values that map to empty buckets.
85073. These numbers are chosen to have a hash utilization % close to 0.9, 0.75, 0.6 and 0.5 respectively. They all create 128 M buckets.
85074. Add keys with colliding hash values.
85075. Make all hash values collide.
85076. Performance tests
85077. Remain same symantic with blockbased filter
85078. Generate the filter using the keys that are added
85079. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85080. namespace rocksdb
85081. Add Key to filter
85082. namespace rocksdb
85083. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85084. 1 is 2nd level index
85085. assuming a good hash function
85086. **comment:** unused
 label: code-design
85087. A low number ensures cutting a block after each key
85088. querying missing keys
85089. querying a key twice
85090. Querying added keys
85091. Initialize what Open normally does as much as necessary for the test
85092. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
85093. last two keys make one flush
85094. namespace rocksdb
85095. Otherwise BlockBasedTable::Close will access variable that are not initialized in our mocked version
85096. Assuming a block is cut, add an entry to the index
85097. handle

85098. A regression test to avoid data race described in <https://github.com/facebook/rocksdb/issues/1267>
85099. Use shorter block size for tests to exercise block boundary conditions more.
85100. reset the cache and reopen the table
85101. the key is greater than any existing keys.
85102. Only data block will be accessed
85103. -- Find keys do not exist, but have common prefix.
85104. rerun with different block cache
85105. Close the table
85106. Open the block
85107. small block size to get big index block
85108. Enable the cache for index/filter blocks
85109. 8 is seq size, 9 k-v totally
85110. keys_not_in_cache=
85111. invalid
85112. Get non-existing key
85113. **comment:** Check if the fetched props matches the expected ones. TODO(kailiu) Use this only when you disabled filter policy!
 label: code-design
85114. Cache hit, bytes read from cache should increase
85115. global_seqno
85116. convert_to_internal_key_
85117. Make each key/value an individual block
85118. Helper function to get version, global_seqno, global_seqno_offset
85119. ROCKSDB_LITE
85120. Binary search index
85121. preloading filter/index blocks is prohibited.
85122. the key/val are slightly smaller than block size, so that each block holds roughly one key/value pair.
85123. The purpose of this test is to test the prefetching operation built into BlockBasedTable.
85124. prefixes that don't exist
85125. Tests against all kinds of tables
85126. Verify Seek
85127. Create a bunch of keys with 10 filters.
85128. no filter policy is used
85129. version
85130. NOTE: to help better highlight the "delta" of each ticker, I use <last_value> + <added_value> to indicate the increment of changed value; other numbers remain the same. index block hit
85131. find the upper bound of prefixes
85132. assert our expectation in cache warmup
85133. **comment:** a hack that just to trigger BlockBasedTable::GetFilter.
 label: code-design
85134. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
85135. seqno=4 is less than 3 so we still should get our key
85136. Default comparator
85137. Test iterating
85138. Return reverse of "key". Used to test non-lexicographic comparators.
85139. column_family_id
85140. ASSERT_OK(regular_iter->status());
85141. Each time we load one more key to the table. the table index block size is expected to be larger than last time's.
85142. Test point lookup only one kv
85143. This test serves as the living tutorial for the prefix scan of user collected properties.
85144. with block-based, we read index and then the filter
85145. namespace
85146. data block hit
85147. Both index and data block get accessed. It first cache index block then data block. But since the cache size is only 1, index block will be purged after data block is inserted.
85148. A helper class that converts internal format keys into user keys
85149. Generate table without filter policy
85150. Test the empty key
85151. -- PART 1: Open with regular block cache. Since block_cache is disabled, no cache activities will be involved.
85152. We get the following data spread : Data block Index ===== [k01 k02 k03] k03 [k04] k04 [k05] k05 [k06 k07] k07
85153. Update global sequence number to 3
85154. No merge operator
85155. In order to make all tests run for plain table format, including those operating on empty keys, create a new prefix transformer which return fixed prefix if the slice is not shorter than the prefix length, and the full slice if it is shorter.
85156. No property collectors
85157. if DB is opened with a prefix extractor of a different name, prefix bloom is skipped when read the file
85158. release the iterator so that the block cache can reset correctly.
85159. upconvert legacy block based
85160. A simple PrefixExtractor that only works for test PrefixAndWholeKeyTest
85161. No filter policy is used
85162. Create a table
85163. index block miss
85164. Generate table with filter policy
85165. Data block will be in cache
85166. null keys
85167. nothing happens in the beginning
85168. find keys with prefix that don't match any of the existing prefixes.
85169. Block cache can contain raw data blocks as well as general objects. If an object depends on the table to be live, it then must be destructed before the table is closed.
 This test makes sure that the only items remains in the cache after the table is closed are raw data blocks.
85170. namespace rocksdb
85171. Construct the data structure from the data in "data"
85172. DummyPropertiesCollector used to test BlockBasedTableProperties
85173. version == 2
85174. Helper function to get the contents of the table InternalIterator
85175. prefetch
85176. data, index and filter block
85177. bloom_filter_type = 0 -- block-based filter bloom_filter_type = 0 -- full filter

85178. invalid values for block_restart_interval (<1) are silently set to 1
85179. key_range=
85180. Hash search index with hash_index_allow_collision
85181. find existing keys
85182. At first, no block will be accessed.
85183. This test include all the basic checks except those for index size and block size, which will be conducted in separated unit tests.
85184. preloading filter/index blocks is enabled.
85185. !ROCKSDB_LITE
85186. Plain table is not supported in ROCKSDB_LITE
85187. filter is already in memory and it figures out that the key doesn't exist
85188. k04 and k05 will be in two consecutive blocks, the index is an arbitrary slice between k04 and k05, either before or after k04
85189. Trigger compaction.
85190. In the second round, turn whole_key_filtering off and expect rocksdb still works.
85191. seek the first element in the block
85192. xxhash block based
85193. prefixes that exist
85194. ASSERT_TRUE(regular_iter->Valid());
85195. A wrapper around LRICache that also keeps track of data blocks (in contrast with the objects) in the cache. The class is very simple and can be used only for trivial tests.
85196. invalid values for block_size_deviation (<0 or >100) are silently set to 0
85197. Check that when we reopen a table we don't lose access to blocks already in the cache. This test checks whether the Table actually makes use of the unique ID from the file.
85198. big enough so we don't ever lose cached values.
85199. Hash search index with filter policy
85200. Due to the difficulties of the intersection between statistics, this test only tests the case when "index block is put to block cache"
85201. Cache miss, Bytes read from cache should not change
85202. Release pinnable slice resources
85203. just the data block
85204. Something small to force merging
85205. Hash search index
85206. keys_in_cache=
85207. Returns nullptr if not running against a DB
85208. Verify data size.
85209. PlainTable expects internal key structure
85210. Replace the deleter with our own so that we keep track of data blocks erased from the cache
85211. last key
85212. For second iteration, delete the table reader object and verify the iterator can still access its metablock's range tombstones.
85213. data block miss
85214. odd
85215. index will be added to block cache. index block miss
85216. If the item was marked for being data block, decrease its usage from the total data block usage of the cache
85217. Return something larger than an existing key
85218. Overridden in DBConstructor
85219. Make sure, by default, index/filter blocks were pre-loaded (meaning we won't use block cache to store them).
85220. if (!BZip2_Supported()) { fprintf(stderr, "skipping bzip2 compression tests\n"); } else { compression_state.push_back(kBZip2Compression); }
85221. A simple tool that takes the snapshot of block cache statistics.
85222. Edge cases
85223. Attempt to return something smaller than an existing key
85224. This is called by the application right after inserting a data block
85225. -- PART 2: Open with very small block cache In this test, no block will ever get hit since the block cache is too small to fit even one entry.
85226. Test point lookup
85227. Finish constructing the data structure with all the keys that have been added so far. Returns the keys in sorted order in "*keys" and stores the key/value pairs in "**kvmap"
85228. The only usage must be for marked data blocks
85229. **comment:** we need to use random keys since the pure human readable texts may be well compressed, resulting in significant change of index block size.
 label: code-design
85230. with full-filter, we read filter first and then we stop
85231. No prefix extractor
85232. **comment:** Wrap around to invalid value
 label: code-design
85233. We must have created enough data to force merging
85234. find the lower bound of the prefix
85235. -- PART 3: Open table with bloom filter enabled but not in SST file
85236. BlockBasedTableTest::PrefetchTest
85237. upconvert legacy plain table
85238. Doing a read to make index/filter loaded into the cache
85239. Helper function to update the value of the global seqno in the file
85240. regular_iter->Seek(prefix);
85241. Open the table
85242. Plain table doesn't use restart index or compression.
85243. index block miss data block miss
85244. keys with prefix length 3, make sure the key/value is big enough to fill one block
85245. It's very hard to figure out the index block size of a block accurately. To make sure we get the index size, we just make sure as key number grows, the filter block size also grows.
85246. Compression type == that set:
85247. To tickle the PrefixMayMatch bug it is important that the user-key is a single byte so that the index key exactly matches the user-key.
85248. Only index block will be accessed
85249. Then call the original deleter
85250. index block hit
85251. Update global sequence number to 10
85252. Return an existing key
85253. Helper class for tests to unify the interface between BlockBuilder/TableBuilder and Block/Table.
85254. Only add compression if it is supported
85255. Open table with filter policy
85256. There must be some pinned data since PinnableSlice has not released them yet
85257. SeekToFirst() accesses data block. With similar reason, we expect data block's cache miss.
85258. -- Table construction
85259. **comment:** TODO(kailiu) DoCompressionTest() doesn't work with BZip2.

label: requirement

85260. Simple

85261. Seek to non-existing prefixes should yield either invalid, or a key with prefix greater than the target.

85262. Reads and returns the string environment variable corresponding to the given flag; if it's not set, returns default_value.

85263. Deletes the test object.

85264. iteration

85265. class Test

85266. Class UnitTestOptions. This class contains functions for processing options the user specifies when running the tests. It has only static members. In most cases, the user can specify an option using either an environment variable or a command line flag. E.g. you can set the test filter using either GTEST_FILTER or --gtest_filter. If both the variable and the flag are present, the latter overrides the former.

85267. **comment:** MSVC 8 deprecates _ftime64(), so we want to suppress warning 4996 (deprecated function) there. TODO(kenton@google.com): Use GetTickCount()? Or use SystemTimeToFileTime()

label: code-design

85268. **comment:** When compiled with MSVC 7.1 in optimized mode, destroying the UnitTest object upon exiting the program messes up the exit code, causing successful tests to appear failed. We have to use a different implementation in this case to bypass the compiler bug. This implementation makes the compiler happy, at the cost of leaking the UnitTest object.

label: code-design

85269. Compares two wide C strings, ignoring case. Returns true iff they have the same content. Unlike wcscasecmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NUL wide C string, including the empty string. NB: The implementations on different platforms slightly differ. On Windows, this method uses _wcsicmp which compares according to LC_CTYPE environment variable. On GNU platform this method uses wcscasecmp which compares according to LC_CTYPE category of the current locale. On Mac OS X, it uses towlower, which also uses LC_CTYPE category of the current locale.

85270. The following two functions only make sense if the system uses UTF-16 for wide string encoding. All supported systems with 16 bit wchar_t (Windows, Cygwin, Symbian OS) do use UTF-16.

85271. Allows user supplied key value pairs to be recorded for later output.

85272. **comment:** However, we want to clean up as much as possible. Hence we will always call TearDown(), even if SetUp() or the test body has failed.

label: code-design

85273. **comment:** Converting via modulus introduces a bit of downward bias, but it's simple, and a linear congruential generator isn't too good to begin with.

label: code-design

85274. The time of the test program start, in ms from the start of the UNIX epoch.

85275. Creates a client socket and connects to the server.

85276. **comment:** The default death test style.

label: test

85277. A working implementation of the OsStackTraceGetterInterface interface.

85278. When we fork the process below, the log file buffers are copied, but the file descriptors are shared. We flush all log files here so that closing the file descriptors in the child process doesn't throw off the synchronization between descriptors and buffers in the parent process. This is as close to the fork as possible to avoid a race condition in case there are multiple threads running before the death test, and another thread writes to the log file.

85279. Prints a ::string object.

85280. Clean up the ThreadLocalValues data structure while holding the lock, but defer the destruction of the ThreadLocalValueHolderBases.

85281. **comment:** Clears the results of all tests, except the ad hoc tests.

label: test

85282. **comment:** For more convenience, we print c's code again in hexadecimal, unless c was already printed in the form '\x##' or the code is in [1, 9].

label: code-design

85283. Continue only if the next hunk is very close.

85284. Our random number generator.

85285. GTEST_HAS_TYPED_TEST_P

85286. Formats a source file path and a line number as they would appear in an error message from the compiler used to compile this code.

85287. Formats the count of tests.

85288. Add a spacer if no FAILURE banner is displayed.

85289. Sets the default_result_printer attribute to the provided listener. The listener is also added to the listener list and previous default_result_printer is removed from it and deleted. The listener can also be NULL in which case it will not be added to the list. Does nothing if the previous and the current listener objects are the same.

85290. Sets the TestCase object for the test that's currently running.

85291. **comment:** TODO(vladl@google.com): Replace the following check with a compile-time assertion when available.

label: code-design

85292. Prints the given array of characters to the ostream. CharType must be either char or wchar_t. The array starts at begin, the length is len, it may include '\0' characters and may not be NUL-terminated.

85293. **comment:** We create the exception message on the heap because VC++ prohibits creation of objects with destructors on stack in functions using __try (see error C2712).

label: code-design

85294. Prints a string containing code-encoded text. The following escape sequences can be used in the string to control the text color: @@ prints a single '@' character. @R changes the color to red. @G changes the color to green. @Y changes the color to yellow. @D changes to the default terminal text color.

TODO(wan@google.com): Write tests for this once we add stdout capturing to Google Test.

85295. Don't strip off trailing separator if path is a root directory on Windows (like "C:\\").

85296. Prints a (const) wchar_t array of 'len' elements, starting at address 'begin'.

85297. Returns true iff Google Test should use colors in the output.

85298. **comment:** Helper class that holds the state for one hunk and prints it out to the stream. It reorders adds/removes when possible to group all removes before all adds. It also adds the hunk header before printing into the stream.

label: code-design

85299. Returns the message describing the last system error, regardless of the platform.

85300. Runs the test only if the test object was created and its constructor didn't generate a fatal failure.

85301. It is an error to pass this method a listener that is already in the list.

85302. namespace testing Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: mheule@google.com (Markus Heule) The Google C++ Testing Framework (Google Test)

85303. **comment:** How many times to repeat the tests? We don't want to repeat them when we are inside the subprocess of a death test.

label: test

85304. Constructs and returns the message for an equality assertion (e.g. ASSERT_EQ, EXPECT_STREQ, etc) failure. The first four parameters are the expressions used in the assertion and their values, as strings. For example, for ASSERT_EQ(foo, bar) where foo is 5 and bar is 6, we have: expected_expression: "foo" actual_expression: "bar" expected_value: "5" actual_value: "6" The ignoring_case parameter is true iff the assertion is a *_STRCASEEQ*. When it's true, the string "(ignoring case)" will be inserted into the message.

85305. **comment:** Assume other platforms have `gettimeofday()`. TODO(kenton@google.com): Use autoconf to detect availability of `gettimeofday()`.
label: requirement
85306. Information about a Google Test trace point.
85307. `GTEST_HAS_CLONE`
85308. Functions for processing the `gtest_filter` flag.
85309. Prints a (const) char array of 'len' elements, starting at address 'begin'.
85310. We are on Windows proper.
85311. Given the total number of shards, the shard index, and the test id, returns true iff the test should be run on this shard. The test id is some arbitrary but unique non-negative integer assigned to each test method. Assumes that $0 \leq \text{shard_index} < \text{total_shards}$.
85312. Ensures that this function does not execute more than once.
85313. Shuffles the death test cases.
85314. If `*pstr` starts with the given prefix, modifies `*pstr` to be right past the prefix and returns true; otherwise leaves `*pstr` unchanged and returns false. None of `pstr`, `*pstr`, and `prefix` can be NULL.
85315. Waits for the child process to exit if it haven't already. This returns immediately if the child has already exited, regardless of whether previous calls to `WaitForMultipleObjects` synchronized on this handle or not.
85316. This helper class can be used to mock out Google Test failure reporting so that we can test Google Test or code that builds on Google Test. An object of this class appends a `TestPartResult` object to the `TestPartResultArray` object given in the constructor whenever a Google Test failure is reported. It can either intercept only failures that are generated in the same thread that created this object or it can intercept all generated failures. The scope of this mock object can be controlled with the second argument to the two arguments constructor.
85317. Creates a Unicode code point from UTF16 surrogate pair.
85318. The vector of `TestCases` in their original order. It owns the elements in the vector.
85319. Destructor of `TestCase`.
85320. class `PrettyUnitTestResultPrinter`
85321. `max_depth`
85322. Gets the value of the flag as a string.
85323. **comment:** TODO(kenton@google.com): Shouldn't this just use `GetSystemTimeAsFileTime()`?
label: code-design
85324. Creates a new `TestInfo` object and registers it with Google Test; returns the created object. Arguments: `test_case_name`: name of the test case name; `name` of the test type `_param`: the name of the test's type parameter, or NULL if this is not a typed or a type-parameterized test. `value_param`: text representation of the test's value parameter, or NULL if this is not a value-parameterized test. `fixture_class_id`: ID of the test fixture class `set_up_tc`: pointer to the function that sets up the test case `tear_down_tc`: pointer to the function that tears down the test case factory: pointer to the factory that creates a test object. The newly created `TestInfo` instance will assume ownership of the factory object.
85325. Creates an ANSI string from the given wide string, allocating memory using `new`. The caller is responsible for deleting the return value using `delete[]`. Returns the ANSI string, or NULL if the input is NULL.
85326. The text used in failure messages to indicate the start of the stack trace.
85327. Checks if `str` contains '`-`', '`&`', '`%`' or '`\n`' characters. If yes, replaces them by "%xx" where xx is their hexadecimal value. For example, replaces "`=`" with "%3D". This algorithm is O(`strlen(str)`) in both time and space -- important as the input `str` may contain an arbitrarily long test failure message and stack trace.
85328. `getcwd` will likely fail in NaCl due to the sandbox, so return something reasonable. The user may have provided a shim implementation for `getcwd`, however, so fallback only when failure is detected.
85329. Returns true iff regex matches a prefix of `str`. `regex` must be a valid simple regular expression and not start with "`^`", or the result is undefined.
85330. We also need to decrement the iterator as we just removed an element.
85331. **comment:** Returns true iff regex matches any substring of `str`. `regex` must be a valid simple regular expression, or the result is undefined. The algorithm is recursive, but the recursion depth doesn't exceed the regex length, so we won't need to worry about running out of stack space normally. In rare cases the time complexity can be exponential with respect to the regex length + the string length, but usually it's must faster (often close to linear).
label: code-design
85332. The helper function for `{ASSERT|EXPECT}_EQ` with int or enum arguments.
85333. Use the `getaddrinfo()` to get a linked list of IP addresses for the given host name.
85334. Makes sure `full_pattern_` starts with '`\n`'.
85335. Makes sure `full_pattern_` ends with '`$`'.
85336. Has `strtol()` consumed all characters in the string?
85337. Returns true iff the test name of `test_info` matches `name_`.
85338. `WindowsDeathTest` uses an anonymous pipe to communicate results of a death test.
85339. Getters for the per-thread Google Test trace stack.
85340. In order to support thread-safe death tests, we need to remember the original working directory when the test program was first invoked. We cannot do this in `RUN_ALL_TESTS()`, as the user may have changed the current directory before calling `RUN_ALL_TESTS()`. Therefore we capture the current directory in `AddTestInfo()`, which is called to register a TEST or TEST_F before `main()` is reached.
85341. Clears the results of ad-hoc test assertions.
85342. Returns true iff `str` ends with the given suffix, ignoring case. Any string is considered to end with an empty suffix.
85343. Formats a countable noun. Depending on its quantity, either the singular form or the plural form is used. e.g. `FormatCountableNoun(1, "formula", "formuli")` returns "1 formula". `FormatCountableNoun(5, "book", "books")` returns "5 books".
85344. Mutex for linked pointers.
85345. Helper template for implementing `FloatLE()` and `DoubleLE()`.
85346. End `PrettyUnitTestResultPrinter`
85347. or if `val1` is almost equal to `val2`.
85348. **comment:** Returns an XML-escaped copy of the input string `str`. If `is_attribute` is true, the text is meant to appear as an attribute value, and normalizable whitespace is preserved by replacing it with character references. Invalid XML characters in `str`, if any, are stripped from the output. It is expected that most, if not all, of the text processed by this module will consist of ordinary English text. If this module is ever modified to produce version 1.1 XML output, most invalid characters can be retained using character references. TODO(wan): It might be nice to have a minimally invasive, human-readable escaping scheme for invalid characters, rather than dropping them.
label: code-design
85349. Gets the random seed used at the start of the current test iteration.
85350. class `UnitTest`
85351. Adds a `TestPartResult` to the current `TestResult` object. All Google Test assertion macros (e.g. `ASSERT_TRUE`, `EXPECT_EQ`, etc) eventually call this to report their results. The user code should use the assertion macros instead of calling this directly.
85352. `str` and `flag` must not be NULL.
85353. `ParseInternalRunDeathTestFlag()` has performed all the necessary processing.
85354. A concrete death test class that forks, then immediately runs the test in the child process.
85355. Symbian OpenC has `PATH_MAX` in `sys/syslimits.h`
85356. Returns true iff the test case failed.
85357. Parses the command line for Google Test flags, without initializing other parts of Google Test.
85358. **comment:** A struct that encompasses the arguments to the child process of a threadsafe-style death test process.
label: code-design
85359. Gets the number of all tests.
85360. `IsSubstring()` and `IsNotSubstring()` check whether `needle` is a substring of `haystack` (NULL is considered a substring of itself only), and return an appropriate error message when they fail.
85361. `!GTEST_OS_QNX`
85362. Google Test - The Google C++ Testing Framework This file implements a universal value printer that can print a value of any type T: void `::testing::internal::UniversalPrinter<T>::Print(value, ostream_ptr);` It uses the `<<` operator when possible, and prints the bytes in the object otherwise. A user can

override its behavior for a class type Foo by defining either operator<<(::std::ostream&, const Foo&) or void PrintTo(const Foo&, ::std::ostream*) in the namespace that defines Foo.

85363. Restores the current working directory.

85364. Converts a wide string to a narrow string in UTF-8 encoding. The wide string is assumed to have the following encoding: UTF-16 if sizeof(wchar_t) == 2 (on Windows, Cygwin, Symbian OS) UTF-32 if sizeof(wchar_t) == 4 (on Linux) Parameter str points to a null-terminated wide string. Parameter num_chars may additionally limit the number of wchar_t characters processed. -1 is used when the entire string should be processed. If the string contains code points that are not valid Unicode code points (i.e. outside of Unicode range U+0 to U+10FFFF) they will be output as '(Invalid Unicode 0xXXXXXXXXX)'. If the string is in UTF16 encoding and contains invalid UTF-16 surrogate pairs, values in those pairs will be encoded as individual Unicode characters from Basic Normal Plane.

85365. A function for deleting an object. Handy for being used as a functor.

85366. Pushes a trace defined by SCOPED_TRACE() on to the per-thread Google Test trace stack.

85367. Gets the number of failed tests in this test case.

85368. Returns a pathname for a file that does not currently exist. The pathname will be directory/base_name.extension or directory/base_name_<number>.extension if directory/base_name.extension already exists. The number will be incremented until a pathname is found that does not already exist. Examples: 'dir/foo_test.xml' or 'dir/foo_test_1.xml'. There could be a race condition if two or more processes are calling this function at the same time -- they could both pick the same filename.

85369. Death test constructor. Increments the running death test count for the current test.

85370. True iff at least one test has failed.

85371. The output file.

85372. Returns a copy of the FilePath with the directory part removed. Example: FilePath("path/to/file").RemoveDirectoryName() returns FilePath("file"). If there is no directory part ("just_a_file"), it returns the FilePath unmodified. If there is no file part ("just_a_dir") it returns an empty FilePath (""). On Windows platform, '\' is the path separator, otherwise it is '/'.

85373. Dynamic mutexes are initialized in the constructor.

85374. The following methods override what's in the TestEventListener class.

85375. Runs the given method and catches and reports C++ and/or SEH-style exceptions, if they are supported; returns the 0-value for type Result in case of an SEH exception.

85376. An error is OK if the directory exists.

85377. The value of GTEST_FLAG(catch_exceptions) at the moment RunAllTests() starts.

85378. Reads the entire content of a file as an std::string.

85379. NOTE: The user code can affect the way in which Google Test handles exceptions by setting GTEST_FLAG(catch_exceptions), but only before RUN_ALL_TESTS() starts. It is technically possible to check the flag after the exception is caught and either report or re-throw the exception based on the flag's value: try { // Perform the test method. } catch (...) { if (GTEST_FLAG(catch_exceptions)) // Report the exception as failure. else throw; // Re-throws the original exception. } However, the purpose of this flag is to allow the program to drop into the debugger when the exception is thrown. On most platforms, once the control enters the catch block, the exception origin information is lost and the debugger will stop the program at the point of the re-throw in this function -- instead of at the point of the original throw statement in the code under test. For this reason, we perform the check early, sacrificing the ability to affect Google Test's exception handling in the method where the exception is thrown.

85380. End XmlUnitTestResultPrinter

85381. **comment:** If we are in the child process of a death test, don't create/delete the premature exit file, as doing so is unnecessary and will confuse the parent process. Otherwise, create/delete the file upon entering/leaving this function. If the program somehow exits before this function has a chance to return, the premature-exit file will be left undeleted, causing a test runner that understands the premature-exit-file protocol to report the test as having failed.

label: code-design

85382. Index of the last death test case registered. Initially -1.

85383. In the debug version, Visual Studio pops up a separate dialog offering a choice to debug the aborted program. We need to suppress this dialog or it will pop up for every EXPECT/ASSERT_DEATH statement executed. Google Test will notify the user of any unexpected failure via stderr. VC++ doesn't define _set_abort_behavior() prior to the version 8.0. Users of prior VC versions shall suffer the agony and pain of clicking through the countless debug dialogs. TODO(vlidl@google.com): find a way to suppress the abort dialog() in the debug mode when compiled with VC 7.1 or lower.

85384. The AssumeRole process for a fork-and-exec death test. It re-executes the main program from the beginning, setting the --gtest_filter and --gtest_internal_run_death_test flags to cause only the current death test to be re-run.

85385. class TestEventRepeater This class forwards events to other event listeners.

85386. Print failure message from the assertion (e.g. expected this and got that).

85387. We don't use sprintf or strncpy, as they trigger a warning when compiled with VC++ 8.0.

85388. Tells the unit test event listener that the tests have just finished.

85389. The parsed value overflows as an Int32.

85390. Parses a string for an Int32 flag, in the form of "--flag=value". On success, stores the value of the flag in *value, and returns true. On failure, returns false without changing *value.

85391. namespace posix GTEST_OS_WINDOWS_MOBILE

85392. When def_optional is true, it's OK to not have a "=value" part.

85393. Sets the test part result reporter for the current thread.

85394. NOLINT GTEST_OS_WINDOWS

85395. The environment variable name for the test shard status file.

85396. **comment:** Prints an XML representation of a TestInfo object. TODO(wan): There is also value in printing properties with the plain printer.

label: code-design

85397. Returns the current OS stack trace as an std::string. The maximum number of stack frames to be included is specified by the gtest_stack_trace_depth flag. The skip_count parameter specifies the number of top frames to be skipped, which doesn't count against the number of frames to be included. For example, if Foo() calls Bar(), which in turn calls CurrentOsStackTraceExceptTop(1), Foo() will be included in the trace but Bar() and CurrentOsStackTraceExceptTop() won't.

85398. This doesn't throw as all user code that can throw are wrapped into exception handling code.

85399. Yes. Shift the remainder of the argv list left by one. Note that argv has (*argc + 1) elements, the last one always being NULL. The following loop moves the trailing NULL element as well.

85400. RemoveFileName returns the directory path with the filename removed. Example: FilePath("path/to/file").RemoveFileName() returns "path/to/". If the FilePath is "a_file" or "/a_file", RemoveFileName returns FilePath("./") or, on Windows, FilePath("\\"). If the filepath does not have a file, like "just/a/dir/", it returns the FilePath unmodified. On Windows platform, '\' is the path separator, otherwise it is '/'.

85401. A valid random seed must be in [1, kMaxRandomSeed].

85402. class UnitTestImpl

85403. **comment:** Since most methods are very similar, use macros to reduce boilerplate. This defines a member that forwards the call to all listeners.

label: code-design

85404. Helper function for *_STREQ on wide strings.

85405. Returns the message describing the last system error in errno.

85406. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

85407. Is the parsed value in the range of an Int32?
85408. Prints the given C string to the ostream.
85409. The list of reserved attributes used in the <testcase> element of XML output.
85410. A test case whose name matches this filter is considered a death test case and will be run before test cases whose name doesn't match this filter.
85411. **comment:** We cannot call numeric_limits::max() as it conflicts with the max() macro on Windows.
 label: code-design
85412. Windows CE doesn't define INVALID_FILE_ATTRIBUTES
85413. Prints a TestPartResult object.
85414. Converts the given epoch time in milliseconds to a date string in the ISO 8601 format, without the timezone information.
85415. Prints an XML summary of unit_test to output stream out.
85416. Initializes the event listener for streaming test results to a socket. Must not be called before InitGoogleTest.
85417. Returns the current time in milliseconds.
85418. Returns ParameterizedTestCaseRegistry object used to keep track of value-parameterized tests and instantiate and register them.
85419. Runs all tests in this UnitTest object, prints the result, and returns true if all tests are successful. If any exception is thrown during a test, this test is considered to be failed, but the rest of the tests will still be run.
85420. The newly initialized handle is accessible only in the parent process. To obtain one accessible within the child, we need to use DuplicateHandle.
85421. Adds a TestInfo to the unit test. Arguments: set_up_tc: pointer to the function that sets up the test case tear_down_tc: pointer to the function that tears down the test case test_info: the TestInfo object
85422. The parent process considers the death test to be a failure if it finds any data in our pipe. So, here we write a single flag byte to the pipe, then exit.
85423. The c'tor sets this object as the test part result reporter used by Google Test. The 'result' parameter specifies where to report the results. This reporter will only catch failures generated in the current thread. DEPRECATED
85424. Converts an array of wide chars to a narrow string using the UTF-8 encoding, and streams the result to the given Message object.
85425. Fired before each iteration of tests starts.
85426. Returns the current OS stack trace as an std::string. The maximum number of stack frames to be included is specified by the gtest_stack_trace_depth flag. The skip_count parameter specifies the number of top frames to be skipped, which doesn't count against the number of frames to be included. For example, if Foo() calls Bar(), which in turn calls GetCurrentOsStackTraceExceptTop(..., 1), Foo() will be included in the trace but Bar() and GetCurrentOsStackTraceExceptTop() won't.
85427. The current color.
85428. GTEST_HAS_GLOBAL_STRING
85429. _MSC_VER
85430. The decomposed components of the gtest_internal_run_death_test flag, parsed when RUN_ALL_TESTS is called.
85431. **comment:** MinGW has gettimeofday() but not _ftime64(). TODO(kenton@google.com): Use autoconf to detect availability of gettimeofday().
 TODO(kenton@google.com): There are other ways to get the time on Windows, like GetTickCount() or GetSystemTimeAsFileTime(). MinGW supports these. consider using them instead.
 label: code-design
85432. Increments the test part result count and remembers the result. This method is from the TestPartResultReporterInterface interface.
85433. Note that Google Test currently only reports elapsed time for each test iteration, not for the entire test program.
85434. May appear in a well-formed XML document?
85435. Possibly yields the rest of the thread's time slice to other threads.
85436. Prints a segment of bytes in the given object.
85437. Everything up to the dash
85438. Everything after the dash
85438. **comment:** Name of the temporary file holding the stderr output.
 label: code-design
85439. Info about the current test.
85440. Prints a wchar_t c as if it's part of a string literal, escaping it when necessary; returns how c was formatted.
85441. Appends a TestPartResult to the array.
85442. Some Linux distributions define PATH_MAX here. GTEST_OS_WINDOWS_MOBILE
85443. Notify the streaming server to stop.
85444. Google Test implements this protocol for catching that a test program exits before returning control to Google Test: 1. Upon start, Google Test creates a file whose absolute path is specified by the environment variable TEST_PREMATURE_EXIT_FILE. 2. When Google Test has finished its work, it deletes the file. This allows a test runner to set TEST_PREMATURE_EXIT_FILE before running a Google-Test-based test program and check the existence of the file at the end of the test execution to see if it has exited prematurely.
85445. This is called from a death test parent process to read a failure message from the death test child process and log it with the FATAL severity. On Windows, the message is read from a pipe handle. On other platforms, it is read from a file descriptor.
85446. Returns the name of the requested output file, or the default if none was explicitly specified.
85447. Generates a textual description of a given exit code, in the format specified by wait(2).
85448. Same as above, but you can choose the interception scope of this object.
85449. True if the death test child process has been successfully spawned.
85450. No need to calculate the full pattern when the regex is invalid.
85451. wchar_t is native
85452. A filter is a colon-separated list of patterns. It matches a test if any pattern in it matches the test.
85453. This code is unreachable but some compilers may not realize that.
85454. Restores the original stream.
85455. Converts a wide C string to an std::string using the UTF-8 encoding. NULL will be converted to "(null)".
85456. Deletes every Environment.
85457. Sets the OS stack trace getter. Does nothing if the input and the current OS stack trace getter are the same; otherwise, deletes the old getter and makes the input the current getter.
85458. Appends the user message if it's non-empty.
85459. Returns true if pathname describes something findable in the file-system, either a file, directory, or whatever.
85460. Restores the text color.
85461. Returns the current OS stack trace getter if it is not NULL; otherwise, creates an OsStackTraceGetter, makes it the current getter, and returns it.
85462. Gets the number of disabled tests.
85463. Parses 'str' for a 32-bit signed integer. If successful, writes the result to *value and returns true; otherwise leaves *value unchanged and returns false.
85464. **comment:** Helpers for printing colored strings to stdout. Note that on Windows, we cannot simply emit special characters and have the terminal change colors.
 This routine must actually emit the characters rather than return a string that would be colored when printed, as can be done on Linux.
 label: code-design
85465. The d'tor restores the values of all Google Test flags.
85466. Clears the object.
85467. iteration
85468. These constants are the same as are used in glibc's rand(3).
85469. Appends the TestPartResult object to the TestPartResultArray received in the constructor. This method is from the TestPartResultReporterInterface interface.
85470. **comment:** Shuffles the tests in this test case.
 label: test
85471. Implements the helper function for {ASSERT|EXPECT}_GE with int or enum arguments.
85472. !GTEST_OS_WINDOWS
85473. The exit status of the child process.
85474. Maximum stack alignment in bytes: For a downward-growing stack, this amount is subtracted from size of the stack space to get an address that is within the stack space and is aligned on all systems we care about. As far as I know there is no ABI with stack alignment greater than 64. We assume stack and stack_size already

have alignment of kMaxStackAlignment.

85475. Sets up the test fixture. A sub-class may override this.

85476. GTEST_IMPLEMENTATION_ is defined to 1 iff the current translation unit is part of Google Test's implementation; otherwise it's undefined.

85477. Dereference NULL through a volatile pointer to prevent the compiler from removing. We use this rather than abort() or __builtin_trap() for portability: Symbian doesn't implement abort() well, and some debuggers don't correctly trap abort().

85478. Matches a repeated regex atom followed by a valid simple regular expression. The regex atom is defined as c if escaped is false, or \c otherwise. repeat is the repetition meta character (?*, ?, or +). The behavior is undefined if str contains too many characters to be indexable by size_t, in which case the test will probably time out anyway. We are fine with this limitation as std::string has it too.

85479. AssertionResult constructors. Used in EXPECT_TRUE/FALSE(assertion_result).

85480. Make sure AddressSanitizer does not tamper with the stack here.

85481. **comment:** Constructs an empty Message. We allocate the stringstream separately because otherwise each use of ASSERT/EXPECT in a procedure adds over 200 bytes to the procedure's stack frame leading to huge stack frames in some cases; gcc does not reuse the stack space.
label: code-design

85482. **comment:** TODO(wan@google.com): fix the source file location in the assertion failures to match where the regex is used in user code.
label: code-design

85483. MSVC and C++ Builder define __int64 instead of the standard long long.

85484. Obtains the current directory and sets it to be closed in the child process.

85485. Assesses the success or failure of a death test, using both private members which have previously been set, and one argument: Private data members: outcome: An enumeration describing how the death test concluded: DIED, LIVED, THREW, or RETURNED. The death test fails in the latter three cases. status: The exit status of the child process. On *nix, it is in the format specified by wait(2). On Windows, this is the value supplied to the ExitProcess() API or a numeric code of the exception that terminated the program. regex: A regular expression object to be applied to the test's captured standard error output; the death test fails if it does not match. Argument: status_ok: true if exit_status is acceptable in the context of this particular death test, which fails if it is false Returns true iff all of the above conditions are met. Otherwise, the first failing condition, in the order given above, is the one that is reported. Also sets the last death test message string.

85486. YYYY-MM-DDThh:mm:ss

85487. Verifies that the given attribute belongs to the given element and streams the attribute as XML.

85488. Clears the results of all tests in this test case.

85489. Makes a failed assertion result.

85490. Used in ThreadLocal.

85491. **comment:** Returns the TestResult for the test that's currently running, or the TestResult for the ad hoc test if no test is running.
label: test

85492. Returns the current OS stack trace as an std::string. Parameters: max_depth - the maximum number of stack frames to be included in the trace. skip_count - the number of top frames to be skipped; doesn't count against max_depth.

85493. Returns true iff the test case passed.

85494. The returned handle will be kept in thread_map and closed by watcher_thread in WatcherThreadFunc.

85495. By default, we want there to be enough precision when printing a double to a Message.

85496. Non-special character. Matches itself.

85497. GTEST_HAS_STD_WSTRING

85498. Parses the command line for Google Test flags, without initializing other parts of Google Test. The type parameter CharType can be instantiated to either char or wchar_t.

85499. Returns the TestInfo object for the test that's currently running, or NULL if no test is running.

85500. class SocketWriter

85501. Gets the random number generator.

85502. The child has acquired the write end of the pipe or exited. We release the handle on our side and continue.

85503. **comment:** First, print c as a literal in the most readable form we can find.
label: code-design

85504. gtest_break_on_failure takes precedence over gtest_throw_on_failure. This allows a user to set the latter in the code (perhaps in order to use Google Test assertions with another testing framework) and specify the former on the command line for debugging.

85505. Iterate the edits until we found enough suffix for the hunk or the input is over.

85506. No. Let's create one.

85507. __clang_analyzer__

85508. A successful match can be anywhere in str.

85509. Skips to the first non-space char in str. Returns an empty string if str contains only whitespace characters.

85510. Returns the i-th test property. i can range from 0 to test_property_count() - 1. If i is not in that range, aborts the program.

85511. Implements the helper function for {ASSERT|EXPECT}_GT with int or enum arguments.

85512. The helper function for {ASSERT|EXPECT}_STRCASENE.

85513. Ensure that Google Test output is printed before, e.g., heapchecker output.

85514. Adds a TestProperty to the current TestResult object when invoked in a context of a test, to current test case's ad_hoc_test_result when invoke from SetUpTestCase/TearDownTestCase, or to the global property set otherwise. If the result already contains a property with the same key, the value will be updated.

85515. class TestResult

85516. We don't want to run the initialization code twice.

85517. An escape sequence

85518. Determines whether a string has a prefix that Google Test uses for its flags, i.e., starts with GTEST_FLAG_PREFIX_ or GTEST_FLAG_PREFIX_DASH_. If Google Test detects that a command line flag has its prefix but is not recognized, it will print its help message. Flags starting with GTEST_INTERNAL_PREFIX_ followed by "internal_" are considered Google Test internal flags and do not trigger the help message.

85519. Non-inheritable.

85520. Routine for aborting the program which is safe to call from an exec-style death test child process, in which case the error message is propagated back to the parent process. Otherwise, the message is simply printed to stderr. In either case, the program then exits with status 1.

85521. GTEST_SRC_GTEST_INTERNAL_INL_H

85522. Resetting with the same handle we already own is invalid.

85523. Reads the GTEST_SHARD_STATUS_FILE environment variable, and creates the file if the variable is present. If a file already exists at this location, this function will write over it. If the variable is present, but the file cannot be created, prints an error and exits.

85524. Parses the environment variable var as an Int32. If it is unset, returns default_val. If it is not an Int32, prints an error and aborts.

85525. Waits for the child in a death test to exit, returning its exit status, or 0 if no child process exists. As a side effect, sets the outcome data member.

85526. Default security. Default stack size

85527. MINGW <time.h> provides neither localtime_r nor localtime_s, but uses Windows' localtime(), which has a thread-local tm buffer. NOLINT

85528. NOLINT NOLINT NOLINT NOLINT

85529. The vector of environments that need to be set-up/torn-down before/after the tests are run.

85530. Returns whether the given character is a valid path separator.

85531. Stops capturing stdout and returns the captured string.

85532. Reads the entire content of a file as a string.

85533. **comment:** WindowsDeathTest implements death tests on Windows. Due to the specifics of starting new processes on Windows, death tests there are always threadsafe, and Google Test considers the --gtest_death_test_style=fast setting to be equivalent to --gtest_death_test_style=threadsafethere. A few implementation notes: Like the Linux version, the Windows implementation uses pipes for child-to-parent communication. But due to the specifics of pipes on Windows, some extra steps are required: 1. The parent creates a communication pipe and stores handles to both ends of it. 2. The parent starts the child and provides it with the information necessary to acquire the handle to the write end of the pipe. 3. The child acquires the write end of the pipe and signals the parent using a Windows event. 4. Now the parent can release the write end of the pipe on its side. If this is done before step 3, the object's reference count goes down to 0 and it is destroyed, preventing the child from acquiring it. The parent now has to release it, or read operations on the read end of the pipe will not return when the child

terminates. 5. The parent reads child's output through the pipe (outcome code and any possible error messages) from the pipe, and its stderr and then determines whether to fail the test. Note: to distinguish Win32 API calls from the local method and function calls, the former are explicitly resolved in the global namespace.

label: code-design

85534. Prints a char c as if it's part of a string literal, escaping it when necessary; returns how c was formatted.

85535. Returns an XML-escaped copy of the input string str. If is_attribute is true, the text is meant to appear as an attribute value, and normalizable whitespace is preserved by replacing it with character references.

85536. class String.

85537. **comment:** GTEST_HAS_DEATH_TEST && !GTEST_OS_WINDOWS

label: test

85538. **comment:** There's no portable way to detect the number of threads, so we just return 0 to indicate that we cannot detect it.

label: code-design

85539. Looks up the human-readable system message for the HRESULT code and since we're not passing any params to FormatMessage, we don't want inserts expanded.

85540. Returns the test part result reporter for the current thread.

85541. Creates the test object.

85542. Finds and returns a TestCase with the given name. If one doesn't exist, creates one and returns it. Arguments: test_case_name: name of the test case type_param: the name of the test's type parameter, or NULL if this is not a typed or a type-parameterized test. set_up_tc: pointer to the function that sets up the test case tear_down_tc: pointer to the function that tears down the test case

85543. Helper functions for implementing IsSubString() and IsNotSubstring().

85544. Compares the name of each test with the user-specified filter to decide whether the test should be run, then records the result in each TestCase and TestInfo object. If shard_tests == true, further filters tests based on sharding variables in the environment - see

<http://code.google.com/p/googletest/wiki/GoogleTestAdvancedGuide>. Returns the number of tests that should run.

85545. There's no guarantee that a test has write access to the current directory, so we create the temporary file in the /tmp directory instead. We use /tmp on most systems, and /sdcard on Android. That's because Android doesn't have /tmp.

85546. Returns the type ID of ::testing::Test. We should always call this instead of GetTypeId< ::testing::Test>() to get the type ID of testing::Test. This is to work around a suspected linker bug when using Google Test as a framework on Mac OS X. The bug causes GetTypeId< ::testing::Test>() to return different values depending on whether the call is from the Google Test framework itself or from user test code. GetTestTypeId() is guaranteed to always return the same value, as it always calls GetTypeId<>() from the gtest.cc, which is within the Google Test framework.

85547. Returns the standard listener responsible for the default console output. Can be removed from the listeners list to shut down default console output. Note that removing this object from the listener list with Release transfers its ownership to the user.

85548. Registers thread_local_instance as having value on the current thread. Returns a value that can be used to identify the thread from other threads.

85549. Internal helper for printing the list of failed tests.

85550. Function for supporting the gtest_catch_exception flag.

85551. Conceptually, we split the string into segments divided by escape sequences. Then we print one segment at a time. At the end of each iteration, the str pointer advances to the beginning of the next segment.

85552. Utility functions for encoding Unicode text (wide strings) in UTF-8.

85553. Maps a thread to a set of ThreadIdToThreadLocals that have values instantiated on that thread and notifies them when the thread exits. A ThreadLocal instance is expected to persist until all threads it has values on have terminated.

85554. Will be overridden by the flag before first use.

85555. An empty regex matches a prefix of anything.

85556. **comment:** CodeGear C++Builder insists on a public destructor for the default implementation. Use this implementation to keep good OO design with private destructor.

label: code-design

85557. Iterates over a vector of TestCases, keeping a running sum of the results of calling a given int-returning method on each. Returns the sum.

85558. Internal helper functions for implementing the simple regular expression matcher.

85559. Matches the full name of each test against the user-specified filter to decide whether the test should run, then records the result in each TestCase and TestInfo object. If shard_tests == HONOR_SHARDING_PROTOCOL, further filters tests based on sharding variables in the environment. Returns the number of tests that should run.

85560. The default output file for XML output.

85561. Compares two C strings. Returns true iff they have the same content. Unlike strcmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL C string, including the empty string.

85562. Names of the flags (needed for parsing Google Test flags).

85563. GTEST_USES_SIMPLE_RE

85564. If this file is included from the user's code, just say no.

85565. Returns true iff ch belongs to the given classification. Unlike similar functions in <ctype.h>, these aren't affected by the current locale.

85566. Implements the helper function for {ASSERT|EXPECT}_LT with int or enum arguments.

85567. NOLINT GTEST_OS_WINDOWS_MINGW

85568. Implemented as an explicit loop since std::count_if() in libCstd on Solaris has a non-standard signature.

85569. Command-line arguments for the child's call to exec File descriptor to close; the read end of a pipe

85570. Gets the number of failed tests.

85571. Destructor of UnitTest.

85572. Gets the number of all test cases.

85573. Returns a newly created InternalRunDeathTestFlag object with fields initialized from the GTEST_FLAG(internal_run_death_test) flag if the flag is specified; otherwise returns NULL.

85574. Returns true iff the test part non-fatally failed.

85575. Utility functions and classes used by the Google C++ testing framework. Author: wan@google.com (Zhanyong Wan) This file contains purely Google Test's internal implementation. Please DO NOT #INCLUDE IT IN A USER PROGRAM.

85576. Registers all parameterized tests defined using TEST_P and INSTANTIATE_TEST_CASE_P, creating regular tests for each test/parameter combination. This method can be called more than once; it has guards protecting from registering the tests more than once. If value-parameterized tests are disabled, RegisterParameterizedTests is present but does nothing.

85577. Streams test results to the given port on the given host machine.

85578. Sets the TestInfo object for the test that's currently running. If current_test_info is NULL, the assertion results will be stored in ad_hoc_test_result_.

85579. This 'if' can only be successful at most once, so theoretically we could break out of the loop here, but we don't bother doing so.

85580. A test whose test case name or test name matches this filter is disabled and not run.

85581. Makes a failed assertion result with the given failure message. Deprecated; use AssertionFailure() << message.

85582. **comment:** We make replace a little more expensive than add/remove to lower their priority.

label: code-design

85583. Constructor. TestNameIs has NO default constructor.

85584. Creates the test object, runs it, records its result, and then deletes it.

85585. Performs an in-place shuffle of a range of the vector's elements. 'begin' and 'end' are element indices as an STL-style range; i.e. [begin, end) are shuffled, where 'end' == size() means to shuffle to the end of the vector.

85586. Signals that the death test code which should have exited, didn't. Should be called only in a death test child process. Writes a status byte to the child's status file descriptor, then calls _exit(1).

85587. **comment:** Used by the GTEST_SUPPRESS_UNREACHABLE_CODE_WARNING_BELOW_ macro to suppress unreachable code warnings.

label: code-design

85588. Default security attributes. Do not reset automatically. Initially unset. Anonymous event.

85589. Protects access to GetThreadLocalsMapLocked() and its return value.

85590. namespace internal namespace testing Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of

conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhangyong Wan)

85591. Generates non-fatal failures and returns false if regex is invalid; otherwise returns true.
85592. Implements the RE class.
85593. Configures listeners for XML output. This makes it possible for users to shut down the default XML output before invoking RUN_ALL_TESTS.
85594. Rounds up to 2-byte boundary.
85595. End of class Streaming Listener GTEST_CAN_STREAM_RESULTS_
85596. If a path to the premature-exit file is specified...
85597. Provides cross platform implementation for some death functionality.
85598. Returns true iff the current test has a non-fatal failure.
85599. # of bytes read in the last fread() # of bytes read so far
85600. The code const char kFoo[] = "foo"; generates an array of 4, not 3, elements, with the last one being '\0'. Therefore when printing a char array, we don't print the last element if it's '\0', such that the output matches the string literal as it's written in the source code.
85601. regfree'ing an invalid regex might crash because the content of the regex is undefined. Since the regex's are essentially the same, one cannot be valid (or invalid) without the other being so too.
85602. Runs all tests in this UnitTest object, prints the result, and returns true if all tests are successful. If any exception is thrown during a test, the test is considered to be failed, but the rest of the tests will still be run. When parameterized tests are enabled, it expands and registers parameterized tests first in RegisterParameterizedTests(). All other functions called from RunAllTests() may safely assume that parameterized tests are ready to be counted and run.
85603. **comment:** Valid only for fast death tests. Indicates the code is running in the child process of a fast style death test.
label: test
85604. On Windows, '\\' is the standard path separator, but many tools and the Windows API also accept '/' as an alternate path separator. Unless otherwise noted, a file path can contain either kind of path separators, or a mixture of them.
85605. Concrete class for actually writing strings to a socket.
85606. Returns true iff the wildcard pattern matches the string. The first ':' or '\0' character in pattern marks the end of it. This recursive algorithm isn't very efficient, but is clear and works well enough for matching test names, which are short.
85607. **comment:** Two fixture classes with the same name appear in two different namespaces, which is not allowed. Tell the user how to fix this.
label: code-design
85608. The regular expression which test output must match. DeathTestImpl doesn't own this object and should not attempt to delete it.
85609. Stores all ThreadIdToThreadLocals having values in a thread, indexed by thread's ID.
85610. Helpers for setting up / tearing down the given environment. They are for use in the ForEach() function.
85611. Helper function for implementing ASSERT_NEAR.
85612. End TestEventRepeater
85613. GTEST_HAS_DEATH_TEST
85614. class OsStackTraceGetter
85615. !GTEST_OS_WINDOWS_MOBILE
85616. Shuffles the tests inside each test case.
85617. Descriptor to the read end of the pipe to the child process. It is always -1 in the child process. The child keeps its write end of the pipe in write_fd_.
85618. Suppress clang analyzer warnings.
85619. Streams an XML representation of a TestInfo object.
85620. Parses a string for a string flag, in the form of "--flag=value". On success, stores the value of the flag in *value, and returns true. On failure, returns false without changing *value.
85621. Called in the parent process only. Reads the result code of the death test child process via a pipe, interprets it to set the outcome_ member, and closes read_fd_. Outputs diagnostics and terminates in case of unexpected codes.
85622. Stops capturing the output stream and returns the captured string.
85623. The working directory when the first TEST() or TEST_F() was executed.
85624. We print the help here instead of in RUN_ALL_TESTS(), as the latter may not be called at all if the user is using Google Test with another testing framework.
85625. namespace
85626. Formats the given time in milliseconds as seconds.
85627. MSVC and C++Builder do not provide a definition of STDERR_FILENO.
85628. MMAP_ANONYMOUS is not defined on Mac, so we use MAP_ANON instead.
85629. Compares two wide C strings. Returns true iff they have the same content. Unlike wcsncmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL C string, including the empty string.
85630. UponLeavingGTest() should be called immediately before Google Test calls user code. It saves some information about the current stack that CurrentStackTrace() will use to find and hide Google Test stack frames.
85631. This class generates an XML output file.
85632. In a particular thread, maps a ThreadLocal object to its value.
85633. Prints the given C-string on a single line by replacing all '\n' characters with string "\n". If the output takes more than max_length characters, only prints the first max_length characters and "...".
85634. **comment:** Returns true iff test_case contains at least one test that should run.
label: test
85635. Returns true iff the test failed.
85636. "\\p" where p is pattern_char.
85637. Fields for saving the original values of flags.
85638. Gets the number of all test cases that contain at least one test that should run.
85639. Shuffles the non-death test cases.
85640. This condition is always false so AlwaysTrue() never actually throws, but it makes the compiler think that it may throw.
85641. **comment:** Spawns a child process with the same executable as the current process in a thread-safe manner and instructs it to run the death test. The implementation uses fork(2) + exec. On systems where clone(2) is available, it is used instead, being slightly more thread-safe. On QNX, fork supports only single-threaded environments, so this function uses spawn(2) there instead. The function dies with an error message if anything goes wrong.
label: code-design
85642. A macro for implementing the helper functions needed to implement ASSERT_?? and EXPECT_?? with integer or enum arguments. It is here just to avoid copy-and-paste of similar code.
85643. Returns the i-th element of the vector, or default_value if i is not in range [0, v.size()).
85644. NOLINT This exception type can only be thrown by a failed Google Test assertion with the intention of letting another testing framework catch it. Therefore we just re-throw it.
85645. **comment:** Gets the system's human readable message string for this HRESULT.
label: code-design
85646. Gets the time of the test program start, in ms from the start of the UNIX epoch.
85647. NOLINT NOLINT Declares vsnprintf(). This header is not available on Windows. NOLINT NOLINT NOLINT

85648. Determines if the arguments constitute UTF-16 surrogate pair and thus should be combined into a single Unicode code point using CreateCodePointFromUtf16SurrogatePair.
85649. GTEST_OS_WINDOWS && !defined(_GNUC_)
85650. Protects read and write access to global_test_part_result_reporter_.
85651. Yes. Inserts the test case after the last death test case defined so far. This only works when the test cases haven't been shuffled. Otherwise we may end up running a death test after a non-death test.
85652. **comment:** TODO(wan@google.com): on Windows \some\path is not an absolute path (as its meaning depends on the current drive), yet the following logic for turning it into an absolute path is wrong. Fix it.
label: code-design
85653. Does not return.
85654. **comment:** Print a unified diff header for one hunk. The format is "@@ -<left_start>,<left_length> +<right_start>,<right_length> @@" where the left/right parts are omitted if unnecessary.
label: code-design
85655. True iff PostFlagParsingInit() has been called.
85656. The private implementation of the UnitTest class. We don't protect the methods under a mutex, as this class is not accessible by a user and the UnitTest class that delegates work to this class does proper locking.
85657. Wait until the child either signals that it has acquired the write end of the pipe or it dies.
85658. A concrete death test class that forks and re-executes the main program from the beginning, with command-line flags set that cause only this specific death test to be run.
85659. Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: mheule@google.com (Markus Heule) Google C++ Testing Framework (Google Test) Sometimes it's desirable to build Google Test by compiling a single file. This file serves this purpose.
85660. Given directory = "dir", relative_path = "test.xml", returns "dir/test.xml". On Windows, uses \ as the separator rather than /.
85661. This predicate-formatter checks that 'results' contains a test part failure of the given type and that the failure message contains the given substring.
85662. Tears down the test fixture. A sub-class may override this.
85663. We are leaking the descriptor here because on some platforms (i.e., when built as Windows DLL), destructors of global objects will still run after calling _exit(). On such systems, write_fd_ will be indirectly closed from the destructor of UnitTestImpl, causing double close if it is also closed here. On debug configurations, double close may assert. As there are no in-process buffers to flush here, we are relying on the OS to close the descriptor after the process terminates when the destructors are not run. Exits w/o any normal exit hooks (we were supposed to crash)
85664. Formats an int value as "%X".
85665. namespace internal2
85666. class TestEventListeners
85667. The helper function for {ASSERT|EXPECT}_STRCASEEQ.
85668. Initializes an RE from its string representation.
85669. If input name has trailing separator character, remove it and return the name, otherwise return the name string unmodified. On Windows platform, uses \ as the separator, other platforms use /.
85670. GTEST_CAN_STREAM_RESULTS_
85671. Is a whitespace character that is normalized to a space character when it appears in an XML attribute value?
85672. Skips the pattern separator (the ':' character).
85673. Returns the output format, or "" for normal printed output.
85674. The parsed value overflows as a long. (strtol() returns LONG_MAX or LONG_MIN when the input overflows.)
85675. Returns the current application's name, removing directory path if that is present. Used by UnitTestOptions::GetOutputFile.
85676. regex isn't empty, isn't "\$", and doesn't start with a repetition. We match the first atom of regex with the first character of str and recurse.
85677. SetErrorMode doesn't exist on CE.
85678. MatchRepetitionAndRegexAtHead() calls MatchRegexAtHead(), so here's an indirect recursion. It terminates as the regex gets shorter in each recursion.
85679. **comment:** TODO(wan): report the reason of the failure. We don't do it for now as: 1. There is no urgent need for it. 2. It's a bit involved to make the errno variable thread-safe on all three operating systems (Linux, Windows, and Mac OS). 3. To interpret the meaning of errno in a thread-safe way, we need the strerror_r() function, which is not available on Windows.
label: requirement
85680. **comment:** task_threads allocates resources in thread_list and we need to free them to avoid leaks.
label: code-design
85681. PID of child process during death test; 0 in the child process itself.
85682. ForkingDeathTest provides implementations for most of the abstract methods of the DeathTest interface. Only the AssumeRole method is left undefined.
85683. Sets *value to the value of the flag.
85684. namespace internal namespace testing
85685. type_expr
85686. Converts the string value to a bool.
85687. The AssumeRole process for a Windows death test. It creates a child process with the same executable as the current process to run the death test. The child process is given the --gtest_filter and --gtest_internal_run_death_test flags such that it knows to run the current death test only.
85688. unit_test
85689. Returns the i-th test among all the tests. i can range from 0 to total_test_count() - 1. If i is not in that range, returns NULL.
85690. Returns the vector of environments that need to be set-up/torn-down before/after the tests are run.
85691. Returns true if FilePath ends with a path separator, which indicates that it is intended to represent a directory. Returns false otherwise. This does NOT check that a directory (or file) actually exists.
85692. **comment:** Converts the given time in milliseconds to a date string in the ISO 8601 format, without the timezone information. N.B.: due to the use of the non-reentrant localtime() function, this function is not thread safe. Do not use it in any code that can be called from multiple threads.
label: code-design
85693. MSVC compiler can be configured to define wchar_t as a typedef of unsigned short. Defining an overload for const wchar_t* in that case would cause pointers to unsigned shorts be printed as wide strings, possibly accessing more memory than intended and causing invalid memory accesses. MSVC defines _NATIVE_WCHAR_T_DEFINED symbol when wchar_t is implemented as a native type.
85694. Prints a (const) char/wchar_t array of 'len' elements, starting at address 'begin'. CharType must be either char or wchar_t.
85695. Disables event forwarding if the control is currently in a death test subprocess. Must not be called before InitGoogleTest.
85696. Returns a pointer to the last occurrence of a valid path separator in the FilePath. On Windows, for example, both "/" and "\\" are valid path separators. Returns NULL if no path separator was found.
85697. **comment:** Notifies the unit test event listener that a test has just finished.
label: test
85698. **comment:** Adds a test to this test case. Will delete the test upon destruction of the TestCase object.
label: test

85699. No error.
85700. **comment:** NOLINT - this is more readable than unbalanced brackets inside #if.
label: code-design
85701. Returns true if pathname describes a directory in the file-system that exists.
85702. Returns true iff regular expression re matches the entire str.
85703. GTEST_OS_QNX
85704. Converts a TestPartResult::Type enum to human-friendly string representation. Both kNonFatalFailure and kFatalFailure are translated to "Failure", as the user usually doesn't care about the difference between the two when viewing the test result.
85705. **comment:** An enumeration describing all of the possible ways that a death test can conclude. DIED means that the process died while executing the test code; LIVED means that process lived beyond the end of the test code; RETURNED means that the test statement attempted to execute a return statement, which is not allowed; THREW means that the test statement returned control by throwing an exception. IN_PROGRESS means the test has not yet concluded.
TODO(vladl@google.com): Unify names and possibly values for AbortReason, DeathTestOutcome, and flag characters above.
label: code-design
85706. Creates and returns a death test by dispatching to the current death test factory.
85707. The list of reserved attributes used in the <testsuites> element of XML output.
85708. Prints a wchar_t as a symbol if it is printable or as its internal code otherwise and also as its code. L'\0' is printed as "L'\\0'".
85709. Even if sharding is not on, test runners may want to use the GTEST_SHARD_STATUS_FILE to query whether the test supports the sharding protocol.
85710. Connect the client socket to the server socket.
85711. **comment:** Static mutexes are leaked intentionally. It is not thread-safe to try to clean them up. TODO(yukawa): Switch to Slim Reader/Writer (SRW) Locks, which requires nothing to clean it up but is available only on Vista and later. <http://msdn.microsoft.com/en-us/library/windows/desktop/aa904937.aspx>
label: requirement
85712. A helper class that creates the premature-exit file in its constructor and deletes the file in its destructor.
85713. Different Windows APIs may use either of these values to represent an invalid handle.
85714. Info about the first test in the current test case.
85715. Generates a textual failure message when a death test finds more than one thread running, or cannot determine the number of threads, prior to executing the given statement. It is the responsibility of the caller not to pass a thread_count of 1.
85716. Object that captures an output stream (stdout/stderr).
85717. A test filter that matches everything.
85718. GTEST_OS_LINUX
85719. AssertHelper constructor.
85720. **comment:** We want to call regcomp(&partial_regex_, ...) even if the previous expression returns false. Otherwise partial_regex_ may not be properly initialized can may cause trouble when it's freed. Some implementation of POSIX regex (e.g. on at least some versions of Cygwin) doesn't accept the empty string as a valid regex. We change it to an equivalent form "() to be safe.
label: code-design
85721. Previous character is of "x.." form and this character can be interpreted as another hexadecimal digit in its number. Break string to disambiguate.
85722. Formats a file location for compiler-independent XML output. Although this function is not platform dependent, we put it next to FormatFileLocation in order to contrast the two functions. Note that FormatCompilerIndependentFileLocation() does NOT append colon to the file location it produces, unlike FormatFileLocation().
85723. A predicate that checks the name of a TestCase against a known value. This is used for implementation of the UnitTest class only. We put it in the anonymous namespace to prevent polluting the outer namespace. TestCaseNameIs is copyable.
85724. Asserts that val1 is less than, or almost equal to, val2. Fails otherwise. In particular, it fails if either val1 or val2 is NaN.
85725. Clear the close-on-exec flag on the write end of the pipe, lest it be closed when the child process does an exec:
85726. Should not get here.
85727. The following lines pull in the real gtest *.cc files. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhangyong Wan) The Google C++ Testing Framework (Google Test)
85728. Repeats the call to the post-flag parsing initialization in case the user didn't call InitGoogleTest.
85729. Generate unique file name.
85730. Captures the value of GTEST_FLAG(catch_exceptions). This value will be used for the duration of the program.
85731. GTEST_HAS_GLOBAL_WSTRING
85732. Configures listeners for streaming test results to the specified server.
85733. Gets the name of the TEST and the name of the TEST_F. Note that first_is_TEST and this_is_TEST cannot both be true, as the fixture IDs are different for the two tests.
85734. Deletes every Test in the collection.
85735. **comment:** Is this a death test case?
label: test
85736. The role interface for getting the OS stack trace as a string.
85737. If def_optional is true and there are more characters after the flag name, or if def_optional is false, there must be a '=' after the flag name.
85738. **comment:** Notifies the unit test event listeners that a test is about to start.
label: test
85739. This class saves the values of all Google Test flags in its c'tor, and restores them in its d'tor.
85740. using this in initializer
85741. This group of overloaded functions return true iff needle is a substring of haystack. NULL is considered a substring of itself only.
85742. Returns map of thread local instances.
85743. **comment:** Compares the full test names with the filter to decide which tests to run.
label: test
85744. On Windows, death tests are thread-safe regardless of the value of the death_test_style flag.
85745. Returns repeater that broadcasts the TestEventListener events to all subscribers.
85746. Set close_fd to be closed after spawn.
85747. Returns true iff the test name of test property matches on key_.
85748. Returns true iff ch appears anywhere in str (excluding the terminating '\0' character).
85749. Returns the number of elements in the given container that satisfy the given predicate.
85750. Parses the environment variable as a decimal integer.
85751. Points to (but doesn't own) the per-thread test part result reporter.
85752. Prints an XML representation of a TestCase object
85753. Converts a Unicode code point to a narrow string in UTF-8 encoding. code_point parameter is of type UInt32 because wchar_t may not be wide enough to contain a code point. If the code_point is not a valid Unicode code point (i.e. outside of Unicode range U+0 to U+10FFFF) it will be converted to "(Invalid Unicode 0xFFFFFFFF)".
85754. all done with this structure

85755. A copy of all command line arguments. Set by InitGoogleTest().
85756. Utilities needed for death tests.
85757. Returns a random seed in range [1, kMaxRandomSeed] based on the given --gtest_random_seed flag value.
85758. Points to (but doesn't own) the global test part result reporter.
85759. Child process handle.
85760. Returns the working directory when the first TEST() or TEST_F() was executed.
label: GTEST_OS_MAC
85762. no source, we're asking system the error no line width restrictions output buffer buf size no arguments for inserts Trims tailing white space (FormatMessage leaves a trailing CR-LF)
85763. The destructor of SingleFailureChecker verifies that the given TestPartResultArray contains exactly one failure that has the given type and contains the given substring. If that's not the case, a non-fatal failure will be generated.
85764. Default buffer size.
85765. Parses a string as a command line flag. The string should have the format "--flag=value". When def_optional is true, the "=value" part can be omitted. Returns the value of the flag, or NULL if the parsing failed.
label: GTEST_INCLUDE_GTEST_GTEST_SPI_H
85766. If the test part succeeded, we don't need to do anything.
85768. **comment:** Gets the number of disabled tests that will be reported in the XML report.
label: test
85769. The internal implementation of InitGoogleTest(). The type parameter CharType can be instantiated to either char or wchar_t.
85770. Loop through all the results and connect to the first we can.
85771. Used by UnitTest::Run() to capture the state of GTEST_FLAG(catch_exceptions) at the moment it starts.
85772. We print the type parameter on a single line to make the output easy to parse by a program.
85773. Returns the random seed used at the start of the current test run.
85774. Create directories so that path exists. Returns true if successful or if the directories already exist; returns false if unable to create directories for any reason.
85775. Returns the current application's name, removing directory path if that is present.
85776. Finds the next pattern in the filter.
85777. We have enough matches at the head, and the tail matches too. Since we only care about *whether* the pattern matches str (as opposed to *how* it matches), there is no need to find greedy match.
85778. Text printed in Google Test's text output and --gunit_list_tests output to label the type parameter and value parameter for a test.
85779. This macro is similar to GTEST_DEATH_TEST_CHECK_, but it is meant for evaluating any system call that fulfills two conditions: it must return -1 on failure, and set errno to EINTR when it is interrupted and should be tried again. The macro expands to a loop that repeatedly evaluates the expression as long as it evaluates to -1 and sets errno to EINTR. If the expression evaluates to -1 but errno is something other than EINTR, DeathTestAbort is called.
85780. Skips the stack frame for this function itself. NOLINT
85781. Do we see a Google Test flag?
85782. Find first edit.
85783. Restores the original test order after the iteration. This allows the user to quickly repro a failure that happens in the N-th iteration without repeating the first (N - 1) iterations. This is not enclosed in "if (GTEST_FLAG(shuffle)) { ... }", in case the user somehow changes the value of the flag somewhere (it's always safe to unshuffle the tests).
85784. Adds a failure if the key is a reserved attribute of the element named xml_element. Returns true if the property is valid.
85785. Signals the parent that the write end of the pipe has been acquired so the parent can release its own write end.
85786. Lists the tests and exits if the --gtest_list_tests flag was specified.
85787. Decrements the argument count.
85788. Gets the number of tests that should run.
85789. The list of event listeners that can be used to track events inside Google Test.
85790. size_t has the same width as pointers on both 32-bit and 64-bit Windows platforms. See <http://msdn.microsoft.com/en-us/library/tcxf1dw6.aspx>.
85791. Implements RE. Currently only needed for death tests.
85792. Returns true if pathname describes an absolute path.
85793. Reserves enough bytes to hold the regular expression used for a full match: we need space to prepend a '^', append a '\$', and terminate the string with '\0'.
85794. Reads and returns a 32-bit integer stored in the environment variable corresponding to the given flag; if it isn't set or doesn't represent a valid 32-bit integer, returns default_value.
85795. Does not own.
85796. We take "yes", "true", "t", and "1" as meaning "yes". If the value is neither one of these nor "auto", we treat it as "no" to be conservative.
85797. Convenience wrapper around EscapeXml when str is not an attribute value.
85798. Both help flag and unrecognized Google Test flags (excluding internal ones) trigger help display.
85799. Returns true if exit_status describes a process that was terminated by a signal, or exited normally with a nonzero exit code.
85800. A predicate that checks the key of a TestProperty against a known key. TestPropertyKeyIs is copyable.
85801. Gets the number of successful test cases.
85802. **comment:** TODO(wan): do not print the value of an expression if it's already a literal.
label: code-design
85803. GTEST_OS_WINDOWS_MOBILE || GTEST_OS_SYMBIAN || GTEST_OS_ZOS The '!= 0' comparison is necessary to satisfy MSVC 7.1.
85804. GTEST_OS_WINDOWS
85805. Convenience function for accessing the global UnitTest implementation object.
85806. Parses a string for an Int32 flag, in the form of "--flag=value". On success, stores the value of the flag in *value, and returns true. On failure, returns false without changing *value.
85807. The name of the file in which the death test is located.
85808. Protects access to GetThreadMapLocked() and its return value.
85809. Owned.
85810. Creates a Test object.
85811. On z/OS we additionally need strings.h for strcasecmp. NOLINT
85812. STL container utilities.
85813. **comment:** Returns the TestCase object for the test that's currently running, or NULL if no test is running.
label: test
85814. GTEST_HAS_STREAM_REDIRECTION
85815. Tells UnitTest where to store test result.
85816. A test is disabled if test case name or test name matches kDisableTestFilter.
85817. Reserves enough bytes to hold the regular expression used for a full match.
85818. Checks whether sharding is enabled by examining the relevant environment variable values. If the variables are present, but inconsistent (e.g., shard_index >= total_shards), prints an error and exits. If in_subprocess_for_death_test, sharding is disabled because it must only be applied to the original test process. Otherwise, we could filter out death tests we intended to execute.
85819. namespace internal
85820. Registers and returns a global test environment. When a test program is run, all global test environments will be set-up in the order they were registered. After all tests in the program have finished, all global test environments will be torn-down in the *reverse* order they were registered. The UnitTest object takes ownership of the given environment. We don't protect this under mutex_, as we only support calling it from the main thread.
85821. This is the default global test part result reporter used in UnitTestImpl. This class should only be used by UnitTestImpl.
85822. **comment:** Returns the given string with all characters invalid in XML removed. Currently invalid characters are dropped from the string. An alternative is to replace them with certain characters such as . or ?.
label: code-design
85823. This should be done before calling OnTestIterationStart(), such that a test event listener can see the actual test order in the event.

85824. Event the child process uses to signal the parent that it has acquired the handle to the write end of the pipe. After seeing this event the parent can release its own handles to make sure its ReadFile() calls return when the child terminates.
85825. num_runnable_tests are the number of tests that will run across all shards (i.e., match filter and are not disabled). num_selected_tests are the number of tests to be run on this shard.
85826. We cannot call abort() as it generates a pop-up in debug mode that cannot be suppressed in VC 7.1 or below.
85827. NOLINT NOLINT
85828. Monitors exit from a given thread and notifies those ThreadIdToThreadLocals about thread termination.
85829. This line ensures that gtest.h can be compiled on its own, even when it's fused.
85830. GTEST_OS_WINDOWS && !GTEST_OS_WINDOWS_MOBILE
85831. Fisher-Yates shuffle, from http://en.wikipedia.org/wiki/Fisher-Yates_shuffle
85832. The random number seed used at the beginning of the test run.
85833. **comment:** Gets the singleton UnitTest object. The first time this method is called, a UnitTest object is constructed and returned. Consecutive calls will return the same object. We don't protect this under mutex_ as a user is not supposed to call this before main() starts, from which point on the return value will never change.
label: code-design
85834. Returns the current working directory, or "" if unsuccessful.
85835. The ctor redirects the stream to a temporary file.
85836. To aid user debugging, we also print c's code in decimal, unless it's 0 (in which case c was printed as "\\0", making the code obvious).
85837. Creates a UTF-16 wide string from the given ANSI string, allocating memory using new. The caller is responsible for deleting the return value using delete[]. Returns the wide string, or NULL if the input is NULL.
85838. A stream to capture.
85839. This points to the TestInfo for the currently running test. It changes as Google Test goes through one test after another. When no test is running, this is set to NULL and Google Test stores assertion results in ad_hoc_test_result_. Initially NULL.
85840. Returns a pointer to the current death test factory.
85841. This function implements either IsSubstring() or IsNotSubstring(), depending on the value of the expected_to_be_substring parameter. StringType here can be const char*, const wchar_t*, ::std::string, or ::std::wstring.
85842. Returns true if c is a printable ASCII character. We test the value of c directly instead of calling isprint(), which is buggy on Windows Mobile.
85843. NOLINT
85844. The d'tor is not virtual. DO NOT INHERIT FROM THIS CLASS.
85845. Converts the buffer in a stringstream to an std::string, converting NUL bytes to "\\0" along the way.
85846. Helper function for IsHRESULT{SuccessFailure} predicates
85847. Waits for any of the handles.
85848. Returns true iff the name of test_case matches name_.
85849. Gets the summary of the failure message by omitting the stack trace in it.
85850. The following routines generate an XML representation of a UnitTest object. This is how Google Test concepts map to the DTD: <testsuites name="AllTests"> <-- corresponds to a UnitTest object <testsuite name="testcase-name"> <-- corresponds to a TestCase object <testcase name="test-name"> <-- corresponds to a TestInfo object <failure message="...">>...</failure> <failure message="...">>...</failure> <failure message="...">>...</failure> <-- individual assertion failures </testcase> </testsuite> </testsuites>
85851. **comment:** TODO(wan@google.com): on Windows a network share like \\server\share can be a root directory, although it cannot be the current directory. Handle this properly.
label: code-design
85852. Returns the number of threads running in the process, or 0 to indicate that we cannot detect it.
85853. The environment variable name for the test shard index.
85854. Do nothing.
85855. Somebody else is already initializing the mutex; spin until they are done.
85856. D'tor.
85857. We pass skip_count + 1 to skip this wrapper function in addition to what the user really wants to skip.
85858. When Google Test is built as a framework on MacOS X, the environ variable is unavailable. Apple's documentation (man environ) recommends using _NSGetEnviron() instead.
85859. GTEST_HAS_PARAM_TEST
85860. Repeats forever if the repeat count is negative.
85861. Attempts to parse a string into a positive integer pointed to by the number parameter. Returns true if that is possible. GTEST_HAS_DEATH_TEST implies that we have ::std::string, so we can use it here.
85862. Gets the elapsed time, in milliseconds.
85863. Sets up all environments beforehand.
85864. Windows CE doesn't support FormatMessage.
85865. Handle to the write end of the pipe to the child process.
85866. No - an invalid character was encountered.
85867. How long the test took to run, in milliseconds.
85868. Redirects all logging to stderr in the child process to prevent concurrent writes to the log files. We capture stderr in the parent process and append the child process' output to a log.
85869. GTEST_HAS_DEATH_TEST implies that we have ::std::string, so we can use it here.
85870. Formats a byte as "%02X".
85871. This function should not be called when the condition is false, but we provide a sensible default in case it is.
85872. The c'tor sets this object as the test part result reporter used by Google Test. The 'result' parameter specifies where to report the results.
85873. **comment:** Removes any redundant separators that might be in the pathname. For example, "bar//foo" becomes "bar/foo". Does not eliminate other redundancies that might be in a pathname involving "." or "...". TODO(wan@google.com): handle Windows network shares (e.g. \\server\share).
label: code-design
85874. GTEST_USES_POSIX_REGEX
85875. Note that the above two checks will both fail if either val1 or val2 is NaN, as the IEEE floating-point standard requires that any predicate involving a NaN must return false.
85876. Splits a given string on a given delimiter, populating a given vector with the fields. GTEST_HAS_DEATH_TEST implies that we have ::std::string, so we can use it here.
85877. namespace edit_distance
85878. **comment:** Unshuffles the tests in each test case.
label: test
85879. When a SIGPROF signal is received while fork() or clone() are executing, the process may hang. To avoid this, we ignore SIGPROF here and re-enable it after the call to fork()/clone() is complete.
85880. Adds an "exception thrown" fatal failure to the current test. This function returns its result via an output parameter pointer because VC++ prohibits creation of objects with destructors on stack in functions using __try (see error C2712).
85881. skip_count
85882. Prohibit instantiation.
85883. Google Test should handle a SEH exception if: 1. the user wants it to, AND 2. this is not a breakpoint exception, AND 3. this is not a C++ exception (VC++ implements them via SEH, apparently). SEH exception code for C++ exceptions. (see <http://support.microsoft.com/kb/185294> for more information).
85884. Sets the global test part result reporter.
85885. Provides access to the event listener list.
85886. Implements the TestPartResultReporterInterface. The implementation just delegates to the current global test part result reporter of *unit_test_
85887. Sets the OS stack trace getter. Does nothing if the input and the current OS stack trace getter are the same; otherwise, deletes the old getter and makes the input the current getter.

85888. Whole string is a positive filter
85889. We are done. We don't want this hunk.
85890. Outside the lock, let the destructor for 'value_holders' deallocate the ThreadLocalValueHolderBases.
85891. Returns true iff regular expression re matches a substring of str (including str itself).
85892. This string is inserted in place of stack frames that are part of Google Test's implementation.
85893. A set of macros for testing Google Test assertions or code that's expected to generate Google Test fatal failures. It verifies that the given statement will cause exactly one fatal Google Test failure with 'substr' being part of the failure message. There are two different versions of this macro. EXPECT_FATAL_FAILURE only affects and considers failures generated in the current thread and EXPECT_FATAL_FAILURE_ON_ALL_THREADS does the same but for all threads. The verification of the assertion is done correctly even when the statement throws an exception or aborts the current function. Known restrictions: - 'statement' cannot reference local non-static variables or non-static members of the current object. - 'statement' cannot return a value. - You cannot stream a failure message to this macro. Note that even though the implementations of the following two macros are much alike, we cannot refactor them to use a common helper macro, due to some peculiarity in how the preprocessor works. The AcceptsMacroThatExpandsToUnprotectedComma test in gtest_unittest.cc will fail to compile if we do that.
85894. Creates an empty UnitTest.
85895. Organizes the bytes into groups of 2 for easy parsing by human.
85896. namespace testing Copyright 2008 Google Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan)
85897. Is the first thing in regex an escape sequence?
85898. This class implements the TestEvent Listener interface. Class PrettyUnitTestResultPrinter is copyable.
85899. Constructor. TestPropertyKeyIs has NO default constructor.
85900. Implements the helper function for {ASSERT|EXPECT}_LE with int or enum arguments.
85901. class TestInfo
85902. True iff we are in a subprocess for running a thread-safe-style death test.
85903. **comment:** Returns an indented copy of stderr output for a death test. This makes distinguishing death test output lines from regular log lines much easier.
label: code-design
85904. Returns the ANSI color code for the given color. COLOR_DEFAULT is an invalid input.
85905. Returns the value of GTEST_FLAG(catch_exceptions) at the moment UnitTest::Run() starts.
85906. **comment:** TODO(kenton@google.com): Use autoconf to detect availability of gettimeofday().
label: requirement
85907. Gets the text streamed to this object so far as an std::string. Each '\0' character in the buffer is replaced with "\\0".
85908. Prints the filter if it's not *. This reminds the user that some tests may be skipped.
85909. Indicates that this translation unit is part of Google Test's implementation. It must come before gtest-internal-inl.h is included, or there will be a compiler error. This trick exists to prevent the accidental inclusion of gtest-internal-inl.h in the user's code.
85910. Some POSIX platforms expect you to declare environ. extern "C" makes it reside in the global namespace.
85911. Parameter to ThreadMainStatic Default creation flags. Need a valid pointer for the call to work under Win98.
85912. Give the watcher thread the same priority as ours to avoid being blocked by it.
85913. Starts capturing stdout.
85914. Returns true iff the test fatally failed.
85915. Performs an in-place shuffle of the vector's elements.
85916. Pops a trace from the per-thread Google Test trace stack.
85917. Returns true iff the user-specified filter matches the test case name and the test name.
85918. See the comment in NoExecDeathTest::AssumeRole for why the next line is necessary.
85919. namespace testing Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: keith.ray@gmail.com (Keith Ray)
85920. OS selector
85921. namespace internal namespace testing Copyright 2007, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan)
85922. Parses the environment variable var as an Int32. If it is unset, returns default_val. If it is not an Int32, prints an error and aborts.
85923. **comment:** TODO(vladl@google.com): Factor the search functionality into Vector::Find.
label: code-design
85924. Replaces NUL with "\\0";
85925. Normally, a user only writes assertions inside a TEST or TEST_F, or inside a function called by a TEST or TEST_F. Since Google Test keeps track of which test is current running, it can associate such an assertion with the test it belongs to. If an assertion is encountered when no TEST or TEST_F is running, Google Test attributes the assertion result to an imaginary "ad hoc" test, and records the result in ad_hoc_test_result_.
85926. 10xxxxxx 10xxxxxx 1110xxxx code_point <= kMaxCodePoint4
85927. Returns true iff the unit test passed (i.e. all test cases passed).
85928. Streams an XML CDATA section, escaping invalid CDATA sequences as needed.
85929. The string representation of the values received in EqFailure() are already escaped. Split them on escaped "\n" boundaries. Leave all other escaped characters the same.

85930. To allow both IPv4 and IPv6 addresses.

85931. Initializes Google Test. This must be called before calling RUN_ALL_TESTS(). In particular, it parses a command line for the flags that Google Test recognizes. Whenever a Google Test flag is seen, it is removed from argv, and *argc is decremented. No value is returned. Instead, the Google Test flag variables are updated. Calling the function for the second time has no user-visible effect.

85932. Utility class for accumulating command-line arguments.

85933. Converts the given wide string to a narrow string using the UTF-8 encoding, and streams the result to this Message object.

85934. Returns the number of the test properties.

85935. Initializes owner_thread_id_ and critical_section_ in static mutexes.

85936. Returns the global test part result reporter.

85937. cpplint thinks that the header is already included, so we want to silence it. NOLINT

85938. All of these virtual functions are inherited from DeathTest.

85939. create the file with a single "0" character in it. I/O errors are ignored as there's nothing better we can do and we don't want to fail the test because of this.

85940. Prints a TestPartResult.

85941. The constructor of SingleFailureChecker remembers where to look up test part results, what type of failure we expect, and what substring the failure message should contain.

85942. There are two different ways to register your own TestPartResultReporter. You can register your own repoter to listen either only for test results from the current thread or for results from all threads. By default, each per-thread test result repoter just passes a new TestPartResult to the global test result reporter, which registers the test part result for the currently running test.

85943. Returns true if "name" matches the ':' separated list of glob-style filters in "filter".

85944. **comment:** Returns the TestResult for the ad hoc test.
label: test

85945. This defines a member that forwards the call to all listeners in reverse order.

85946. Tells UnitTest to stop associating assertion results to this test.

85947. The AssumeRole process for a fork-and-run death test. It implements a straightforward fork, with a simple pipe to transmit the status byte.

85948. Generates a random number from [0, range), using a Linear Congruential Generator (LCG). Crashes if 'range' is 0 or greater than kMaxRange.

85949. Pops the info pushed by the c'tor.

85950. Aborts if the parsing failed.

85951. Returns the name of the environment variable corresponding to the given flag. For example, FlagToEnvVar("foo") will return "GTEST_FOO" in the open-source version.

85952. namespace testing

85953. This is the default per thread test part result reporter used in UnitTestImpl. This class should only be used by UnitTestImpl.

85954. **comment:** Delegates to PrintBytesInObjectToImpl() to print the bytes in the given object. The delegation simplifies the implementation, which uses the << operator and thus is easier done outside of the ::testing::internal namespace, which contains a << operator that sometimes conflicts with the one in STL.
label: code-design

85955. The c'tor.

85956. The helper function for {ASSERT|EXPECT}_STREQ.

85957. spawn is a system call.

85958. **comment:** We don't call OutputDebugString*() on Windows Mobile, as printing to stdout is done by OutputDebugString() there already - we don't want the same message printed twice.
label: code-design

85959. Returns true if pathname describes a root directory. (Windows has one root directory per disk drive.)

85960. Declares the flags. We don't want the users to modify this flag in the code, but want Google Test's own unit tests to be able to access it. Therefore we declare it here as opposed to in gtest.h.

85961. The two possible mocking modes of this object.

85962. Performs initialization dependent upon flag values obtained in ParseGoogleTestFlagsOnly. Is called from InitGoogleTest after the call to ParseGoogleTestFlagsOnly. In case a user neglects to call InitGoogleTest this function is also called from RunAllTests. Since this function can be called more than once, it has to be idempotent.

85963. The c'tor sets this object as the test part result reporter used by Google Test. The 'result' parameter specifies where to report the results. Intercepts only failures from the current thread.

85964. Message assignment, for assertion streaming support.

85965. **comment:** Adds a test part result to the list.
label: test

85966. Big enough for the largest valid code point.

85967. If the test program runs in Visual Studio or a debugger, the following statements add the test part result message to the Output window such that the user can double-click on it to jump to the corresponding source code location; otherwise they do nothing.

85968. Shuffles test cases and tests if requested.

85969. Initializes event listeners performing XML output as specified by UnitTestOptions. Must not be called before InitGoogleTest.

85970. _WIN32_WCE

85971. Tells the user how big the object is.

85972. Runs the test and updates the test result.

85973. Escapes '=', '&', '%', and 'n' characters in str as "%xx".

85974. Flush the log buffers since the log streams are shared with the child.

85975. Finds and returns a TestCase with the given name. If one doesn't exist, creates one and returns it. It's the CALLER'S RESPONSIBILITY to ensure that this function is only called WHEN THE TESTS ARE NOT SHUFFLED. Arguments: test_case_name: name of the test case type_param: the name of the test case's type parameter, or NULL if this is not a typed or a type-parameterized test case. set_up_tc: pointer to the function that sets up the test case tear_down_tc: pointer to the function that tears down the test case

85976. Constructor.

85977. ParameterizedTestRegistry object used to register value-parameterized tests.

85978. Returns true iff "\c" is a supported escape sequence.

85979. ExitedWithCode function-call operator.

85980. The d'tor restores the previous test part result reporter.

85981. Returns if no more pattern can be found.

85982. Sends a string to the socket.

85983. Returns a copy of the FilePath with the case-insensitive extension removed. Example: FilePath("dir/file.exe").RemoveExtension("EXE") returns FilePath("dir/file"). If a case-insensitive extension is not found, returns a copy of the original FilePath.

85984. read_fd_ is expected to be closed and cleared by a derived class.

85985. Starts capturing an output stream (stdout/stderr).

85986. Copyright 2007, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan) Utilities for testing Google Test itself and code that uses Google Test (e.g. frameworks built on top of Google Test).

85987. Holds the thread id and thread handle that we pass from StartWatcherThreadFor to WatcherThreadFunc.

85988. No. Appends to the end of the list.

85989. Closes the socket.

85990. A Unicode code-point can have upto 21 bits, and is encoded in UTF-8 like this: Code-point length Encoding 0 - 7 bits 0xxxxxx 8 - 11 bits 110xxxxx 10xxxxxx 12 - 16 bits 1110xxxx 10xxxxxx 10xxxxxx 17 - 21 bits 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

85991. class StreamingListener

85992. Makes sure InitGoogleTest() was called.

85993. Reset count when a non match is found.

85994. Formats an int value as "%02d".

85995. Returns the absolute path of the requested output file, or the default (test_detail.xml in the original working directory) if none was explicitly specified.

85996. Adds a TestProperty to the current TestResult object when invoked in a context of a test or a test case, or to the global property set. If the result already contains a property with the same key, the value will be updated.

85997. **comment:** The main function for a threadsafe-style death test child process. This function is called in a clone()-ed process and thus must avoid any potentially unsafe operations like malloc or libc functions.

label: code-design

85998. Returns true iff the test part fatally failed.

85999. GTEST_OS_NACL

86000. Prints a wide or narrow char c as a character literal without the quotes, escaping it when necessary; returns how c was formatted. The template argument UnsignedChar is the unsigned version of Char, which is the type of c.

86001. Creates a concrete DeathTest-derived class that depends on the --gtest_death_test_style flag, and sets the pointer pointed to by the "test" argument to its address. If the test should be skipped, sets that pointer to NULL. Returns true, unless the flag is set to an invalid value.

86002. Windows CE doesn't have a current directory, so we just return something reasonable.

86003. GTEST_IMPLEMENTATION_

86004. 10xxxxxx 110xxxxx

86005. Restores the test cases and tests to their order before the first shuffle.

86006. Called after an assertion failure.

86007. Indicates whether RegisterParameterizedTests() has been called already.

86008. Keeps reading the file until we cannot read further or the pre-determined file size is reached.

86009. Runs the tests only if there was no fatal failure during global set-up.

86010. Makes a successful assertion result.

86011. Implements the helper function for {ASSERT|EXPECT}_NE with int or enum arguments.

86012. Constructs a TestInfo object. It assumes ownership of the test factory object.

86013. We are not on Windows.

86014. Google Test requires all tests in the same test case to use the same test fixture class. This function checks if the current test has the same fixture class as the first test in the current test case. If yes, it returns true; otherwise it generates a Google Test failure and returns false.

86015. **comment:** Gets the number of disabled tests in this test case.

label: test

86016. KilledBySignal constructor.

86017. Split --gtest_filter at ':', if there is one, to separate into positive filter and negative filter portions

86018. The maximum code-point a one-byte UTF-8 sequence can represent.

86019. Updates the critical_section_init_phase_ to 2 to signal initialization complete.

86020. Returns the TestResult containing information on test failures and properties logged outside of individual test cases.

86021. Either `:` or `^` marks the end of the pattern.

86022. Using DebugBreak on Windows allows gtest to still break into a debugger when a failure happens and both the --gtest_break_on_failure and the --gtest_catch_exceptions flags are specified.

86023. Get the number of tests in this test case that should run.

86024. Returns true iff the current test has a fatal failure.

86025. Tells the unit test event listeners that the tests are about to start.

86026. The list of reserved attributes used in the <testsuite> element of XML output.

86027. Restores the test order to before the first shuffle.

86028. Disable Microsoft deprecation warnings for POSIX functions called from this class (creat, dup, dup2, and close)

86029. Stops capturing stderr and returns the captured string.

86030. Skips the flag name.

86031. Controls whether events will be forwarded by the repeater to the listeners in the list.

86032. Returns true iff the given atom (specified by escaped and pattern) matches ch. The result is undefined if the atom is invalid.

86033. **comment:** On Windows the TERM variable is usually not set, but the console there does support colors.

label: code-design

86034. Returns the string after `=

86035. We print the value parameter on a single line to make the output easy to parse by a program.

86036. Class ScopedTrace

86037. No stack trace, either.

86038. Constructs a ForkingDeathTest.

86039. The environment variable name for the total number of test shards.

86040. Death test children can be terminated with _abort(). On Windows, _abort() can show a dialog with a warning message. This forces the abort message to go to stderr instead.

86041. Create a list of diff hunks in Unified diff format. Each hunk has a header generated by PrintHeader above plus a body with lines prefixed with '' for no change, '-' for deletion and '+' for addition. 'context' represents the desired unchanged prefix/suffix around the diff. If two hunks are close enough that their contexts overlap, then they are joined into one hunk.

86042. Both TEST and TEST_F appear in same test case, which is incorrect. Tell the user how to fix this.

86043. Retuned process handle is not inheritable. Retuned thread handle is not inheritable. Child inherits all inheritable handles (for write_handle_). Default creation flags. Inherit the parent's environment.

86044. Produces a string representing the test properties in a result as space delimited XML attributes based on the property key="value" pairs.

86045. Prints a ::wstring object.

86046. Reconstruct the best path. We do it in reverse order.

86047. **comment:** Helper function used by ValidateRegex() to format error messages.

label: code-design

86048. Comparing two pointers of which only one is NULL is undefined.

86049. Prints the given wide C string to the ostream.

86050. The read() here blocks until data is available (signifying the failure of the death test) or until the pipe is closed (signifying its success), so it's okay to call this in the parent before the child process has exited.

86051. Returns the TestPartResult at the given index (0-based).

86052. Returns EXCEPTION_EXECUTE_HANDLER if Google Test should handle the given SEH exception, or EXCEPTION_CONTINUE_SEARCH otherwise. This function is useful as an __except condition.

86053. The helper function for {ASSERT|EXPECT}_STRNE.

86054. Checks whether sharding is enabled by examining the relevant environment variable values. If the variables are present, but inconsistent (i.e., shard_index >= total_shards), prints an error and exits. If in_subprocess_for_death_test, sharding is disabled because it must only be applied to the original test process. Otherwise,

we could filter out death tests we intended to execute.

86055. Flushes the buffers and, if severity is GTEST_FATAL, aborts the program.

86056. Helper function for *_STRNE on wide strings.

86057. We can safely call chdir() as it's a direct system call.

86058. Find the first line to include in the hunk.

86059. The maximum code-point a four-byte UTF-8 sequence can represent.

86060. "\$" only matches the end of a string. Note that regex being valid guarantees that there's nothing after "\$" in it.

86061. **comment:** This function is a friend of UnitTest and as such has access to AddTestPartResult.

label: code-design

86062. On non-Windows platforms, we rely on the TERM variable.

86063. Formats the count of test cases.

86064. Gets the number of successful tests.

86065. Recreates the pipe and event handles from the provided parameters, signals the event, and returns a file descriptor wrapped around the pipe handle. This function is called in the child process only.

86066. The UnitTest object that owns this implementation object.

86067. Tears down all environments in reverse order afterwards.

86068. The event will automatically reset to non-signaled state. The initial state is non-signaled. The even is unnamed.

86069. Produces a string representing the test properties in a result as space delimited XML attributes based on the property key="value" pairs. When the std::string is not empty, it includes a space at the beginning, to delimit this attribute from prior attributes.

86070. Destroys a TestInfo object.

86071. The OS stack trace getter. Will be deleted when the UnitTest object is destructed. By default, an OsStackTraceGetter is used, but the user can set this field to use a custom getter if that is desired.

86072. Gets the number of failed test cases.

86073. This points to the TestCase for the currently running test. It changes as Google Test goes through one test case after another. When no test is running, this is set to NULL and Google Test stores assertion results in ad_hoc_test_result_. Initially NULL.

86074. A helper class for implementing EXPECT_FATAL_FAILURE() and EXPECT_NONFATAL_FAILURE(). Its destructor verifies that the given TestPartResultArray contains exactly one failure that has the given type and contains the given substring. If that's not the case, a non-fatal failure will be generated.

86075. The list of listeners that receive events.

86076. Prints the bytes in the given value to the given ostream.

86077. Will be overridden by the flag before first use. Will be reseeded before first use.

86078. ExitedWithCode constructor.

86079. Returns the size (in bytes) of a file.

86080. Windows CE doesn't have a current directory. You should not use the current directory in tests on Windows CE, but this at least provides a reasonable fallback.

86081. No info about the source file where the exception occurred. We have no info on which line caused the exception.

86082. Is the first test defined using TEST?

86083. Indicates that this translation unit is part of Google Test's implementation. It must come before gtest-internal-inl.h is included, or there will be a compiler error. This trick is to prevent a user from accidentally including gtest-internal-inl.h in his code.

86084. Sends the given message and a newline to the socket.

86085. We need to flush the stream buffers into the console before each SetConsoleTextAttribute call lest it affect the text that is already printed but has not yet reached the console.

86086. The default test part result reporters.

86087. Adds an "exception thrown" fatal failure to the current test.

86088. Resets the terminal to default. GTEST_OS_WINDOWS && !GTEST_OS_WINDOWS_MOBILE

86089. Creates an empty TestResult.

86090. The mutex is already initialized and ready for use.

86091. Note: Android applications are expected to call the framework's Context.getExternalStorageDirectory() method through JNI to get the location of the world-writable SD Card directory. However, this requires a Context handle, which cannot be retrieved globally from native code. Doing so also precludes running the code as part of a regular standalone executable, which doesn't run in a Dalvik process (e.g. when running it through 'adb shell'). The location /sdcard is directly accessible from native code and is the only location (unofficially) supported by the Android team. It's generally a symlink to the real SD Card mount point which can be /mnt/sdcard, /mnt/sdcard0, /system/media/sdcard, or other OEM-customized locations. Never rely on these, and always use /sdcard.

86092. Android, Mac OS X and Cygwin don't define wcscasecmp. Other unknown OSes may not define it either.

86093. GTEST_HAS_EXCEPTIONS

86094. The d'tor restores the test part result reporter used by Google Test before.

86095. Either the user wants Google Test to catch exceptions thrown by the tests or this is executing in the context of death test child process. In either case the user does not want to see pop-up dialogs about crashes - they are expected.

86096. **comment:** Shuffles all test cases, and the tests within each test case, making sure that death tests are still run first.

label: test

86097. Gets the i-th test case among all the test cases. i can range from 0 to total_test_case_count() - 1. If i is not in that range, returns NULL.

86098. (_MSC_VER == 1310 && !defined(_DEBUG)) || defined(__BORLANDC__)

86099. If critical_section_init_phase_ was 0 before the exchange, we are the first to test it and need to perform the initialization.

86100. **comment:** On a POSIX system, this function may be called from a threadsafe-style death test child process, which operates on a very small stack. Use the heap for any additional non-minuscule memory requirements.

label: code-design

86101. Not an escape sequence.

86102. GTestIsInitialized() returns true iff the user has initialized Google Test. Useful for catching the user mistake of not initializing Google Test before calling RUN_ALL_TESTS(). A user must call testing::InitGoogleTest() to initialize Google Test. g_init_gtest_count is set to the number of times InitGoogleTest() has been called. We don't protect this variable under a mutex as it is only accessed in the main thread.

86103. Deletes every TestCase.

86104. The maximum code-point a two-byte UTF-8 sequence can represent.

86105. substr_expr

86106. Returns the character attribute for the given color.

86107. We want to preserve failures generated by ad-hoc test assertions executed before RUN_ALL_TESTS().

86108. The line number on which the death test is located.

86109. A per-thread stack of traces created by the SCOPED_TRACE() macro.

86110. Gets the number of successful tests in this test case.

86111. Returns true iff the unit test failed (i.e. some test case failed or something outside of all tests failed).

86112. Reads and returns the Boolean environment variable corresponding to the given flag; if it's not set, returns default_value. The value is considered true iff it's not "0".

86113. The value of GetTestId() as seen from within the Google Test library. This is solely for testing GetTestId().

86114. The textual content of the code this object is testing. This class doesn't own this string and should not attempt to delete it.

86115. 0xxxxxxxx

86116. Runs each test case if there is at least one test to run.

86117. The constructor remembers the arguments.

86118. Swaps two AssertionResults.

86119. Compares two C strings, ignoring case. Returns true iff they have the same content. Unlike strcasecmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL C string, including the empty string.

86120. Functions for processing the gtest_output flag.

86121. Sends a string and a newline to the socket.

86122. **comment:** TODO(yukawa): Consider to use _beginthreadex instead.
label: code-design
86123. **comment:** Two utility routines that together determine the direction the stack grows. This could be accomplished more elegantly by a single recursive function, but we want to guard against the unlikely possibility of a smart compiler optimizing the recursion away. GTEST_NO_INLINE_ is required to prevent GCC 4.6 from inlining StackLowerThanAddress into StackGrowsDown, which then doesn't give correct answer.
label: code-design
86124. Clean up the ThreadIdToThreadLocals data structure while holding the lock, but defer the destruction of the ThreadLocalValueHolderBases.
86125. Returns true iff the test has a non-fatal failure.
86126. True iff ?, *, or + can follow the previous atom.
86127. Sets the default_xml_generator attribute to the provided listener. The listener is also added to the listener list and previous default_xml_generator is removed from it and deleted. The listener can also be NULL in which case it will not be added to the list. Does nothing if the previous and the current listener objects are the same.
86128. Adds a TestProperty to the current TestResult object when invoked from inside a test, to current TestCase's ad_hoc_test_result_ when invoked from SetUpTestCase or TearDownTestCase, or to the global property set when invoked elsewhere. If the result already contains a property with the same key, the value will be updated.
86129. **comment:** Helper functions for naming files in a directory for xml output.
label: code-design
86130. We are on Windows CE.
86131. Abstract base class for writing strings to a socket.
86132. Returns the assertion's negation. Used with EXPECT/ASSERT_FALSE.
86133. Intercepts only thread local failures. Intercepts all failures.
86134. Treat '-test1' as the same as '*-test1'
86135. Found a match. Consume it.
86136. StringType here can be either ::std::string or ::std::wstring.
86137. Returns success if val1 is less than val2,
86138. Create the directory so that path exists. Returns true if successful or if the directory already exists; returns false if unable to create the directory for any reason, including if the parent directory does not exist. Not named ".CreateDirectory" because that's a macro on Windows.
86139. Controls whether events will be forwarded to listeners_. Set to false in death test child processes.
86140. Requested privileges ignored since DUPLICATE_SAME_ACCESS is used. Request non-inheritable handler.
86141. Creates a new XmlUnitTestResultPrinter.
86142. This overloaded version can be used in Windows programs compiled in UNICODE mode.
86143. Adds a test property to the list. If a property with the same key as the supplied property is already represented, the value of this test_property replaces the old value for that key.
86144. **comment:** How the death test concluded.
label: test
86145. These two overloads allow streaming a wide C string to a Message using the UTF-8 encoding.
86146. Clears the test part results.
86147. Given directory = "dir", base_name = "test", number = 0, extension = "xml", returns "dir/test.xml". If number is greater than zero (e.g., 12), returns "dir/test_12.xml". On Windows platform, uses \ as the separator rather than /.
86148. Populate for empty right.
86149. protects all internal state
86150. Helper class to convert string into ids with deduplication.
86151. Runs the given method and handles SEH exceptions it throws, when SEH is supported; returns the 0-value for type Result in case of an SEH exception. (Microsoft compilers cannot handle SEH and C++ exceptions in the same function. Therefore, we provide a separate wrapper function for handling SEH exceptions.)
86152. This method expands all parameterized tests registered with macros TEST_P and INSTANTIATE_TEST_CASE_P into regular tests and registers those. This will be done just once during the program runtime.
86153. Difference between 1970-01-01 and 1601-01-01 in milliseconds. <http://analogous.blogspot.com/2005/04/epoch.html>
86154. g_help_flag is true iff the --help flag or an equivalent form is specified on the command line.
86155. Normalizes the actual seed to range [1, kMaxRandomSeed] such that it's easy to type.
86156. We will run the test only if SetUp() was successful.
86157. Registers parameterized tests. This makes parameterized tests available to the UnitTest reflection API without running RUN_ALL_TESTS.
86158. Gets the number of all test parts. This is the sum of the number of successful test parts and the number of failed test parts.
86159. Constructs a failure message for Boolean assertions such as EXPECT_TRUE.
86160. Descriptor to the child's write end of the pipe to the parent process. It is always -1 in the parent process. The parent keeps its end of the pipe in read_fd_.
86161. Does nothing if the current thread holds the mutex. Otherwise, crashes with high probability.
86162. **comment:** A macro for testing Google Test assertions or code that's expected to generate Google Test non-fatal failures. It asserts that the given statement will cause exactly one non-fatal Google Test failure with 'substr' being part of the failure message. There are two different versions of this macro. EXPECT_NONFATAL_FAILURE only affects and considers failures generated in the current thread and EXPECT_NONFATAL_FAILURE_ON_ALL_THREADS does the same but for all threads. 'statement' is allowed to reference local variables and members of the current object. The verification of the assertion is done correctly even when the statement throws an exception or aborts the current function. Known restrictions: - You cannot stream a failure message to this macro. Note that even though the implementations of the following two macros are much alike, we cannot refactor them to use a common helper macro, due to some peculiarity in how the preprocessor works. If we do that, the code won't compile when the user gives EXPECT_NONFATAL_FAILURE() a statement that contains a macro that expands to code containing an unprotected comma. The AcceptsMacroThatExpandsToUnprotectedComma test in gtest_unittest.cc catches that. For the same reason, we have to write if (::testing::internal::AlwaysTrue() { statement; } instead of GTEST_SUPPRESS_UNREACHABLE_CODE_WARNING_BELOW_(statement) to avoid an MSVC warning on unreachable code.
label: code-design
86163. Matches any single character.
86164. Picks a new random seed for each iteration.
86165. Application pathname gotten in InitGoogleTest.
86166. **comment:** Implements the TestPartResultReporterInterface. Reports the test part result in the current test.
label: test
86167. Event forwarding to the listeners of event listener API must be shut down in death test subprocesses.
86168. Is this test defined using TEST?
86169. Fail fast if the given string does not begin with a digit; this bypasses strtoXXX's "optional leading whitespace and plus or minus sign" semantics, which are undesirable here.
86170. We don't protect writing to owner_thread_id_ here, as it's the caller's responsibility to ensure that the current thread holds the mutex when this is called.
86171. Initializes the event listener performing XML output as specified by UnitTestOptions. Must not be called before InitGoogleTest.
86172. Runs all tests in this UnitTest object and prints the result. Returns 0 if successful, or 1 otherwise. We don't protect this under mutex_, as we only support calling it from the main thread.
86173. If, however, the last element in the array is not '\0', e.g. const char kFoo[] = { 'f', 'o', 'o' }; we must print the entire array. We also print a message to indicate that the array is not NUL-terminated.
86174. Returns the list of event listeners that can be used to track events inside Google Test.
86175. Clear the following flags: pop-up window, core dump.
86176. We need to execute the test program in the same environment where it was originally invoked. Therefore we change to the original working directory first.
86177. Initializes event listeners for streaming test results in string form. Must not be called before InitGoogleTest.
86178. Resets the index of each test case.
86179. We know that the atom matches each of the first i characters in str.
86180. Print at most this many characters for each type/value parameter.

86181. We save the stack frame below the frame that calls user code. We do this because the address of the frame immediately below the user code changes between the call to UponLeavingGTest() and any calls to CurrentStackTrace() from within the user code.

86182. Matches any string (possibly empty) of characters.

86183. Applies a function/functor to each element in the container.

86184. Prints a TestPartResult to an std::string.

86185. Gets the number of tests to be printed in the XML report.

86186. The c'tor saves the values of all Google Test flags.

86187. Depending on the value of a char (or wchar_t), we print it in one of three formats: - as is if it's a printable ASCII (e.g. 'a', '2', ' '), - as a hexadecimal escape sequence (e.g. '\x7F'), or - as a special escape sequence (e.g. '\r', '\n').

86188. cpplint thinks that the header is already included, so we want to silence it. NOLINT NOLINT

86189. The child process will share the standard handles with the parent.

86190. **comment:** Can we find a TestCase with the given name?
label: test

86191. Starts capturing stderr.

86192. Populate for empty left.

86193. KilledBySignal function-call operator.

86194. Removes the given event listener from the list and returns it. It then becomes the caller's responsibility to delete the listener. Returns NULL if the listener is not found in the list.

86195. We need to pass a valid thread ID pointer into CreateThread for it to work correctly under Win98.

86196. BiggestConvertible is the largest integer type that system-provided string-to-number conversion routines can return.

86197. **comment:** TODO(vladl@google.com): Convert this to compile time assertion when it is available.
label: code-design

86198. Gets the result and clears it.

86199. results_expr

86200. GTEST_HAS_SEH

86201. We can safely call execve() as it's a direct system call. We cannot use execvp() as it's a libc function and thus potentially unsafe. Since execve() doesn't search the PATH, the user must invoke the test program via a valid path that contains at least one path separator.

86202. This must be called *after* FilterTests() has been called.

86203. Returns the number of TestPartResult objects in the array.

86204. GTEST_HAS_STD_WSTRING || GTEST_HAS_GLOBAL_WSTRING

86205. Constants.

86206. A predicate that checks the test name of a TestInfo against a known value. This is used for implementation of the TestCase class only. We put it in the anonymous namespace to prevent polluting the outer namespace. TestNameIs is copyable.

86207. Prints a wide or narrow character c and its code. '\0' is printed as "\\0", other unprintable characters are also properly escaped using the standard C++ escape sequence. The template argument UnsignedChar is the unsigned version of Char, which is the type of c.

86208. **comment:** We put these helper functions in the internal namespace as IBM's xlC compiler rejects the code if they were declared static.
label: code-design

86209. Transfers ownership.

86210. Called after the unit test ends.

86211. A replacement for CHECK that calls DeathTestAbort if the assertion fails.

86212. Returns a Boolean value indicating whether the caller is currently executing in the context of the death test child process. Tools such as Valgrind heap checkers may need this to modify their behavior in death tests. IMPORTANT: This is an internal utility. Using it may break the implementation of death tests. User code MUST NOT use it.

86213. Advance indices, depending on edit type.

86214. There is no next edit or it is too far away.

86215. Returns the first valid random seed after 'seed'. The behavior is undefined if 'seed' is invalid. The seed after kMaxRandomSeed is considered to be 1.

86216. Creates a TestCase with the given name. Arguments: name: name of the test case a_type_param: the name of the test case's type parameter, or NULL if this is not a typed or a type-parameterized test case. set_up_tc: pointer to the function that sets up the test case tear_down_tc: pointer to the function that tears down the test case

86217. Returns the current OS stack trace as an std::string. Parameters: max_depth - the maximum number of stack frames to be included in the trace. skip_count - the number of top frames to be skipped; doesn't count against max_depth.

86218. TestResult appropriate for property recording.

86219. **comment:** GTEST_IS_THREADSsafe && GTEST_OS_WINDOWS
label: test

86220. Flag characters for reporting a death test that did not die.

86221. Gets the current text color.

86222. If the object size is bigger than kThreshold, we'll have to omit some details by printing only the first and the last kChunkSize bytes. TODO(wan): let the user control the threshold using a flag.

86223. unit_test

86224. GTEST_OS_WINDOWS_MOBILE

86225. Parses a string for a bool flag, in the form of either "--flag=value" or "--flag". In the former case, the value is taken as true as long as it does not start with '0', 'f', or 'F'. In the latter case, the value is taken as true. On success, stores the value of the flag in *value, and returns true. On failure, returns false without changing *value.

86226. Verifies that registered_tests match the test names in defined_test_names_; returns registered_tests if successful, or aborts the program otherwise.

86227. Provides a level of indirection for the test case list to allow easy shuffling and restoring the test case order. The i-th element of this vector is the index of the i-th test case in the shuffled order.

86228. Returns a pointer to the parsed --gtest_internal_run_death_test flag, or NULL if that flag was not specified. This information is useful only in a death test child process. Must not be called before a call to InitGoogleTest.

86229. Default security. Default stack size.

86230. socket file descriptor

86231. For strtoll/_strtoul64/malloc/free. For memmove.

86232. TestResult contains some private methods that should be hidden from Google Test user but are required for testing. This class allows our tests to access them. This class is supplied only for the purpose of testing Google Test's own constructs. Do not use it in user tests, either directly or indirectly.

86233. Returns the given string with all characters invalid in XML removed.

86234. Prints the names of the tests matching the user-specified filter flag.

86235. The environment variable is not set.

86236. namespace testing Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan), vladl@google.com (Vlad Losev) This file implements death tests.

86237. The maximum code-point a three-byte UTF-8 sequence can represent.
86238. GTEST_OS_LINUX_ANDROID
86239. Chops off the n lowest bits from a bit pattern. Returns the n lowest bits. As a side effect, the original bit pattern will be shifted to the right by n bits.
86240. Appends the user-supplied message to the Google-Test-generated message.
86241. Returns the i-th test part result among all the results. i can range from 0 to total_part_count() - 1. If i is not in that range, aborts the program.
86242. The flag must start with "--" followed by GTEST_FLAG_PREFIX_.
86243. class TestCase
86244. Pushes the given source file location and message onto a per-thread trace stack maintained by Google Test.
86245. Convenience wrapper around EscapeXml when str is an attribute value.
86246. Runs every test in this TestCase.
86247. 10xxxxxx 10xxxxxx 10xxxxxx 11110xxx
86248. Called at the start of streaming to notify the receiver what protocol we are using.
86249. Utilities
86250. Do not run any test if the --help flag was specified.
86251. This generic version is used when k is 0.
86252. class CartesianProductGenerator3
86253. Type and function utilities for implementing parameterized tests.
86254. Generates a random number from [0, range). Crashes if 'range' is 0 or greater than kMaxRange.
86255. A test case, which consists of a vector of TestInfos. TestCase is not copyable.
86256. A compile-time bool constant that is true if and only if x is a null pointer literal (i.e. NULL or any 0-valued compile-time integral constant).
86257. These two compilers are known to support SEH.
86258. Makes sure T is a complete type.
86259. Inside a test, refer to TypeParam to get the type parameter.
86260. Then prints the value itself.
86261. Declares the flags.
86262. Sets the should_run member.
86263. Implements printing a non-reference type T by letting the compiler pick the right overload of PrintTo() for T.
86264. Appends an event listener to the end of the list. Google Test assumes the ownership of the listener (i.e. it will delete the listener when the test program finishes).
86265. GTEST_INCLUDE_GTEST_GTEST_H
86266. Compares two wide C strings, ignoring case. Returns true iff they have the same content. Unlike wcscasecmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL wide C string, including the empty string. NB: The implementations on different platforms slightly differ. On windows, this method uses _wcsicmp which compares according to LC_CTYPE environment variable. On GNU platform this method uses wcscasecmp which compares according to LC_CTYPE category of the current locale. On MacOS X, it uses towlower, which also uses LC_CTYPE category of the current locale.
86267. Defines the abstract factory interface that creates instances of a Test object.
86268. EXPECT_DEBUG_DEATH asserts that the given statements die in debug mode. The death testing framework causes this to have interesting semantics, since the sideeffects of the call are only visible in opt mode, and not in debug mode. In practice, this can be used to test functions that utilize the LOG(DFATAL) macro using the following style: int DieInDebugOr12(int* sideeffect) { if (sideeffect) { *sideeffect = 12; } LOG(DFATAL) << "death"; return 12; } TEST(TestCase, TestDieOr12WorksInDbgAndOpt) { int sideeffect = 0; // Only asserts in dbg. EXPECT_DEBUG_DEATH(DieInDebugOr12(&sideeffect), "death"); #ifdef NDEBUG // opt-mode has sideeffect visible. EXPECT_EQ(12, sideeffect); #else // dbg-mode no visible sideeffect. EXPECT_EQ(0, sideeffect); #endif } This will assert that DieInDebugReturn12InOpt() crashes in debug mode, usually due to a DCHECK or LOG(DFATAL), but returns the appropriate fallback value (12 in this case) in opt mode. If you need to test that a function has appropriate side-effects in opt mode, include assertions against the side-effects. A general pattern for this is: EXPECT_DEBUG_DEATH({ // Side-effects here will have an effect after this statement in // opt mode, but none in debug mode. EXPECT_EQ(12, DieInDebugOr12(&sideeffect)); }, "death");
86269. t2
86270. #ifdef __GNUC__ is too general here. It is possible to use gcc without using libstdc++ (which is where cxxabi.h comes from).
86271. class CartesianProductGenerator9::Iterator
86272. Returns the elapsed time, in milliseconds.
86273. Initializes this object with a copy of the input.
86274. Overloads for wide C strings
86275. **comment:** We print a protobuf using its ShortDebugString() when the string doesn't exceed this many characters; otherwise we print it using DebugString() for better readability.
label: code-design
86276. Provides a level of indirection for the test list to allow easy shuffling and restoring the test order. The i-th element in this vector is the index of the i-th test in the shuffled test list.
86277. The test failure summary. The test failure message.
86278. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: wan@google.com (Zhangyong Wan) Low-level types and utilities for porting Google Test to various platforms. All macros ending with _ and symbols defined in an internal namespace are subject to change without notice. Code outside Google Test MUST NOT USE THEM DIRECTLY. Macros that don't end with _ are part of Google Test's public API and can be used by code outside Google Test. This file is fundamental to Google Test. All other Google Test source files are expected to #include this. Therefore, it cannot #include any other Google Test header.
86279. class CartesianProductHolder5
86280. When this flag is specified, a failed assertion will throw an exception if exceptions are enabled, or exit the program with a non-zero code otherwise.
86281. 'index' is the index of the test in the type list 'Types' specified in INSTANTIATE_TYPED_TEST_CASE_P(Prefix, TestCase, Types). Valid values for 'index' are [0, N - 1] where N is the length of Types.
86282. GTEST_HAS_TYPED_TEST_P
86283. Formats a source file path and a line number as they would appear in an error message from the compiler used to compile this code.
86284. class ValuesInIteratorRangeGenerator
86285. CopyArray() copies a k-dimensional native array using the elements' operator=, where k can be any integer ≥ 0 . When k is 0, CopyArray() degenerates into copying a single value.
86286. A dummy implementation of synchronization primitives (mutex, lock, and thread-local variable). Necessary for compiling Google Test where mutex is not supported - using Google Test in multiple threads is not supported on such platforms.
86287. **comment:** Adds a failure if the key is a reserved attribute of Google Test testcase tags. Returns true if the property is valid. TODO(russr): Validate attribute names are legal and human readable.
label: code-design
86288. Sets the default_result_printer attribute to the provided listener. The listener is also added to the listener list and previous default_result_printer is removed from it and deleted. The listener can also be NULL in which case it will not be added to the list. Does nothing if the previous and the current listener objects are the same.
86289. __GNUC__ && (GTEST_GCC_VER_ \geq 30400) && !COMPILER_ICC

86290. **comment:** Prints the given value to the given ostream. If the value is a protocol message, its debug string is printed; if it's an enum or of a type implicitly convertible to BiggestInt, it's printed as an integer; otherwise the bytes in the value are printed. This is what UniversalPrinter<T>::Print() does when it knows nothing about type T and T has neither << operator nor PrintTo(). A user can override this behavior for a class type Foo by defining a << operator in the namespace where Foo is defined. We put this operator in namespace 'internal' instead of 'internal' to simplify the implementation, as much code in 'internal' needs to use << in STL, which would conflict with our own << were it defined in 'internal'. Note that this operator<< takes a generic std::basic_ostream<Char, CharTraits> type instead of the more restricted std::ostream. If we define it to take an std::ostream instead, we'll get an "ambiguous overloads" compiler error when trying to print a type Foo that supports streaming to std::basic_ostream<Char, CharTraits>, as the compiler cannot tell whether operator<<(std::ostream&, const T&) or operator<<(std::basic_stream<Char, CharTraits>, const Foo&) is more specific.

label: code-design

86291. You can implement all the usual class fixture members here.

86292. Take over ownership of a raw pointer. This should happen as soon as possible after the object is created.

86293. Initializes this object with a reference of the input.

86294. DeathTest is a class that hides much of the complexity of the GTEST_DEATH_TEST macro. It is abstract; its static Create method returns a concrete class that depends on the prevailing death test style, as defined by the --gtest_death_test_style and/or --gtest_internal_run_death_test flags.

86295. __GXX_RTTI

86296. Returns true iff the current test has a (either fatal or non-fatal) failure.

86297. **comment:** Separate the error generating code from the code path to reduce the stack frame size of CmpHelperOP. This helps reduce the overhead of some sanitizers when calling EXPECT_OP in a tight loop.

label: code-design

86298. class CartesianProductHolder4

86299. Fired before any test activity starts.

86300. UnitTest class invokes this method to register tests in this test case right before running them in RUN_ALL_TESTS macro. This method should not be called more than once on any single instance of a ParameterizedTestCaseInfoBase derived class.

86301. GTEST_INCLUDE_GTEST_GTEST_TEST_PART_H Copyright 2008 Google Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhangyong Wan)

86302. Implements thread-local storage on Windows systems. // Thread 1 ThreadLocal<int> tl(100); // 100 is the default value for each thread. // Thread 2 tl.set(150); // Changes the value for thread 2 only. EXPECT_EQ(150, tl.get()); // Thread 1 EXPECT_EQ(100, tl.get()); // In thread 1, tl has the original value. tl.set(200); EXPECT_EQ(200, tl.get()); The template type argument T must have a public copy constructor. In addition, the default ThreadLocal constructor requires T to have a public default constructor. The users of a ThreadLocal instance have to make sure that all but one threads (including the main one) using that instance have exited before destroying it. Otherwise, the per-thread objects managed for them by the ThreadLocal instance are not guaranteed to be destroyed on all platforms. Google Test only uses global ThreadLocal objects. That means they will die after main() has returned. Therefore, no per-thread object managed by Google Test will be leaked as long as all threads using Google Test have exited when main() returns.

86303. Typed tests need <typeinfo> and variadic macros, which GCC, VC++ 8.0, Sun Pro CC, IBM Visual Age, and HP aCC support.

86304. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE DIRECTLY. Expands to the name of the variable used to remember the names of the registered tests in the given test case.

86305. Allows a controller thread to pause execution of newly created threads until notified. Instances of this class must be created and destroyed in the controller thread. This class is only for testing Google Test's own constructs. Do not use it in user tests, either directly or indirectly.

86306. Starting with version 4.3.2, gcc defines __GXX_RTTI iff RTTI is enabled.

86307. STL-style container methods.

86308. class CartesianProductGenerator2::Iterator

86309. **comment:** Tell the compiler to warn about unused return values for functions declared with this macro. The macro should be used on function declarations following the argument list: Sprocket* AllocateSprocket() GTEST_MUST_USE_RESULT_;

label: code-design

86310. The vector of TestPartResults

86311. A secret type that Google Test users don't know about. It has no definition on purpose. Therefore it's impossible to create a Secret object, which is what we want.

86312. Constructs and returns the message for an equality assertion (e.g. ASSERT_EQ, EXPECT_STREQ, etc) failure. The first four parameters are the expressions used in the assertion and their values, as strings. For example, for ASSERT_EQ(foo, bar) where foo is 5 and bar is 6, we have: expected_expression: "foo" actual_expression: "bar" expected_value: "5" actual_value: "6" The ignoring_case parameter is true iff the assertion is a *_STRCASEEQ*. When it's true, the string "(ignoring case)" will be inserted into the message.

86313. Internal macro for implementing {EXPECT|ASSERT}_PRED5. Don't use this in your code.

86314. This assumes that non-Windows OSes provide unistd.h. For OSes where this is not the case, we need to include headers that provide the functions mentioned above.

86315. GTEST_HAS_RTTI

86316. GTEST_HAS_CLONE

86317. **comment:** FullMatch(str, re) returns true iff regular expression re matches the entire str. PartialMatch(str, re) returns true iff regular expression re matches a substring of str (including str itself). TODO(wan@google.com): make FullMatch() and PartialMatch() work when str contains NUL characters.

label: code-design

86318. Helpers for ThreadLocal.

86319. Defines RE.

86320. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. ParameterizedTestCaseInfoBase is a generic interface to ParameterizedTestCaseInfo classes. ParameterizedTestCaseInfoBase accumulates test information provided by TEST_P macro invocations and generators provided by

INSTANTIATE_TEST_CASE_P macro invocations and uses that information to register all resulting test instances in RegisterTests method. The ParameterizeTestCaseRegistry class holds a collection of pointers to the ParameterizedTestCaseInfo objects and calls RegisterTests() on each of them when asked.

86321. If *pstr starts with the given prefix, modifies *pstr to be right past the prefix and returns true; otherwise leaves *pstr unchanged and returns false. None of pstr, *pstr, and prefix can be NULL.

86322. Called by pthread to delete thread-local data stored by pthread_setspecific().

86323. The helper function for {ASSERT|EXPECT}_STRNE. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

86324. **comment:** Formats a file location for compiler-independent XML output. Although this function is not platform dependent, we put it next to FormatFileLocation in order to contrast the two functions.

label: code-design

86325. Determines whether test results can be streamed to a socket.

86326. **comment:** The friend relationship of some of these classes is cyclic. If we don't forward declare them the compiler might confuse the classes in friendship clauses with same named classes on the scope.

label: code-design

86327. This specialization is used when the first argument to ASSERT_EQ() is a null pointer literal, like NULL, false, or 0.

86328. Destructor of TestCase.

86329. Determines whether Google Test can use tr1/tuple. You can define this macro to 0 to prevent Google Test from using tuple (any feature depending on tuple with be disabled in this mode).

86330. The maximum number of elements to print.
86331. ID of the test fixture class True iff this test should run True iff this test is disabled True if this test matches the user-specified filter. The factory that creates the test object
86332. The text used in failure messages to indicate the start of the stack trace.
86333. The thread holding the mutex.
86334. Used to print an STL-style container when the user doesn't define a PrintTo() for it.
86335. class CartesianProductHolder7
86336. This flag controls whether Google Test emits a detailed XML report to a file in addition to its normal textual output.
86337. No type parameter.
86338. Unary predicate assertion macros.
86339. Stat(), RmDir(), and IsDir() are not needed on Windows CE at this time and thus not defined there.
86340. Given directory = "dir", relative_path = "test.xml", returns "dir/test.xml". On Windows, uses \ as the separator rather than /.
86341. Sets up, executes, and tears down the test.
86342. Returns the result of the test.
86343. TestEventListeners lets users add listeners to track events in Google Test.
86344. namespace gtest_internal
86345. When a thread exits, DeleteThreadLocalValue() will be called on the object managed for that thread.
86346. gcc defines __EXCEPTIONS to 1 iff exceptions are enabled.
86347. A simple Linear Congruential Generator for generating random numbers with a uniform distribution. Unlike rand() and srand(), it doesn't use global state (and therefore can't interfere with user code). Unlike rand_r(), it's portable. An LCG isn't very random, but it's good enough for our purposes.
86348. If the type list contains only one type, you can write that type directly without Types<...>: INSTANTIATE_TYPED_TEST_CASE_P(My, FooTest, int);
86349. Tears down the test fixture.
86350. Sun Pro CC supports exceptions. However, there is no compile-time way of detecting whether they are enabled or not. Therefore, we assume that they are enabled unless the user tells us otherwise.
86351. State of the definition of a type-parameterized test case.
86352. Returns true iff the test is disabled and will be reported in the XML report.
86353. Releases resources associated with the key. This will *not* delete managed objects for other threads.
86354. Fired before the test case starts.
86355. Returns true iff the test case failed.
86356. The user didn't tell us not to do it, so we assume it's OK.
86357. line
86358. Instead of 1/0, we want to see true/false for bool values.
86359. GTEST_HAS_PTHREAD && !GTEST_OS_WINDOWS_MINGW
86360. This flag controls whether Google Test catches all test-thrown exceptions and logs them as failures.
86361. Returns the test case name.
86362. GTEST_HAS_STD_MOVE_
86363. Gets the number of all tests.
86364. signed/unsigned mismatch
86365. Helper function for implementing {EXPECT|ASSERT}_PRED4. Don't use this in your code.
86366. STL-style container typedefs.
86367. Logs a property for the current test, test case, or for the entire invocation of the test program when used outside of the context of a test case. Only the last value for a given key is remembered. These are public static so they can be called from utility functions that are not members of the test fixture. Calls to RecordProperty made during lifespan of the test (from the moment its constructor starts to the moment its destructor finishes) will be output in XML as attributes of the < testcase > element. Properties recorded from fixture's SetUpTestCase or TearDownTestCase are logged as attributes of the corresponding < testsuite > element. Calls to RecordProperty made in the global context (before or after invocation of RUN_ALL_TESTS and from SetUp/TearDown method of Environment objects registered with Google Test) will be output as attributes of the < testsuites > element.
86368. Sets up the test fixture.
86369. Returns true iff test is disabled.
86370. **comment:** A TestInfo object stores the following information about a test: Test case name Test name Whether the test should be run A function pointer that creates the test object when invoked Test result The constructor of TestInfo registers itself with the UnitTest singleton such that the RUN_ALL_TESTS() macro knows which tests to run.
label: test
86371. Pushes a trace defined by SCOPED_TRACE() on to the per-thread Google Test trace stack.
86372. **comment:** Provides leak-safe Windows kernel handle ownership. Used in death tests and in threading support.
label: code-design
86373. The user didn't tell us, so we need to figure it out.
86374. Gets the number of failed tests in this test case.
86375. Defining a variable of type CompileAssertTypesEqual<T1, T2> will cause a compiler error iff T1 and T2 are different types.
86376. Returns a pathname for a file that does not currently exist. The pathname will be directory/base_name.extension or directory/base_name_<number>.extension if directory/base_name.extension already exists. The number will be incremented until a pathname is found that does not already exist. Examples: 'dir/foo_test.xml' or 'dir/foo_test_1.xml'. There could be a race condition if two or more processes are calling this function at the same time -- they could both pick the same filename.
86377. **comment:** Parses 'str' for a 32-bit signed integer. If successful, writes the result to *value and returns true; otherwise leaves *value unchanged and returns false.
TODO(chandlerc): Find a better way to refactor flag and environment parsing out of both gtest-port.cc and gtest.cc to avoid exporting this utility function.
label: code-design
86378. **comment:** To write value-parameterized tests, first you should define a fixture class. It is usually derived from testing::TestWithParam<T> (see below for another inheritance scheme that's sometimes useful in more complicated class hierarchies), where the type of your parameter values. TestWithParam<T> is itself derived from testing::Test. T can be any copyable type. If it's a raw pointer, you are responsible for managing the lifespan of the pointed values.
label: code-design
86379. The thread can be created only after all fields except thread_ have been initialized.
86380. GTEST_IS_THREADSAFE
86381. Returns a copy of the FilePath with the directory part removed. Example: FilePath("path/to/file").RemoveDirectoryName() returns FilePath("file"). If there is no directory part ("just_a_file"), it returns the FilePath unmodified. If there is no file part ("just_a_dir") it returns an empty FilePath (""). On Windows platform, \ is the path separator, otherwise it is '/'.
86382. It's a NAN if the exponent bits are all ones and the fraction bits are not entirely zeros.
86383. Static methods
86384. TypeParameterizedTest<Fixture, TestSel, Types>::Register() registers a list of type-parameterized tests with Google Test. The return value is insignificant - we just need to return something such that we can call this function in a namespace scope. Implementation note: The GTEST_TEMPLATE_ macro declares a template template parameter. It's defined in gtest-type-util.h.
86385. Acquires this mutex.
86386. LocalTestInfo structure keeps information about a single test registered with TEST_P macro.
86387. **comment:** Defines logging utilities: GTEST_LOG_(severity) - logs messages at the specified severity level. The message itself is streamed into the macro. LogToStderr() - directs all log messages to stderr. FlushInfoLog() - flushes informational log messages.
label: code-design
86388. Creates an ANSI string from the given wide string, allocating memory using new. The caller is responsible for deleting the return value using delete[]. Returns the ANSI string, or NULL if the input is NULL. The returned string is created using the ANSI codepage (CP_ACP) to match the behaviour of the ANSI versions of Win32 calls and the C runtime.
86389. Determines the platform on which Google Test is compiled.

86390. Implements the helper function for {ASSERT|EXPECT}_LT
86391. These two functions are overloaded. Given an expression Helper(x), the compiler will pick the first version if x can be implicitly converted to type To; otherwise it will pick the second version. The first version returns a value of size 1, and the second version returns a value of size 2. Therefore, by checking the size of Helper(x), which can be done at compile time, we can tell which version of Helper() is used, and hence whether x can be implicitly converted to type To.
86392. Return value used only to run this method in namespace scope.
86393. Returns the text representation of the value parameter, or NULL if this is not a value-parameterized test.
86394. We use our own TR1 tuple if we aren't sure the user has an implementation of it already. At this time, libstdc++ 4.0.0+ and MSVC 2010 are the only mainstream standard libraries that come with a TR1 tuple implementation. NVIDIA's CUDA NVCC compiler pretends to be GCC by defining __GNUC__ and friends, but cannot compile GCC's tuple implementation. MSVC 2008 (9.0) provides TR1 tuple in a 323 MB Feature Pack download, which we cannot assume the user has. QNX's QCC compiler is a modified GCC but it doesn't support TR1 tuple. libc++ only provides std::tuple, in C++11 mode, and it can be used with some compilers that define __GNUC__.
86395. Since the lock is being released the owner_ field should no longer be considered valid. We don't protect writing to has_owner_ here, as it's the caller's responsibility to ensure that the current thread holds the mutex when this is called.
86396. NOLINT GTEST_OS_WINDOWS
86397. Ensures that there is at least one operator<< in the global namespace. See Message& operator<<(...) below for why.
86398. **comment:** String - an abstract class holding static string utilities.
label: code-design
86399. RemoveFileName returns the directory path with the filename removed. Example: FilePath("path/to/file").RemoveFileName() returns "path/to/". If the FilePath is "a_file" or "/a_file", RemoveFileName returns FilePath("./") or, on Windows, FilePath("\\"). If the filepath does not have a file, like "just/a/dir/", it returns the FilePath unmodified. On Windows platform, '\' is the path separator, otherwise it is '/'.
86400. Enough has been printed.
86401. The relation between an NativeArray object (see below) and the native array it represents. We use 2 different structs to allow non-copyable types to be used, as long as RelationToSourceReference() is passed.
86402. Returns the text streamed into this AssertionResult. Test assertions use it when they fail (i.e., the predicate's outcome doesn't match the assertion's expectation). When nothing has been streamed into the object, returns an empty string.
86403. Overload for wchar_t type. Prints a wchar_t as a symbol if it is printable or as its internal code otherwise and also as its decimal code (except for L'0'). The L'0' char is printed as "L'\0'". The decimal code is printed as signed integer when wchar_t is implemented by the compiler as a signed type and is printed as an unsigned integer when wchar_t is implemented as an unsigned type.
86404. **comment:** We don't want to require the users to write TemplatesN<...> directly, as that would require them to count the length. Templates<...> is much easier to write, but generates horrible messages when there is a compiler error, as gcc insists on printing out each template argument, even if it has the default value (this means Templates<list> will appear as Templates<list, NoneT, NoneT, ..., NoneT> in the compiler errors). Our solution is to combine the best part of the two approaches: a user would write Templates<T1, ..., TN>, and Google Test will translate that to TemplatesN<T1, ..., TN> internally to make error messages readable. The translation is done by the 'type' member of the Templates template.
label: code-design
86405. If the compiler is not GCC 4.0+, we assume the user is using a spec-conforming TR1 implementation. IWYU pragma: export // NOLINT
 GTEST_USE_OWN_TR1_TUPLE
86406. Macros for defining flags.
86407. Prints TestPartResult object.
86408. test_part_result
86409. The Mutex class can only be used for mutexes created at runtime. It shares its API with MutexBase otherwise.
86410. Helper function for printing a tuple. T must be instantiated with a tuple type.
86411. namespace posix
86412. Utilities for char.
86413. Depending on the platform, different string classes are available. On Linux, in addition to ::std::string, Google also makes use of class ::string, which has the same interface as ::std::string, but has a different implementation. You can define GTEST_HAS_GLOBAL_STRING to 1 to indicate that ::string is available AND is a distinct type to ::std::string, or define it to 0 to indicate otherwise. If ::std::string and ::string are the same class on your platform due to aliasing, you should define GTEST_HAS_GLOBAL_STRING to 0. If you do not define GTEST_HAS_GLOBAL_STRING, it is defined heuristically.
86414. We disallow copying UnitTest.
86415. This macro is used for implementing macros such as EXPECT_DEATH_IF_SUPPORTED and ASSERT_DEATH_IF_SUPPORTED on systems where death tests are not supported. Those macros must compile on such systems iff EXPECT_DEATH and ASSERT_DEATH compile with the same parameters on systems that support death tests. This allows one to write such a macro on a system that does not support death tests and be sure that it will compile on a death-test supporting system. Parameters: statement - A statement that a macro such as EXPECT_DEATH would test for program termination. This macro has to make sure this statement is compiled but not executed, to ensure that EXPECT_DEATH_IF_SUPPORTED compiles with a certain parameter iff EXPECT_DEATH compiles with it. regex - A regex that a macro such as EXPECT_DEATH would use to test the output of statement. This parameter has to be compiled but not evaluated by this macro, to ensure that this macro only accepts expressions that a macro such as EXPECT_DEATH would accept. terminator - Must be an empty statement for EXPECT_DEATH_IF_SUPPORTED and a return statement for ASSERT_DEATH_IF_SUPPORTED. This ensures that ASSERT_DEATH_IF_SUPPORTED will not compile inside functions where ASSERT_DEATH doesn't compile. The branch that has an always false condition is used to ensure that statement and regex are compiled (and thus syntactically correct) but never executed. The unreachable code macro protects the terminator statement from generating an 'unreachable code' warning in case statement unconditionally returns or throws. The Message constructor at the end allows the syntax of streaming additional messages into the macro, for compilational compatibility with EXPECT_DEATH/ASSERT_DEATH.
86416. Prints a value using the type inferred by the compiler. The difference between this and UniversalTersePrint() is that for a (const) char pointer, this prints both the pointer and the NUL-terminated string.
86417. Define macro GTEST_DONT_DEFINE_ASSERT_XY to 1 to omit the definition of ASSERT_XY(), which clashes with some users' own code.
86418. GTEST_HASH_CXXABI_H
86419. Generates values from a range of two comparable values. Can be used to generate sequences of user-defined types that implement operator+() and operator<(). This class is used in the Range() function.
86420. t
86421. Two overloaded helpers for checking at compile time whether an expression is a null pointer literal (i.e. NULL or any 0-valued compile-time integral constant). Their return values have different sizes, so we can use sizeof() to test which version is picked by the compiler. These helpers have no implementations, as we only need their signatures. Given IsNullLiteralHelper(x), the compiler will pick the first version if x can be implicitly converted to Secret*, and pick the second version otherwise. Since Secret is a secret and incomplete type, the only expression a user can write that has type Secret* is a null pointer literal. Therefore, we know that x is a null pointer literal if and only if the first version is picked by the compiler.
86422. Returns the current OS stack trace as an std::string. The maximum number of stack frames to be included is specified by the gtest_stack_trace_depth flag. The skip_count parameter specifies the number of top frames to be skipped, which doesn't count against the number of frames to be included. For example, if Foo() calls Bar(), which in turn calls GetCurrentOsStackTraceExceptTop(..., 1), Foo() will be included in the trace but Bar() and GetCurrentOsStackTraceExceptTop() won't.
86423. **comment:** MSVC's and C++Builder's implementations of the STL use the _HAS_EXCEPTIONS macro to enable exceptions, so we'll do the same. Assumes that exceptions are enabled by default.
label: code-design
86424. A concrete DeathTestFactory implementation for normal use.
86425. Text representation of the value parameter, or NULL if this is not a value-parameterized test.
86426. The convenience class for users who need to override just one or two methods and are not concerned that a possible change to a signature of the methods they override will not be caught during the build. For comments about each method please see the definition of TestEventListener above.
86427. Holds test properties recorded during execution of SetUpTestCase and TearDownTestCase.
86428. GTEST_HAS_GLOBAL_STRING
86429. This namespace MUST NOT BE NESTED IN ::testing, or the name look-up magic needed for implementing UniversalPrinter won't work.
86430. **comment:** clang defines __EXCEPTIONS iff exceptions are enabled before clang 220714, but iff cleanups are enabled after that. In Obj-C++ files, there can be cleanups for ObjC exceptions which also need cleanups, even if C++ exceptions are disabled. clang has __has_feature(cxx_exceptions) which checks for C++

exceptions starting at clang r206352, but which checked for cleanups prior to that. To reliably check for C++ exception availability with clang, check for `__EXCEPTIONS && __has_feature(cxx_exceptions)`.

label: code-design

86431. `_MSC_VER`

86432. Override this to define how to tear down the environment.

86433. `GTEST_INCLUDE_GTEST_GTEST_PRINTERS_H_`

86434. Appends the contents of message to message_.

86435. This template class serves as a compile-time function from size to type. It maps a size in bytes to a primitive type with that size. e.g. `TypeWithSize<4>::UInt` is typedef-ed to be unsigned int (unsigned integer made up of 4 bytes). Such functionality should belong to STL, but I cannot find it there. Google Test uses this class in the implementation of floating-point comparison. For now it only handles UInt (unsigned int) as that's all Google Test needs. Other types can be easily added in the future if need arises.

86436. Uses a GTestFlagSaver to save and restore all Google Test flags.

86437. Parses a bool/Int32/string from the environment variable corresponding to the given Google Test flag.

86438. Constructs a FloatingPoint from a raw floating-point number. On an Intel CPU, passing a non-normalized NAN (Not a Number) around may change its bits, although the new value is guaranteed to be also a NAN. Therefore, don't expect this constructor to preserve the bits in x when x is a NAN.

86439. Gets the (immutable) vector of TestInfos in this TestCase.

86440. Advances iterator to point to the next element provided by the generator. The caller is responsible for not calling Advance() on an iterator equal to BaseGenerator()->End().

86441. When the compiler sees expression `IsContainerTest<C>(0)`, if C is an STL-style container class, the first overload of IsContainerTest will be viable (since both `C::iterator*` and `C::const_iterator*` are valid types and `NULL` can be implicitly converted to them). It will be picked over the second overload as 'int' is a perfect match for the type of argument 0. If `C::iterator` or `C::const_iterator` is not a valid type, the first overload is not viable, and the second overload will be picked. Therefore, we can determine whether C is a container class by checking the type of `IsContainerTest<C>(0)`. The value of the expression is insignificant. Note that we look for both `C::iterator` and `C::const_iterator`. The reason is that C++ injects the name of a class as a member of the class itself (e.g. you can refer to class iterator as either 'iterator' or 'iterator::iterator'). If we look for `C::iterator` only, for example, we would mistakenly think that a class named iterator is an STL container. Also note that the simpler approach of overloading `IsContainerTest(typename C::const_iterator*)` and `IsContainerTest(...)` doesn't work with Visual Age C++ and Sun C++.

86442. **comment:** Result of an assertion. Represents a failure message. Represents a test. Information about a test. Result of a test part. A collection of test cases.

label: test

86443. `GTEST_n_TYPENAMES_(T)` declares a list of n typenames.

86444. Implements the helper function for `{ASSERT|EXPECT}_NE`

86445. `INSTANTIATE_TEST_CASE_P` macro uses AddGenerator() to record information about a generator.

86446. Gets the line in the source file where the test part took place, or -1 if it's unknown.

86447. 6.1.3.3 Tuple helper classes.

86448. Listener responsible for the creation of the XML output file.

86449. class `CartesianProductGenerator5`

86450. `EXPECT_DEATH_IF_SUPPORTED(statement, regex)` and `ASSERT_DEATH_IF_SUPPORTED(statement, regex)` expand to real death tests if death tests are supported; otherwise they just issue a warning. This is useful when you are combining death test assertions with normal test assertions in one test.

86451. Overloads for other simple built-in types.

86452. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Expands to the name of the typedef for the type parameters of the given test case.

86453. Implementation details of `GTEST_COMPILE_ASSERT_`: (In C++11, we simply use `static_assert` instead of the following) - `GTEST_COMPILE_ASSERT_` works by defining an array type that has -1 elements (and thus is invalid) when the expression is false. - The simpler definition `#define`

`GTEST_COMPILE_ASSERT_(expr, msg)` `typedef char msg[(expr) ? 1 : -1]` does not work, as gcc supports variable-length arrays whose sizes are determined at run-time (this is gcc's extension and not part of the C++ standard). As a result, gcc fails to reject the following code with the simple definition: `int foo;`
`GTEST_COMPILE_ASSERT_(foo, msg); // not supposed to compile as foo is // not a compile-time constant.` - By using the type `CompileAssert<(bool(expr))>`, we ensures that expr is a compile-time constant. (Template arguments must be determined at compile-time.) - The outer parentheses in `CompileAssert<(bool(expr))>` are necessary to work around a bug in gcc 3.4.4 and 4.0.1. If we had written `CompileAssert<bool(expr)>` instead, these compilers will refuse to compile `GTEST_COMPILE_ASSERT_(5 > 0, some_message);` (They seem to think the ">" in "5 > 0" marks the end of the template argument list.) - The array size is `(bool(expr) ? 1 : -1)`, instead of simply `((expr) ? 1 : -1)`. This is to avoid running into a bug in MS VC 7.1, which causes `((0.0) ? 1 : -1)` to incorrectly evaluate to 1.

86454. Elapsed time, in milliseconds.

86455. We used to have a second template parameter instead of `Secret*`. That template parameter would deduce to 'long', making this a better match than the first overload even without the first overload's `EnableIf`. Unfortunately, gcc with -Wconversion-null warns when "passing NULL to non-pointer argument" (even a deduced integral argument), so the old implementation caused warnings in user code.

86456. The actual list of listeners.

86457. Define this macro to 1 to omit the definition of `SUCCEED()`, which is a generic name and clashes with some other libraries.

86458. The `TypeList` template makes it possible to use either a single type or a `Types<...>` list in `TYPED_TEST_CASE()` and `INSTANTIATE_TYPED_TEST_CASE_P()`.

86459. Gets the number of disabled tests.

86460. Constructs an empty Message.

86461. Gets the singleton `UnitTest` object. The first time this method is called, a `UnitTest` object is constructed and returned. Consecutive calls will return the same object.

86462. Holds a value of T. Can be deleted via its base class without the caller knowing the type of T.

86463. Functions deprecated by MSVC 8.0.

86464. This overload prints a (const) `wchar_t` array compactly.

86465. For static mutexes, we rely on these members being initialized to zeros by the linker.

86466. Gets the summary of the failure message by omitting the stack trace in it.

86467. `GTEST_INCLUDE_GTEST_INTERNAL_GTEST_STRING_H_` Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: keith.ray@gmail.com (Keith Ray) Google Test filepath utilities This header file declares classes and functions used internally by Google Test. They are subject to change without notice. This file is #included in `<gtest/internal/gtest-internal.h>`. Do not include this header file separately!

86468. The empty type list.

86469. Returns true iff the given string ends with the given suffix, ignoring case. Any string is considered to end with an empty suffix.

86470. Holds a value of type T.

86471. Clears the object.

86472. These fields are immutable properties of the test. Test case name Test name Name of the parameter type, or NULL if this is not a typed or a type-parameterized test.

86473. Internal macro for implementing `{EXPECT|ASSERT}_PRED_FORMAT1`. Don't use this in your code.

86474. Tersely prints the first N fields of a tuple to a string vector, one element for each field.

86475. This macro is for implementing `ASSERT_DEATH*`, `EXPECT_DEATH*`, `ASSERT_EXIT*`, and `EXPECT_EXIT*`.

86476. The line in the source file where the test part took place, or -1 if the line number is unknown.

86477. We must report iterators equal if they both point beyond their respective ranges. That can happen in a variety of fashions, so we have to consult AtEnd().
86478. Older versions of MSVC don't have __pragma.
86479. **comment:** Returns true iff this number is at most kMaxUlps ULP's away from rhs. In particular, this function: - returns false if either number is (or both are) NAN.
- treats really large numbers as almost equal to infinity. - thinks +0.0 and -0.0 are 0 DLP's apart.
label: code-design
86480. !GTEST_OS_WINDOWS
86481. The explicit constructor call suppresses a false warning emitted by gcc when supplied with the -Wextra option.
86482. **comment:** Shuffles the tests in this test case.
label: test
86483. Like ASSERT_EXIT, but continues on to successive tests in the test case, if any:
86484. Returns true iff the test part passed.
86485. dummy
86486. Adds a test property to the list. The property is validated and may add a non-fatal failure if invalid (e.g., if it conflicts with reserved key names). If a property is already recorded for the same key, the value will be updated, rather than storing multiple values for the same key. xml_element specifies the element for which the property is being recorded and is used for validation.
86487. Interface for iterating over elements provided by an implementation of ParamGeneratorInterface<T>.
86488. The d'tor is virtual as we need to subclass Environment.
86489. Since T has no << operator or PrintTo() but can be implicitly converted to BiggestInt, we print it as a BiggestInt. Most likely T is an enum type (either named or unnamed), in which case printing it as an integer is the desired behavior. In case T is not an enum, printing it as an integer is the best we can do given that it has no user-defined printer.
86490. **comment:** Define this macro to 1 to omit the definition of TEST(), which is a generic name and clashes with some other libraries.
label: code-design
86491. Implements Boolean test assertions such as EXPECT_TRUE. expression can be either a boolean expression or an AssertionResult. text is a textual representation of expression as it was passed into the EXPECT_TRUE.
86492. **comment:** We don't want to require the users to write TypesN<...> directly, as that would require them to count the length. Types<...> is much easier to write, but generates horrible messages when there is a compiler error, as gcc insists on printing out each template argument, even if it has the default value (this means Types<int> will appear as Types<int, None, None, ..., None> in the compiler errors). Our solution is to combine the best part of the two approaches: a user would write Types<T1, ..., TN>, and Google Test will translate that to TypesN<T1, ..., TN> internally to make error messages readable. The translation is done by the 'type' member of the Types template.
label: code-design
86493. By default, ::testing::internal::PrintTo() is used for printing the value. Thanks to Koenig look-up, if T is a class and has its own PrintTo() function defined in its namespace, that function will be visible here. Since it is more specific than the generic ones in ::testing::internal, it will be picked by the compiler in the following statement - exactly what we want.
86494. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_INTERNAL_H_
86495. Appends the given TestPartResult to the array.
86496. Returns the maximum representable finite floating-point number.
86497. Clones the iterator object. Used for implementing copy semantics of ParamIterator<T>.
86498. namespace testing_internal
86499. Stores result of the assertion predicate.
86500. Clears the results of all tests in this test case.
86501. Fired after the test ends.
86502. Returns true if FilePath describes something in the file-system, either a file, directory, or whatever, and that something exists.
86503. Makes a failed assertion result.
86504. GTEST_USE_OWN_TR1_TUPLE
86505. The underlying pthread mutex. has_owner_ indicates whether the owner_ field below contains a valid thread ID and is therefore safe to inspect (e.g., to use in pthread_equal()). All accesses to the owner_ field should be protected by a check of this field. An alternative might be to memset() owner_ to all zeros, but there's no guarantee that a zero'd pthread_t is necessarily invalid or even different from pthread_self().
86506. class CartesianProductHolder10
86507. The key supplied by the user.
86508. Returns true iff the test case passed.
86509. Internal macro for implementing {EXPECT|ASSERT}_PRED3. Don't use this in your code.
86510. ChDir(), FReopen(), FDOpen(), Read(), Write(), Close(), and StrError() aren't needed on Windows CE at this time and thus not defined there.
86511. TypeWithoutFormatter<T, kTypeKind>::PrintValue(value, os) is called by the universal printer to print a value of type T when neither operator<< nor PrintTo() is defined for T, where kTypeKind is the "kind" of T as defined by enum TypeKind.
86512. **comment:** A ScopedTrace object does its job in its c'tor and d'tor. Therefore it doesn't need to be used otherwise.
label: code-design
86513. GTEST_HAS_STD_WSTRING
86514. The vector of TestProperties
86515. BOOST_HAS_TR1_TUPLE
86516. See the comments in Message& operator <<(const T&) above for why we need this using statement.
86517. **comment:** Returns a human-readable outcome message regarding the outcome of the last death test.
label: code-design
86518. Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: vladl@google.com (Vlad Losev) Macros and functions for implementing parameterized tests in Google C++ Testing Framework (Google Test) This file is generated by a SCRIPT. DO NOT EDIT BY HAND!
86519. Returns the TestInfo object for the test that's currently running, or NULL if no test is running.
86520. _HAS_EXCEPTIONS
86521. Returns the optimal edits to go from 'left' to 'right'. All edits cost the same, with replace having lower priority than add/remove. Simple implementation of the Wagner-Fischer algorithm. See http://en.wikipedia.org/wiki/Wagner-Fischer_algorithm
86522. Copy constructor.
86523. Compiling in at least C++11 mode.
86524. Range() returns generators providing sequences of values in a range. Synopsis: Range(start, end) - returns a generator producing a sequence of values {start, start+1, start+2, ...}. Range(start, end, step) - returns a generator producing a sequence of values {start, start+step, start+step+step, ...}. Notes: * The generated sequences never include end. For example, Range(1, 5) returns a generator producing a sequence {1, 2, 3, 4}. Range(1, 9, 2) returns a generator producing {1, 3, 5, 7}. * start and end must have the same type. That type may be any integral or floating-point type or a user defined type satisfying these conditions: * It must be assignable (have operator=() defined). * It must have operator+(operator+(int-compatible type) for two-operand version). * It must have operator<() defined. Elements in the resulting sequences will also have that type. * Condition start < end must be satisfied in order for resulting sequences to contain any elements.
86525. If the type list contains only one type, you can write that type directly without Types<...>: TYPED_TEST_CASE(FooTest, int);

86526. Returns true iff the assertion succeeded. NOLINT
86527. Known limitations: we don't support passing an std::tr1::reference_wrapper<T> to make_tuple(). And we don't implement tie().
86528. A handy wrapper for AddRef.
86529. const_it
86530. This template class represents an IEEE floating-point number (either single-precision or double-precision, depending on the template parameters). The purpose of this class is to do more sophisticated number comparison. (Due to round-off error, etc, it's very unlikely that two floating-points will be equal exactly. Hence a naive comparison by the == operation often doesn't work.) Format of IEEE floating-point: The most-significant bit being the leftmost, an IEEE floating-point looks like sign_bit exponent_bits fraction_bits Here, sign_bit is a single bit that designates the sign of the number. For float, there are 8 exponent bits and 23 fraction bits. For double, there are 11 exponent bits and 52 fraction bits. More details can be found at http://en.wikipedia.org/wiki/IEEE_floating-point_standard. Template parameter: RawType: the raw floating-point type (either float or double)
86531. Returns the i-th test property. i can range from 0 to test_property_count() - 1. If i is not in that range, aborts the program.
86532. A copy constructor is required by the Standard to initialize object references from r-values.
86533. class TestResult
86534. pthread_key_create() requires DeleteThreadLocalValue() to have C-linkage. Therefore it cannot be templated to access ThreadLocal<T>. Hence the need for class ThreadLocalValueHolderBase.
86535. Complain about incorrect usage of Google Test facilities and terminate the program since we cannot guarantee correct test case setup and tear-down in this case.
86536. A function level attribute to disable ThreadSanitizer instrumentation.
86537. **comment:** When you need to test the private or protected members of a class, use the FRIEND_TEST macro to declare your tests as friends of the class. For example: class MyClass { private: void MyMethod(); FRIEND_TEST(MyClassTest, MyMethod); }; class MyClassTest : public testing::Test { // ... }; TEST_F(MyClassTest, MyMethod) { // Can call MyClass::MyMethod() here. }
- label:** test
86538. On Android, <regex.h> is only available starting with Gingerbread.
86539. By default, we assume that stream redirection is supported on all platforms except known mobile ones.
86540. Prints the address of the value. We use reinterpret_cast here as static_cast doesn't compile when T is a function type.
86541. When building against STLport with the Android NDK and with -frtti -fno-exceptions, the build fails at link time with undefined references to __cxa_bad_typeid. Note sure if STL or toolchain bug, so disable RTTI when detected.
86542. Definitions in the 'internal' and 'internal2' name spaces are subject to change without notice. DO NOT USE THEM IN USER CODE!
86543. The variables defined in the type-parameterized test macros are static as typically these macros are used in a .h file that can be #included in multiple translation units linked together.
86544. IsSubstring() and IsNotSubstring() are intended to be used as the first argument to {EXPECT,ASSERT}_PRED_FORMAT2(), not by themselves. They check whether needle is a substring of haystack (NULL is considered a substring of itself only), and return an appropriate error message when they fail. The {needle,haystack}_expr arguments are the stringified expressions that generated the two real arguments.
86545. Internal macro for implementing {EXPECT|ASSERT}_PRED_FORMAT3. Don't use this in your code.
86546. Adapts a native array to a read-only STL-style container. Instead of the complete STL container concept, this adaptor only implements members useful for Google Mock's container matchers. New members should be added as needed. To simplify the implementation, we only support Element being a raw type (i.e. having no top-level const or reference modifier). It's the client's responsibility to satisfy this requirement. Element can be an array type itself (hence multi-dimensional arrays are supported).
86547. Non-static methods
86548. A handy wrapper for ByRef.
86549. Runs SetUpTestCase() for this TestCase. This wrapper is needed for catching exceptions thrown from SetUpTestCase().
86550. NOLINT NOLINT NOLINT NOLINT
86551. **comment:** This helper reduces code bloat. If we instead put its logic inside the previous ArrayEq() function, arrays with different sizes would lead to different copies of the template code.
- label:** code-design
86552. Overload for ::std::tr1::tuple. Needed for printing function arguments, which are packed as tuples.
86553. Running count of death tests.
86554. Reinterprets a bit pattern as a floating-point number. This function is needed to test the AlmostEquals() method.
86555. Determines whether Google Test's own tr1 tuple implementation should be used.
86556. class CartesianProductHolder3
86557. class CartesianProductGenerator8
86558. for isspace, etc for ptrdiff_t
86559. scripts/fuse_gtest.py depends on gtest's own header being #included *unconditionally*. Therefore these #includes cannot be moved inside #if GTEST_HAS_PARAM_TEST. Copyright 2003 Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: Dan Egnor (egnor@google.com) A "smart" pointer type with reference tracking. Every pointer to a particular object is kept on a circular linked list. When the last pointer to an object is destroyed or reassigned, the object is deleted. Used properly, this deletes the object when the last reference goes away. There are several caveats: - Like all reference counting schemes, cycles lead to leaks. - Each smart pointer is actually two pointers (8 bytes instead of 4). - Every time a pointer is assigned, the entire list of pointers to that object is traversed. This class is therefore NOT SUITABLE when there will often be more than two or three pointers to a particular object. - References are only tracked as long as linked_ptr<> objects are copied. If a linked_ptr<> is converted to a raw pointer and back, BAD THINGS will happen (double deletion). A good use of this class is storing object references in STL containers. You can safely put linked_ptr<> in a vector<>. Other uses may not be as good. Note: If you use an incomplete type with linked_ptr<>, the class *containing* linked_ptr<> must have a constructor and destructor (even if they do nothing!). Bill Gibbons suggested we use something like this. Thread Safety: Unlike other linked_ptr implementations, in this implementation a linked_ptr object is thread-safe in the sense that: - it's safe to copy linked_ptr objects concurrently, - it's safe to copy *from* a linked_ptr and read its underlying raw pointer (e.g. via get()) concurrently, and - it's safe to write to two linked_ptrs that point to the same shared object concurrently. TODO(wan@google.com): rename this to safe_linked_ptr to avoid confusion with normal linked_ptr.
86560. Helper class for testing Google Test's multi-threading constructs. To use it, write: void ThreadFunc(int param) { /* Do things with param */ } Notification thread_can_start; ... // The thread_can_start parameter is optional; you can supply NULL. ThreadWithParam<int> thread(&ThreadFunc, 5, &thread_can_start); thread_can_start.Notify(); These classes are only for testing Google Test's own constructs. Do not use them in user tests, either directly or indirectly.
86561. C++11 specifies that <tuple> provides std::tuple. Some platforms still might not have it, however.
86562. No value parameter.
86563. C'tor. TestPartResult does NOT have a default constructor. Always use this constructor (with parameters) to create a TestPartResult object.
86564. **comment:** Stores the message describing the condition in case the expectation construct is not satisfied with the predicate's outcome. Referenced via a pointer to avoid taking too much stack frame space with test assertions.
- label:** code-design
86565. Keeps pairs of <Instantiation name, Sequence generator creation function> received from INSTANTIATE_TEST_CASE_P macros.
86566. Prints a value tersely: for a reference type, the referenced value (but not the address) is printed; for a (const) char pointer, the NUL-terminated string (but not the pointer) is printed.
86567. We must report iterator past the end of the range when either of the component iterators has reached the end of its range.
86568. Swap the contents of this AssertionResult with other.

86569. **comment:** A UnitTest consists of a vector of TestCases. This is a singleton class. The only instance of UnitTest is created when UnitTest::GetInstance() is first called. This instance is never deleted. UnitTest is not copyable. This class is thread-safe as long as the methods are called according to their specification.

label: code-design

86570. Removes the reference from a type if it is a reference type, otherwise leaves it unchanged. This is the same as tr1::remove_reference, which is not widely available yet.

86571. GCC 4.0+ implements tr1/tuple in the <tr1/tuple> header. This does not conform to the TR1 spec, which requires the header to be <tuple>.

86572. We only implement == and !=, as we don't have a need for the rest yet.

86573. Given two numbers in the sign-and-magnitude representation, returns the distance between them as an unsigned number.

86574. The following family of struct and struct templates are used to represent template lists. In particular, TemplatesN<T1, T2, ..., TN> represents a list of N templates (T1, T2, ..., and TN). Except for Templates0, every struct in the family has two member types: Head for the selector of the first template in the list, and Tail for the rest of the list.

86575. **comment:** Here's what happens when an ASSERT_DEATH* or EXPECT_DEATH* is executed: 1. It generates a warning if there is more than one active thread. This is because it's safe to fork() or clone() only when there is a single thread. 2. The parent process clone(s) a sub-process and runs the death test in it; the sub-process exits with code 0 at the end of the death test, if it hasn't exited already. 3. The parent process waits for the sub-process to terminate. 4. The parent process checks the exit code and error message of the sub-process. Examples: ASSERT_DEATH(server.SendMessage(56, "Hello"), "Invalid port number"); for (int i = 0; i < 5; i++) { EXPECT_DEATH(server.ProcessRequest(i), "Invalid request * in ProcessRequest()") << "Failed to die on request " << i; } ASSERT_EXIT(server.ExitNow(), ::testing::ExitedWithCode(0), "Exiting"); bool KilledBySIGHUP(int exit_code) { return WIFSIGNALED(exit_code) && WTERMSIG(exit_code) == SIGHUP; } ASSERT_EXIT(client.HangUpServer(), KilledBySIGHUP, "Hanging up!"); On the regular expressions used in death tests: On POSIX-compliant systems (*nix), we use the <regex.h> library, which uses the POSIX extended regex syntax. On other platforms (e.g. Windows), we only support a simple regex syntax implemented as part of Google Test. This limited implementation should be enough most of the time when writing death tests; though it lacks many features you can find in PCRE or POSIX extended regex syntax. For example, we don't support union ("x|y"), grouping ("(xy)"), brackets ("[xy]"), and repetition count ("x{5,7}"), among others. Below is the syntax that we do support. We chose it to be a subset of both PCRE and POSIX extended regex, so it's easy to learn wherever you come from. In the following: 'A' denotes a literal character, period (.), or a single \ escape sequence; 'x' and 'y' denote regular expressions; 'm' and 'n' are for natural numbers. c matches any literal character c \d matches any decimal digit \D matches any character that's not a decimal digit \f matches f \n matches \r \n matches \r \s matches any ASCII whitespace, including \n \S matches any character that's not a whitespace \t matches \t \w matches \w \w matches any letter, _ or decimal digit \W matches any character that \w doesn't match \c matches any literal character c, which must be a punctuation . matches any single character except \n A? matches 0 or 1 occurrences of A A* matches 0 or many occurrences of A A+ matches 1 or many occurrences of A ^ matches the beginning of a string (not that of each line) \$ matches the end of a string (not that of each line) xy matches x followed by y If you accidentally use PCRE or POSIX extended regex features not implemented by us, you will get a run-time failure. In that case, please try to rewrite your regular expression within the above syntax. This implementation is *not* meant to be as highly tuned or robust as a compiled regex library, but should perform well enough for a death test, which already incurs significant overhead by launching a child process. Known caveats: A "threadsafe" style death test obtains the path to the test program from argv[0] and re-executes it in the sub-process. For simplicity, the current implementation doesn't search the PATH when launching the sub-process. This means that the user must invoke the test program via a path that contains at least one path separator (e.g. path/to/foo_test and /absolute/path/to/bar_test are fine, but foo_test is not). This is rarely a problem as people usually don't put the test binary directory in PATH.

TODO(wan@google.com): make thread-safe death tests search the PATH.

label: code-design

86576. Copyright 2009 Google Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhangyong Wan)

86577. **comment:** A macro to disallow operator= This should be used in the private: declarations for a class.

label: code-design

86578. **comment:** Use this annotation at the end of a struct/class definition to prevent the compiler from optimizing away instances that are never used. This is useful when all interesting logic happens inside the c'tor and / or d'tor. Example: struct Foo { Foo() { ... } } GTEST_ATTRIBUTE_UNUSED_; Also use it after a variable or parameter declaration to tell the compiler the variable/parameter does not have to be used.

label: code-design

86579. Create a diff of the input strings in Unified diff format.

86580. Many linked_ptr operations may change p.link_ for some linked_ptr variable p in the same circle as this object. Therefore we need to prevent two such operations from occurring concurrently. Note that different types of linked_ptr objects can coexist in a circle (e.g. linked_ptr<Base>, linked_ptr<Derived1>, and linked_ptr<Derived2>). Therefore we must use a single mutex to protect all linked_ptr objects. This can create serious contention in production code, but is acceptable in a testing framework.

86581. Override this to define how to set up the environment.

86582. Name of the test case.

86583. Registers thread_local_instance as having value on the current thread. Returns a value that can be used to identify the thread from other threads.

86584. **comment:** How many ULP's (Units in the Last Place) we want to tolerate when comparing two numbers. The larger the value, the more error we allow. A 0 value means that two numbers must be exactly the same to be considered equal. The maximum error of a single floating-point operation is 0.5 units in the last place. On Intel CPU's, all floating-point calculations are done with 80-bit precision, while double has 64 bits. Therefore, 4 should be enough for ordinary use. See the following article for more details on ULP: <http://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/>

label: code-design

86585. class CartesianProductHolder2

86586. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Expands to the namespace name that the type-parameterized tests for the given type-parameterized test case are defined in. The exact name of the namespace is subject to change without notice.

86587. Dereferences the current iterator and provides (read-only) access to the pointed value. It is the caller's responsibility not to call Current() on an iterator equal to BaseGenerator()->End(). Used for implementing ParamGenerator<T>::operator*().

86588. Determines whether clone(2) is supported. Usually it will only be available on Linux, excluding Linux on the Itanium architecture. Also see <http://linux.die.net/man/2/clone>.

86589. Formats a comparison assertion (e.g. ASSERT_EQ, EXPECT_LT, and etc) operand to be used in a failure message. The type (but not value) of the other operand may affect the format. This allows us to print a char* as a raw pointer when it is compared against another char* or void*, and print it as a C string when it is compared against an std::string object, for example. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

86590. Implements typed tests.

86591. GTEST_HAS_POSIX_REGEX

86592. Allows the user to #include <tr1/functional> if he chooses to.

86593. We'll hold the text streamed to this object here.

86594. The user didn't tell us whether ::string is available, so we need to figure it out.

86595. **comment:** Used to print a value that is not an STL-style container when the user doesn't define PrintTo() for it.

label: code-design

86596. <regex.h> may not be available on this platform. Use our own simple regex implementation instead.

86597. Constructs an RE from a string. NOLINT

86598. The testing::internal::posix namespace holds wrappers for common POSIX functions. These wrappers hide the differences between Windows/MSVC and POSIX systems. Since some compilers define these standard functions as macros, the wrapper cannot have the same name as the wrapped function.

86599. GTESTUSES_SIMPLE_RE

86600. Helper class for testing Google Test's multi-threading constructs.

86601. **comment:** Replaces multiple consecutive separators with a single separator. For example, "bar///foo" becomes "bar/foo". Does not eliminate other redundancies that might be in a pathname involving "." or "..". A pathname with multiple consecutive separators may occur either through user error or as a result of some scripts or APIs that generate a pathname with a trailing separator. On other platforms the same API or script may NOT generate a pathname with a trailing "/". Then elsewhere that pathname may have another "/" and pathname components added to it, without checking for the separator already being there. The script language and operating system may allow paths like "foo/bar" but some of the functions in FilePath will not handle that correctly. In particular, RemoveTrailingPathSeparator() only removes one separator, and it is called in CreateDirectoriesRecursively() assuming that it will change a pathname from directory syntax (trailing separator) to filename syntax. On Windows this method also replaces the alternate path separator '/' with the primary path separator '\\', so that for example "bar\\\\\\foo" becomes "bar\\foo".

label: code-design

86602. GTEST_ELLIPSIS_NEEDS_POD_

86603. Names of the flags (needed for parsing Google Test flags).

86604. Prints the given array, omitting some elements when there are too many.

86605. StaticAssertTypeEqHelper is used by StaticAssertTypeEq defined in gtest.h. This template is declared, but intentionally undefined.

86606. class CartesianProductHolder9

86607. Message assignment is a semantic trick to enable assertion streaming; see the GTEST_MESSAGE_ macro below.

86608. We disallow copying Tests.

86609. Gets the number of failed tests.

86610. gcc's implementation of typeid(T).name() mangles the type name, so we have to demangle it.

86611. Ask the compiler to never inline a given function.

86612. This implementation of scoped_ptr is PARTIAL - it only contains enough stuff to satisfy Google Test's need.

86613. GCC 4.4.6

86614. **comment:** MutexBase and Mutex implement mutex on pthreads-based platforms.

label: requirement

86615. Returns a newly created InternalRunDeathTestFlag object with fields initialized from the GTEST_FLAG(internal_run_death_test) flag if the flag is specified; otherwise returns NULL.

86616. Returns true if any test in this test case should run.

86617. Gets the user supplied value.

86618. Gets the number of all test cases.

86619. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: wan@google.com (Zhanyong Wan), eefacm@gmail.com (Sean McAfee) The Google C++ Testing Framework (Google Test) This header file defines internal utilities needed for implementing death tests. They are subject to change without notice.

86620. Returns the exponent bits of this number.

86621. Returns true iff the test part non-fatally failed.

86622. **comment:** Gets the outcome of the test part.

label: test

86623. GTEST_HAS_PTHREAD

86624. Returns true if this test should run, that is if the test is not disabled (or it is disabled but the also_run_disabled_tests flag has been specified) and its full name matches the user-specified filter. Google Test allows the user to filter the tests by their full names. The full name of a test Bar in test case Foo is defined as "Foo.Bar". Only the tests that match the filter will run. A filter is a colon-separated list of glob (not regex) patterns, optionally followed by a '-' and a colon-separated list of negative patterns (tests to exclude). A test is run if it matches one of the positive patterns and does not match any of the negative patterns. For example, *A*:Foo.* is a filter that matches any string that contains the character 'A' or starts with "Foo".

86625. test_case

86626. Returns the standard listener responsible for the default console output. Can be removed from the listeners list to shut down default console output. Note that removing this object from the listener list with Release transfers its ownership to the caller and makes this function return NULL the next time.

86627. Determines whether RTTI is available.

86628. Tests that an exit code describes a normal exit with a given exit code.

86629. Fired after environment tear-down for each iteration of tests ends.

86630. Value-parameterized tests allow you to test your code with different parameters without writing multiple copies of the same test. Here is how you use value-parameterized tests:

86631. class ParameterizedTestCaseInfo

86632. Creates the test object, runs it, records its result, and then deletes it.

86633. UniversalPrinter<T>::Print(value, ostream_ptr) prints the given value to the given ostream. The caller must ensure that 'ostream_ptr' is not NULL, or the behavior is undefined. We define UniversalPrinter as a class template (as opposed to a function template), as we need to partially specialize it for reference types, which cannot be done with function templates.

86634. We cannot use UniversalPrint(value.first, os) here, as T1 may be a reference type. The same for printing value.second.

86635. This flag specifies the random number seed.

86636. Blacklist of patch releases of older branches:

86637. A helper class for creating scoped traces in user programs.

86638. Fired after a failed assertion or a SUCCEED() invocation.

86639. Returns true iff the current test has a non-fatal failure.

86640. A handy wrapper around RemoveConst that works when the argument T depends on template parameters.

86641. This is used internally by all instances of linked_ptr<T>. It needs to be a non-template class because different types of linked_ptr<T> can refer to the same object (linked_ptr<Superclass>(obj) vs linked_ptr<Subclass>(obj)). So, it needs to be possible for different types of linked_ptr to participate in the same circular linked list, so we need a single class type here. DO NOT USE THIS CLASS DIRECTLY YOURSELF. Use linked_ptr<T>.

86642. A function to convert T* into linked_ptr<T>. Doing e.g. make_linked_ptr(new FooBarBaz<type>(arg)) is a shorter notation for linked_ptr<FooBarBaz<type> > (new FooBarBaz<type>(arg))

86643. This macro is for implementing ASSERT/EXPECT_DEBUG_DEATH when compiled in NDEBUG mode. In this case we need the statements to be executed, the regex is ignored, and the macro must accept a streamed message even though the message is never printed.

86644. Downcasts the pointer of type Base to Derived. Derived must be a subclass of Base. The parameter MUST point to a class of type Derived, not any subclass of it. When RTTI is available, the function performs a runtime check to enforce this.

86645. T is not a function type. We just call << to print p, relying on ADL to pick up user-defined << for their pointer types, if any.

86646. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_PARAM_UTIL_GENERATED_H_

86647. **comment:** The 'Types' template argument below must have spaces around it since some compilers may choke on '>>' when passing a template instance (e.g. Types<int>)

label: code-design

86648. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. A class that enables one to stream messages to assertion macros

86649. Internal macro for implementing {EXPECT|ASSERT}_PRED2. Don't use this in your code.
86650. GTEST_HAS_DEATH_TEST
86651. We have to put the 'public' section after the 'private' section, or MSVC refuses to compile the code.
86652. Used to define __ANDROID_API__ matching the target NDK API level. NOLINT
86653. Notifies all threads created with this notification to start. Must be called from the controller thread.
86654. A copyable object representing the result of a test part (i.e. an assertion or an explicit FAIL(), ADD_FAILURE(), or SUCCESS()). Don't inherit from TestPartResult as its destructor is not virtual.
86655. Defines types for pointers to functions that set up and tear down a test case.
86656. __CYGWIN__
86657. FormatForComparison<ToPrint, OtherOperand>::Format(value) formats a value of type ToPrint that is an operand of a comparison assertion (e.g. ASSERT_EQ). OtherOperand is the type of the other operand in the comparison, and is used to help determine the best way to format the value. In particular, when the value is a C string (char pointer) and the other operand is an STL string object, we want to format the C string as a string, since we know it is compared by value with the string object. If the value is a char pointer but the other operand is not an STL string object, we don't know whether the pointer is supposed to point to a NUL-terminated string, and thus want to print it as a pointer to be safe. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.
86658. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: wan@google.com (Zhangyong Wan), eefacm@gmail.com (Sean McAfee) The Google C++ Testing Framework (Google Test) This header file declares functions and macros used internally by Google Test. They are subject to change without notice.
86659. A function that returns an instance of appropriate generator type.
86660. This class provides implementation of TeastFactoryBase interface. It is used in TEST and TEST_F macros.
86661. D'tor
86662. ParamIterator assumes ownership of the impl_ pointer.
86663. Environment variables which we programmatically clear will be set to the empty string rather than unset (NULL). Handle that case.
86664. Like ASSERT_DEATH, but continues on to successive tests in the test case, if any:
86665. Copyright 2008 Google Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: vlad@google.com (Vlad Losev)
86666. NULL if not a type-parameterized test NULL if not a value-parameterized test
86667. The helper function for {ASSERT|EXPECT}_STREQ. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.
86668. Returns true iff the test failed.
86669. **comment:** Increments the number of death tests encountered in this test so far.
- label:** test
86670. Integer types of known sizes.
86671. **comment:** When you upcast (that is, cast a pointer from type Foo to type SuperclassOfFoo), it's fine to use ImplicitCast_<>, since upcasts always succeed. When you downcast (that is, cast a pointer from type Foo to type SubclassOfFoo), static_cast_<> isn't safe, because how do you know the pointer is really of type SubclassOfFoo? It could be a bare Foo, or of type DifferentSubclassOfFoo. Thus, when you downcast, you should use this macro. In debug mode, we use dynamic_cast_<> to double-check the downcast is legal (we die if it's not). In normal mode, we do the efficient static_cast_<> instead. Thus, it's important to test in debug mode to make sure the cast is legal! This is the only place in the code we should use dynamic_cast_<>. In particular, you SHOULDN'T be using dynamic_cast_<> in order to do RTTI (eg code like this: if (dynamic_cast<Subclass1>(foo)) HandleASubclass1Object(foo); if (dynamic_cast<Subclass2>(foo)) HandleASubclass2Object(foo); You should design the code some other way not to need this. This relatively ugly name is intentional. It prevents clashes with similar functions users may have (e.g., down_cast). The internal namespace alone is not enough because the function can be found by ADL. use like this: DownCast_<T*>(foo); so we only accept pointers Ensures that To is a sub-type of From *. This test is here only for compile-time type checking, and has no overhead in an optimized build at run-time, as it will be optimized away completely.
- label:** code-design
86672. MS C++ compiler emits warning when a conditional expression is compile time constant. In some contexts this warning is false positive and needs to be suppressed. Use the following two macros in such cases: GTEST_INTENTIONAL_CONST_COND_PUSH_() while (true) { GTEST_INTENTIONAL_CONST_COND_POP_0 }
86673. Gets the number of all test cases that contain at least one test that should run.
86674. The default value for each thread.
86675. Returns the type ID of ::testing::Test. Always call this instead of GetTypeId< ::testing::Test>() to get the type ID of ::testing::Test, as the latter may give the wrong result due to a suspected linker bug when compiling Google Test as a Mac OS X framework.
86676. class CartesianProductGenerator4
86677. class CartesianProductHolder6
86678. We rely on kStaticMutex being 0 as it is to what the linker initializes type_ in static mutexes. critical_section_ will be initialized lazily in ThreadSafeLazyInit().
86679. Implements thread-local storage on pthreads-based systems.
86680. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Helper classes providing Combine() with polymorphic features. They allow casting CartesianProductGeneratorN<T> to ParamGenerator<U> if T is convertible to U.
86681. This flags control whether Google Test prints the elapsed time for each test.
86682. Only use C++11 library features if the library provides them.
86683. Macros that execute statement and check that it doesn't generate new fatal failures in the current thread. * {ASSERT|EXPECT}_NO_FATAL_FAILURE(statement); Examples: EXPECT_NO_FATAL_FAILURE(Process()); ASSERT_NO_FATAL_FAILURE(Process()) << "Process() failed";
86684. class CartesianProductGenerator7::Iterator
86685. Gets the time of the test program start, in ms from the start of the UNIX epoch.
86686. These predicate format functions work on floating-point values, and can be used in {ASSERT|EXPECT}_PRED_FORMAT2*(), e.g. EXPECT_PRED_FORMAT2(testing::DoubleLE, Foo(), 5.0);
86687. The user told us that ::std::string isn't available.
86688. In order to avoid having to include <windows.h>, use forward declaration assuming CRITICAL_SECTION is a typedef of _RTL_CRITICAL_SECTION. This assumption is verified by WindowsTypesTest.CRITICAL_SECTIONIs_RTL_CRITICAL_SECTION.
86689. Gets the vector of TestPartResults.

86690. **comment:** TODO(vlidl@google.com): Remove this after making sure no clients use it. Deprecated; please use message() instead.

label: code-design

86691. Some libraries overload << for STL containers. These overloads are defined in the global namespace instead of ::std. C++'s symbol lookup rule (i.e. Koenig lookup) says that these overloads are visible in either the std namespace or the global namespace, but not other namespaces, including the testing namespace which Google Test's Message class is in. To allow STL containers (and other types that has a << operator defined in the global namespace) to be used in Google Test assertions, testing::Message must access the custom << operator from the global namespace. With this using declaration, overloads of << defined in the global namespace and those visible via Koenig lookup are both exposed in this function.

86692. sam represents a positive number.

86693. A helper class that aborts a death test when it's deleted.

86694. A function level attribute to disable AddressSanitizer instrumentation.

86695. Used in the Values() function to provide polymorphic capabilities.

86696. Pointer to the function that tears down the test case.

86697. Name of the parameter type, or NULL if this is not a typed or a type-parameterized test.

86698. This flag controls whether Google Test includes Google Test internal stack frames in failure stack traces.

86699. Gets the summary of the failure message.

86700. Listener responsible for the standard result output.

86701. Import tuple and friends into the ::testing namespace. It is part of our interface, having them in ::testing allows us to change their types as needed.

86702. deprecated function

86703. Registers and returns a global test environment. When a test program is run, all global test environments will be set-up in the order they were registered. After all tests in the program have finished, all global test environments will be torn-down in the *reverse* order they were registered. The UnitTest object takes ownership of the given environment. This method can only be called from the main thread.

86704. Factory interface for death tests. May be mocked out for testing.

86705. Gets the content of the stringstream's buffer as an std::string. Each '\0' character in the buffer is replaced with "\0".

86706. **comment:** We need the following helper functions only for their types. They have no implementations.

label: code-design

86707. **comment:** This interface knows how to report a test part result.

label: test

86708. Sets the elapsed time.

86709. fd

86710. Returns true iff test failed.

86711. This prevents <boost/tr1/detail/config.hpp>, which defines BOOST_HAS_TR1_TUPLE, from being #included by Boost's <tuple>.

86712. A static mutex may be used before main() is entered. It may even be used before the dynamic initialization stage. Therefore we must be able to initialize a static mutex object at link time. This means MutexBase has to be a POD and its member variables have to be public.

86713. The specialization for size 8.

86714. Formats an int value as "%X".

86715. Combine() allows the user to combine two or more sequences to produce values of a Cartesian product of those sequences' elements. Synopsis: Combine(gen1, gen2, ..., genN) - returns a generator producing sequences with elements coming from the Cartesian product of elements from the sequences generated by gen1, gen2, ..., genN. The sequence elements will have a type of tuple<T1, T2, ..., TN> where T1, T2, ..., TN are the types of elements from sequences produced by gen1, gen2, ..., genN. Combine can have up to 10 arguments. This number is currently limited by the maximum number of elements in the tuple implementation used by Google Test. Example: This will instantiate tests in test case AnimalTest each one with the parameter values tuple("cat", BLACK), tuple("cat", WHITE), tuple("dog", BLACK), and tuple("dog", WHITE): enum Color { BLACK, GRAY, WHITE }; class AnimalTest : public testing::TestWithParam<tuple<const char*, Color>> { ... }; TEST_P(AnimalTest, AnimalLooksNice) { ... } INSTANTIATE_TEST_CASE_P(AnimalVariations, AnimalTest, Combine(Values("cat", "dog"), Values(BLACK, WHITE))); This will instantiate tests in FlagDependentTest with all variations of two Boolean flags: class FlagDependentTest : public testing::TestWithParam<tuple<bool, bool>> { virtual void SetUp() { // Assigns external_flag_1 and external_flag_2 values from the tuple. tie(external_flag_1, external_flag_2) = GetParam(); } }; TEST_P(FlagDependentTest, TestFeature1) { // Test your code using external_flag_1 and external_flag_2 here. } INSTANTIATE_TEST_CASE_P(TwoBoolSequence, FlagDependentTest, Combine(Bool(), Bool()));

86716. **comment:** Defines a test. The first parameter is the name of the test case, and the second parameter is the name of the test within the test case. The convention is to end the test case name with "Test". For example, a test case for the Foo class can be named FooTest. Test code should appear between braces after an invocation of this macro. Example: TEST(FooTest, InitializesCorrectly) { Foo foo; EXPECT_TRUE(foo.StatusIsOK()); }

label: code-design

86717. Then, use TYPED_TEST_P() to define as many type-parameterized tests for this type-parameterized test case as you want.

86718. Determines whether to support Combine(). This only makes sense when value-parameterized tests are enabled. The implementation doesn't work on Sun Studio since it doesn't understand templated conversion operators.

86719. Class iterating over elements provided by an implementation of ParamGeneratorInterface<T>. It wraps ParamIteratorInterface<T> and implements the const forward iterator concept.

86720. For now, the XML report includes all tests matching the filter. In the future, we may trim tests that are excluded because of sharding.

86721. This constructor intentionally does nothing. It relies on type_ being statically initialized to 0 (effectively setting it to kStatic) and on ThreadSafeLazyInit() to lazily initialize the rest of the members.

86722. Types of SetUpTestCase() and TearDownTestCase() functions.

86723. The data type used to store the actual floating-point number.

86724. Macros for indicating success/failure in test code.

86725. Generator interface definition

86726. Wraps ParamGeneratorInterface<T> and provides general generator syntax compatible with the STL Container concept. This class implements copy initialization semantics and the contained ParamGeneratorInterface<T> instance is shared among all copies of the original object. This is possible because that instance is immutable.

86727. **comment:** Advance should not be called on beyond-of-range iterators so no component iterators must be beyond end of range, either.

label: code-design

86728. We define two overloaded versions of Compare(). The first version will be picked when the second argument to ASSERT_EQ() is NOT a pointer, e.g. ASSERT_EQ(0, AnIntFunction()) or EXPECT_EQ(false, a_bool).

86729. Implements the helper function for {ASSERT|EXPECT}_GE

86730. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhangyong Wan) The Google C++ Testing Framework (Google Test) This header file defines the Message class. IMPORTANT NOTE: Due to limitation of the C++ language, we have to leave some internal implementation details in this header file. They are clearly marked by comments like this: // INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM. Such code is NOT meant to be used by a user directly, and is subject to CHANGE WITHOUT NOTICE. Therefore DO NOT DEPEND ON IT in a user program!

86731. **comment:** The tests from the instantiation above will have these names: * AnotherInstantiationName/FooTest.DoesBlah/0 for "cat" *

AnotherInstantiationName/FooTest.DoesBlah/1 for "dog" * AnotherInstantiationName/FooTest.HasBlahBlah/0 for "cat" *

AnotherInstantiationName/FooTest.HasBlahBlah/1 for "dog" Please note that INSTANTIATE_TEST_CASE_P will instantiate all tests in the given test case,

whether their definitions come before or AFTER the INSTANTIATE_TEST_CASE_P statement. Please also note that generator expressions (including parameters to the generators) are evaluated in InitGoogleTest(), after main() has started. This allows the user on one hand, to adjust generator parameters in order to dynamically determine a set of tests to run and on the other hand, give the user a chance to inspect the generated tests with Google Test reflection API before RUN_ALL_TESTS() is executed. You can see samples/sample7_unittest.cc and samples/sample8_unittest.cc for more examples. In the future, we plan to publish the API for defining new parameter generators. But for now this interface remains part of the internal implementation and is subject to change. A parameterized test fixture must be derived from testing::Test and from testing::WithParamInterface<T>, where T is the type of the parameter values. Inheriting from TestWithParam<T> satisfies that requirement because TestWithParam<T> inherits from both Test and WithParamInterface. In more complicated hierarchies, however, it is occasionally useful to inherit separately from Test and WithParamInterface. For example:

label: code-design

86732. ADD_FAILURE unconditionally adds a failure to the current test. SUCCEED generates a success - it doesn't automatically make the current test successful, as a test is only successful when it has no failure. EXPECT_* verifies that a certain condition is satisfied. If not, it behaves like ADD_FAILURE. In particular: EXPECT_TRUE verifies that a Boolean condition is true, EXPECT_FALSE verifies that a Boolean condition is false. FAIL and ASSERT_* are similar to ADD_FAILURE and EXPECT_*, except that they will also abort the current function on failure. People usually want the fail-fast behavior of FAIL and ASSERT_*, but those writing data-driven tests often find themselves using ADD_FAILURE and EXPECT_* more.

86733. As a C-function, ThreadFuncWithCLinkage cannot be templated itself. Consequently, it cannot select a correct instantiation of ThreadWithParam in order to call its Run(). Introducing ThreadWithParamBase as a non-templated base class for ThreadWithParam allows us to bypass this problem.

86734. namespace internal namespace testing

86735. 40302 means version 4.3.2.

86736. test_names

86737. Expands to the name of the class that implements the given test.

86738. We disallow copying TestCases.

86739. **comment:** A class for indicating whether an assertion was successful. When the assertion wasn't successful, the AssertionResult object remembers a non-empty message that describes how it failed. To create an instance of this class, use one of the factory functions (AssertionSuccess() and AssertionFailure()). This class is useful for two purposes: 1. Defining predicate functions to be used with Boolean test assertions EXPECT_TRUE/EXPECT_FALSE and their ASSERT_ counterparts 2. Defining predicate-format functions to be used with predicate assertions (ASSERT_PRED_FORMAT*, etc). For example, if you define IsEven predicate: testing::AssertionResult IsEven(int n) { if ((n % 2) == 0) return testing::AssertionSuccess(); else return testing::AssertionFailure() << n << " is odd"; } Then the failed expectation EXPECT_TRUE(IsEven(Fib(5))) will print the message Value of: IsEven(Fib(5)) Actual: false (5 is odd) Expected: true instead of a more opaque Value of: IsEven(Fib(5)) Actual: false Expected: true in case IsEven is a simple Boolean predicate. If you expect your predicate to be reused and want to support informative messages in EXPECT_FALSE and ASSERT_FALSE (negative assertions show up about half as often as positive ones in our tests), supply messages for both success and failure cases: testing::AssertionResult IsEven(int n) { if ((n % 2) == 0) return testing::AssertionSuccess() << n << " is even"; else return testing::AssertionFailure() << n << " is odd"; } Then a statement EXPECT_FALSE(IsEven(Fib(6))) will print Value of: IsEven(Fib(6)) Actual: true (8 is even) Expected: false NB: Predicates that support negative Boolean assertions have reduced performance in positive ones so be careful not to use them in tests that have lots (tens of thousands) of positive Boolean assertions. To use this class with EXPECT_PRED_FORMAT assertions such as: // Verifies that Foo() returns an even number. EXPECT_PRED_FORMAT1(IsEven, Foo()); you need to define: testing::AssertionResult IsEven(const char* expr, int n) { if ((n % 2) == 0) return testing::AssertionSuccess(); else return testing::AssertionFailure() << "Expected: " << expr << " is even\n Actual: it's " << n; } If Foo() returns 5, you will see the following message: Expected: Foo() is even Actual: it's 5

label: code-design

86740. Makes a failed assertion result with the given failure message. Deprecated; use AssertionFailure() << msg.

86741. Returns the i-th test among all the tests. i can range from 0 to total_test_count() - 1. If i is not in that range, returns NULL.

86742. First, define a fixture class template. It should be parameterized by a type. Remember to derive it from testing::Test.

86743. Returns the i-th test part result among all the results. i can range from 0 to test_property_count() - 1. If i is not in that range, aborts the program.

86744. gcc and clang define __GXX_EXPERIMENTAL_CXX0X__ when -std={c,gnu}++{0x,11} is passed. The C++11 standard specifies a value for __cplusplus, and recent versions of clang, gcc, and probably other compilers set that too in C++11 mode.

86745. Returns true if FilePath ends with a path separator, which indicates that it is intended to represent a directory. Returns false otherwise. This does NOT check that a directory (or file) actually exists.

86746. An Environment object is capable of setting up and tearing down an environment. You should subclass this to define your own environment(s). An Environment object does the set-up and tear-down in virtual methods SetUp() and TearDown() instead of the constructor and the destructor, as: 1. You cannot safely throw from a destructor. This is a problem as in some cases Google Test is used where exceptions are enabled, and we may want to implement ASSERT_* using exceptions where they are available. 2. You cannot use ASSERT_* directly in a constructor or destructor.

86747. # of bits in a number.

86748. **comment:** The pure interface class that all value-parameterized tests inherit from. A value-parameterized class must inherit from both ::testing::Test and ::testing::WithParamInterface. In most cases that just means inheriting from ::testing::TestWithParam, but more complicated test hierarchies may need to inherit from Test and WithParamInterface at different levels. This interface has support for accessing the test parameter value via the GetParam() method. Use it with one of the parameter generator defining functions, like Range(), Values(), ValuesIn(), Bool(), and Combine(). class FooTest : public ::testing::TestWithParam<int> { protected: FooTest() { // Can use GetParam() here. } virtual ~FooTest() { // Can use GetParam() here. } virtual void SetUp() { // Can use GetParam() here. } virtual void TearDown() { // Can use GetParam() here. } }; TEST_P(FooTest, DoesBar) { // Can use GetParam() method here. Foo foo; ASSERT_TRUE(foo.DoesBar(GetParam())); } INSTANTIATE_TEST_CASE_P(OneToTenRange, FooTest, ::testing::Range(1, 10));

label: code-design

86749. Base case.

86750. The empty template list.

86751. sam represents a negative number.

86752. **comment:** We are on Windows CE, which has no environment variables. To prevent 'unused argument' warning.

label: code-design

86753. Fired after environment set-up for each iteration of tests ends.

86754. prefix

86755. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_DEATH_TEST_INTERNAL_H_

86756. Returns true if pathname describes a directory in the file-system that exists.

86757. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Generates values from the Cartesian product of values produced by the argument generators.

86758. A class representing the parsed contents of the --gtest_internal_run_death_test flag, as it existed when RUN_ALL_TESTS was called.

86759. class CartesianProductGenerator10

86760. Two predicate classes that can be used in {ASSERT,EXPECT}_EXIT*:

86761. Releases this mutex.

86762. When this flag is specified, tests' order is randomized on every iteration.

86763. The user didn't tell us whether ::wstring is available, so we need to figure it out.

86764. Generates a fatal failure with a generic message.

86765. The template "selector" struct TemplateSel<Tmpl> is used to represent Tmpl, which must be a class template with one type parameter, as a type.

TemplateSel<Tmpl>::Bind<T>::type is defined as the type Tmpl<T>. This allows us to actually instantiate the template "selected" by TemplateSel<Tmpl>. This trick is necessary for simulating typeid for class templates, which C++ doesn't support directly.

86766. **comment:** Internal utilities ----- The following macros and utilities are for Google Test's INTERNAL use only. Code outside Google Test MUST NOT USE THEM DIRECTLY. Macros for basic C++ coding: GTEST_AMBIGUOUS_ELSE_BLOCKER_- for disabling a gcc warning.

GTEST_ATTRIBUTE_UNUSED_- declares that a class' instances or a variable don't have to be used. GTEST_DISALLOW_ASSIGN_- disables operator=.

GTEST_DISALLOW_COPY_AND_ASSIGN_- disables copy ctor and operator=. GTEST_MUST_USE_RESULT_- declares that a function's result must be used. GTEST_INTENTIONAL_CONST_COND_PUSH_- start code section where MSVC C4127 is suppressed (constant conditional).

GTEST_INTENTIONAL_CONST_COND_POP_- finish code section where MSVC C4127 is suppressed. C++11 feature wrappers: testing::internal::move - portability wrapper for std::move. Synchronization: Mutex, MutexLock, ThreadLocal, GetThreadCount() - synchronization primitives. Template meta

programming: is_pointer - as in TR1; needed on Symbian and IBM XL C/C++ only. IteratorTraits - partial implementation of std::iterator_traits, which is not available in libCstd when compiled with Sun C++. Smart pointers: scoped_ptr - as in TR2. Regular expressions: RE - a simple regular expression class using the POSIX Extended Regular Expression syntax on UNIX-like platforms, or a reduced regular exception syntax on other platforms, including Windows. Logging:

GTEST_LOG_() - logs messages at the specified severity level. LogToStderr() - directs all log messages to stderr. FlushInfoLog() - flushes informational log messages. Stdout and stderr capturing: CaptureStdout() - starts capturing stdout. GetCapturedStdout() - stops capturing stdout and returns the captured string. CaptureStderr() - starts capturing stderr. GetCapturedStderr() - stops capturing stderr and returns the captured string. Integer types: TypeWithSize - maps an integer to a int type. Int32, UInt32, Int64, UInt64, TimeInMillis - integers of known sizes. BiggestInt - the biggest signed integer type. Command-line utilities: GTEST_DECLARE_*() - declares a flag. GTEST_DEFINE_*() - defines a flag. GetInjectableArgs() - returns the command line as a vector of strings. Environment variable utilities: GetEnv() - gets the value of an environment variable. BoolFromGTestEnv() - parses a bool environment variable. Int32FromGTestEnv() - parses an Int32 environment variable. StringFromGTestEnv() - parses a string environment variable.

label: code-design

86767. Defines this to true iff Google Test can use POSIX regular expressions.

86768. Returns the name of the parameter type, or NULL if this is not a typed or a type-parameterized test.

86769. Type and function utilities for implementing parameterized tests. This file is generated by a SCRIPT. DO NOT EDIT BY HAND! Currently Google Test supports at most 50 arguments in Values, and at most 10 arguments in Combine. Please contact googletestframework@googlegroups.com if you need more. Please note that the number of arguments to Combine is limited by the maximum arity of the implementation of tuple which is currently set at 10.

86770. Determines whether the system compiler uses UTF-16 for encoding wide strings.

86771. When this flag is set with a "host:port" string, on supported platforms test results are streamed to the specified port on the specified host machine.

86772. **comment:** Type-parameterized tests are abstract test patterns parameterized by a type. Compared with typed tests, type-parameterized tests allow you to define the test pattern without knowing what the type parameters are. The defined pattern can be instantiated with different types any number of times, in any number of translation units. If you are designing an interface or concept, you can define a suite of type-parameterized tests to verify properties that any valid implementation of the interface/concept should have. Then, each implementation can easily instantiate the test suite to verify that it conforms to the requirements, without having to write similar tests repeatedly. Here's an example:

label: test

86773. is_pointer

86774. **comment:** We put our data in a struct so that the size of the AssertHelper class can be as small as possible. This is important because gcc is incapable of re-using stack space even for temporary variables, so every EXPECT_EQ reserves stack space for another AssertHelper.

label: code-design

86775. **comment:** Separate the error generating code from the code path to reduce the stack frame size of CmpHelperEQ. This helps reduce the overhead of some sanitizers when calling EXPECT_* in a tight loop.

label: code-design

86776. GTEST_OS_LINUX

86777. IWYU pragma: export

86778. Forward declarations of ValuesIn(), which is implemented in include/gtest/gtest-param-test.h.

86779. 6.1.3.5 Relational operators

86780. Constructs a TestInfo object. The newly constructed instance assumes ownership of the factory object.

86781. !GTEST_OS_WINDOWS_MOBILE && !GTEST_OS_SYMBIAN GTEST_HAS_STREAM_REDIRECTION

86782. Asserts that val1 is less than, or almost equal to, val2. Fails otherwise. In particular, it fails if either val1 or val2 is NaN.

86783. Converts a streamable value to an std::string. A NULL pointer is converted to "(null)". When the input value is a ::string, ::std::string, ::wstring, or ::std::wstring object, each NUL character in it is replaced with "\\0".

86784. A handy wrapper around RemoveReference that works when the argument T depends on template parameters.

86785. This default version is called when kTypeKind is kOtherType.

86786. A cached value of *iterator_. We keep it here to allow access by pointer in the wrapping iterator's operator->(). value_ needs to be mutable to be accessed in Current(). Use of scoped_ptr helps manage cached value's lifetime, which is bound by the lifespan of the iterator itself.

86787. **comment:** Compile-time assertion for type equality. StaticAssertTypeEq<type1, type2>() compiles iff type1 and type2 are the same type. The value it returns is not interesting. Instead of making StaticAssertTypeEq a class template, we make it a function template that invokes a helper class template. This prevents a user from misusing StaticAssertTypeEq<T1, T2> by defining objects of that type. CAVEAT: When used inside a method of a class template, StaticAssertTypeEq<T1, T2>() is effective ONLY IF the method is instantiated. For example, given: template <typename T> class Foo { public: void Bar() { testing::StaticAssertTypeEq<int, T>(); } }; the code: void Test1() { Foo<bool> foo; } will NOT generate a compiler error, as Foo<bool>::Bar() is never actually instantiated. Instead, you need: void Test2() { Foo<bool> foo; foo.Bar(); } to cause a compiler error.

label: code-design

86788. Returns true if the given test should run.

86789. Waits for the death test to finish and returns its status.

86790. By default, print C string as pointers to be safe, as we don't know whether they actually point to a NUL-terminated string.

86791. If input name has a trailing separator character, removes it and returns the name, otherwise return the name string unmodified. On Windows platform, uses \ as the separator, other platforms use /.

86792. GTEST_HAS_GLOBAL_WSTRING

86793. This version will be picked when the second argument to ASSERT_EQ() is a pointer, e.g. ASSERT_EQ(NULL, a_pointer).

86794. The GNU compiler emits a warning if nested "if" statements are followed by an "else" statement and braces are not used to explicitly disambiguate the "else" binding. This leads to problems with code like: if (gate) ASSERT_*(condition) << "Some message"; The "switch (0) case 0:" idiom is used to suppress this.

86795. Now the tricky part: you need to register all test patterns before you can instantiate them. The first argument of the macro is the test case name; the rest are the names of the tests in this test case.

86796. A helper for suppressing warnings on constant condition. It just returns 'condition'.

86797. Use this function in main() to run all tests. It returns 0 if all tests are successful, or 1 otherwise. RUN_ALL_TESTS() should be invoked after the command line has been parsed by InitGoogleTest(). This function was formerly a macro; thus, it is in the global namespace and has an all-caps name.

86798. __has_feature(thread_sanitizer)

86799. Predicate-formatters for implementing the HRESULT checking macros {ASSERT|EXPECT}_HRESULT_{SUCCEEDED|FAILED} We pass a long instead of HRESULT to avoid causing an include dependency for the HRESULT type.

86800. For other compilers, we assume exceptions are disabled to be conservative.

86801. Overloads for ::wstring and ::std::wstring.

86802. Returns true iff the test passed (i.e. no test part failed).

86803. Used in the EXPECT_TRUE/FALSE(bool_expression). T must be contextually convertible to bool. The second parameter prevents this overload from being considered if the argument is implicitly convertible to AssertionResult. In that case we want AssertionResult's copy constructor to be used.

86804. Helper function for implementing {EXPECT|ASSERT}_PRED3. Don't use this in your code.

86805. 6.1.4 Pairs. Unimplemented.

86806. class RangeGenerator::Iterator

86807. Sets a new value, overriding the one supplied in the constructor.

86808. NDEBUG for EXPECT_DEBUG_DEATH GTEST_HAS_DEATH_TEST

86809. GTEST_INCLUDE_GTEST_GTEST_PROD_H Copyright 2008, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: mheule@google.com (Markus Heule)

86810. WINAPI_FAMILY defined but no known partition matched. Default to desktop.

86811. The helper function for {ASSERT|EXPECT}_EQ.
86812. Returns repeater that broadcasts the TestEventListener events to all subscribers.
86813. ParamGeneratorInterface<T> is the binary interface to access generators defined in other translation units.
86814. The following macros are useful for writing death tests.
86815. Overloads for C strings.
86816. Assume no SEH.
86817. **comment:** The result of a single Test. This includes a list of TestPartResults, a list of TestProperties, a count of how many death tests there are in the Test, and how much time it took to run the Test. TestResult is not copyable.
label: test
86818. Inspired by boost/config/stdcpp3.hpp, http://gcc.gnu.org/gcc-4.2/changes.html and
<http://gcc.gnu.org/onlinedocs/libstdc++/manual/bk01pt01ch01.html#manual.intro.status.standard.200x>
86819. Represents time in milliseconds.
86820. Assumes one of the above roles.
86821. A copy of all command line arguments. Set by InitGoogleTest().
86822. Finally, you are free to instantiate the pattern with the types you want. If you put the above code in a header file, you can #include it in multiple C++ source files and instantiate it multiple times. To distinguish different instances of the pattern, the first argument to the INSTANTIATE_* macro is a prefix that will be added to the actual test case name. Remember to pick unique prefixes for different instances.
86823. Compares two wide C strings. Returns true iff they have the same content. Unlike wcscmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL C string, including the empty string.
86824. With this overloaded version, we allow anonymous enums to be used in {ASSERT|EXPECT}_EQ when compiled with gcc 4, as anonymous enums can be implicitly cast to BiggestInt. Even though its body looks the same as the above version, we cannot merge the two, as it will make anonymous enums unhappy.
86825. Internal macro for implementing {EXPECT|ASSERT}_PRED4. Don't use this in your code.
86826. Determines whether to support type-driven tests.
86827. **comment:** The user didn't tell us whether ::std::wstring is available, so we need to figure it out. TODO(wan@google.com): uses autoconf to detect whether ::std::wstring is available.
label: requirement
86828. Returns the test name.
86829. A simple C++ wrapper for <regex.h>. It uses the POSIX Extended Regular Expression syntax.
86830. This flag temporary enables the disabled tests.
86831. **comment:** Gets the number of disabled tests that will be reported in the XML report.
label: test
86832. class RangeGenerator
86833. Copyright 2008 Google Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan)
86834. Typed (aka type-driven) tests repeat the same test for types in a list. You must know which types you want to test with when writing typed tests. Here's how you do it:
86835. namespace tr1 namespace std
86836. Returns true iff this test will appear in the XML report.
86837. This flag enables using colors in terminal output. Available values are "yes" to enable colors, "no" (disable colors), or "auto" (the default) to let Google Test decide.
86838. Returns the random seed used at the start of the current test run.
86839. The following line prevents this overload from being considered if T2 is not a pointer type. We need this because ASSERT_EQ(NULL, my_ptr) expands to Compare("", "", NULL, my_ptr), which requires a conversion to match the Secret* in the other overload, which would otherwise make this template match better.
86840. Not meant to be instantiated. class String
86841. This flag controls the style of death tests. Valid values are "threadsafe", meaning that the death test child process will re-execute the test binary from the start, running only a single death test, or "fast", meaning that the child process will execute the test logic immediately after forking.
86842. The default case.
86843. Defines scoped_ptr.
86844. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. ParameterizedTestCaseRegistry contains a map of ParameterizedTestCaseInfoBase classes accessed by test case names. TEST_P and INSTANTIATE_TEST_CASE_P macros use it to locate their corresponding ParameterizedTestCaseInfo descriptors.
86845. **comment:** A macro to disallow copy constructor and operator= This should be used in the private: declarations for a class.
label: code-design
86846. The elapsed time, in milliseconds.
86847. Includes the auto-generated header that implements a family of generic predicate assertion macros. Copyright 2006, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
86848. Protects mutable state of the property vector and of owned properties, whose values may be updated.
86849. class FilePath
86850. Next, associate a list of types with the test case, which will be repeated for each type in the list. The typedef is necessary for the macro to parse correctly.
86851. D'tor. Do not inherit from TestResult.
86852. INTERNAL IMPLEMENTATION - DO NOT USE. GTEST_CHECK_ is an all-mode assert. It aborts the program if the condition is not satisfied. Synopsis: GTEST_CHECK_(boolean_condition); or GTEST_CHECK_(boolean_condition) << "Additional message"; This checks the condition and if the condition is not satisfied it prints message about the condition violation, including the condition itself, plus additional message streamed into it, if any, and then it aborts the program. It aborts the program irrespective of whether it is built in the debug mode or not.
86853. **comment:** Often a user misspells SetUp() as Setup() and spends a long time wondering why it is never called by Google Test. The declaration of the following method is solely for catching such an error at compile time: - The return type is deliberately chosen to be not void, so it will be a conflict if void Setup() is declared in the user's test fixture. - This method is private, so it will be another compiler error if the method is called from the user's test fixture. DO NOT OVERRIDE THIS FUNCTION. If you see an error about overriding the following function or about it being private, you have mis-spelled SetUp() as Setup().
label: code-design
86854. **comment:** NOLINT NOLINT GTEST_OS_WINDOWS

label: test

86855. Gets the number of tests that should run.
86856. Returns true if pathname describes an absolute path.
86857. Visual Studio 2010, 2012, and 2013 define symbols in std::tr1 that conflict with our own definitions. Therefore using our own tuple does not work on those compilers.

86858. **comment:** Suppresses MSVC warnings 4072 (unreachable code) for the code following statement if it returns or throws (or doesn't return or throw in some situations).

label: code-design

86859. Returns true if exit_status describes a process that was terminated by a signal, or exited normally with a nonzero exit code.

86860. Protects copying of all linked_ptr objects.

86861. Gets the number of all tests in this test case.

86862. GCC 4.4.7

86863. Turns const U&, U&, const U, and U all into U.

86864. Gets the number of successful test cases.

86865. GTEST_OS_WINDOWS

86866. Determines whether Google Test can use the pthreads library.

86867. Allows streaming basic output manipulators such as endl or flush into this object.

86868. **comment:** Google Test - The Google C++ Testing Framework This file implements a universal value printer that can print a value of any type T: void ::testing::internal::UniversalPrinter<T>::Print(value, ostream_ptr); A user can teach this function how to print a class type T by defining either operator<<() or PrintTo() in the namespace that defines T. More specifically, the FIRST defined function in the following list will be used (assuming T is defined in namespace foo): 1. foo::PrintTo(const T&, ostream*) 2. operator<<(ostream&, const T&) defined in either foo or the global namespace. If none of the above is defined, it will print the debug string of the value if it is a protocol buffer, or print the raw bytes in the value otherwise. To aid debugging: when T is a reference type, the address of the value is also printed; when T is a (const) char pointer, both the pointer value and the NUL-terminated string it points to are printed. We also provide some convenient wrappers: // Prints a value to a string. For a (const or not) char // pointer, the NUL-terminated string (but not the pointer) is // printed. std::string ::testing::PrintToString(const T& value); // Prints a value tersely: for a reference type, the referenced // value (but not the address) is printed; for a (const or not) char // pointer, the NUL-terminated string (but not the pointer) is // printed. void ::testing::internal::UniversalTersePrint(const T& value, ostream*); // Prints value using the type inferred by the compiler. The difference // from UniversalTersePrint() is that this function prints both the // pointer and the NUL-terminated string for a (const or not) char pointer. void ::testing::internal::UniversalPrint(const T& value, ostream*); // Prints the fields of a tuple tersely to a string vector, one // element for each field. Tuple support must be enabled in // gtest-port.h. std::vector<string> UniversalTersePrintTupleFieldsToStrings(const Tuple& value); Known limitation: The print primitives print the elements of an STL-style container using the compiler-inferred type of *iter where iter is a const_iterator of the container. When const_iterator is an input iterator but not a forward iterator, this inferred type may not match value_type, and the print output may be incorrect. In practice, this is rarely a problem as for most containers const_iterator is a forward iterator. We'll fix this if there's an actual need for it. Note that this fix cannot rely on value_type being defined as many user-defined container types don't have value_type.

label: code-design

86869. The c'tor pushes the given source file location and message onto a trace stack maintained by Google Test.

86870. Brings in definitions for functions used in the testing::internal::posix namespace (read, write, close, chdir, isatty, stat). We do not currently use them on Windows Mobile.

86871. Determines whether to support stream redirection. This is used to test output correctness and to implement death tests.

86872. Creates a Test object.

86873. class CartesianProductGenerator4::Iterator

86874. **comment:** The abstract class that all tests inherit from. In Google Test, a unit test program contains one or many TestCases, and each TestCase contains one or many Tests. When you define a test using the TEST macro, you don't need to explicitly derive from Test - the TEST macro automatically does this for you. The only time you derive from Test is when defining a test fixture to be used a TEST_F. For example: class FooTest : public testing::Test { protected: void SetUp() override { ... } void TearDown() override { ... } ... }; TEST_F(FooTest, Bar) { ... } TEST_F(FooTest, Baz) { ... } Test is not copyable.

label: code-design

86875. This flag sets up the filter to select by name using a glob pattern the tests to run. If the filter is not given all tests are executed.

86876. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_PORT_H_

86877. GTEST_HAS_STREAM_REDIRECTION

86878. Base class for ValueHolder<T>. Allows a caller to hold and delete a value without knowing its type.

86879. **comment:** Returns the TestCase object for the test that's currently running, or NULL if no test is running.

label: test

86880. defined(_MSC_VER) || defined(__BORLANDC__) GTEST_HAS_EXCEPTIONS

86881. 6.1.3.2 Tuple creation functions.

86882. Anything in namespace gtest_internal is Google Test's INTERNAL IMPLEMENTATION DETAIL and MUST NOT BE USED DIRECTLY in user code.

86883. This overload prints a (const) char array compactly.

86884. **comment:** FilePath - a class for file and directory pathname manipulation which handles platform-specific conventions (like the pathname separator). Used for helper functions for naming files in a directory for xml output. Except for Set methods, all methods are const or static, which provides an "immutable value object" -- useful for peace of mind. A FilePath with a value ending in a path separator ("like>this/") represents a directory, otherwise it is assumed to represent a file. In either case, it may or may not represent an actual file or directory in the file system. Names are NOT checked for syntax correctness -- no checking for illegal characters, malformed paths, etc.

label: code-design

86885. GTEST_INCLUDE_GTEST_GTEST_TYPED_TEST_H_

86886. For FullMatch(). For PartialMatch().

86887. STLport, provided with the Android NDK, has neither <tr1/tuple> or <tuple>.

86888. This is the only specialization that allows VC++ 7.1 to remove const in 'const int[3]' and 'const int[3][4]'. However, it causes trouble with GCC and thus needs to be conditionally compiled.

86889. IWYU pragma: export // NOLINT

86890. namespace internal

86891. Helper function for *_STRNE on wide strings. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

86892. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. TestMetaFactory creates test factories for passing into MakeAndRegisterTestInfo function. Since MakeAndRegisterTestInfo receives ownership of test factory pointer, same factory object cannot be passed into that method twice. But ParameterizedTestCaseInfo is going to call it for each Test/Parameter value combination. Thus it needs meta factory creator class.

86893. Passing non-POD classes through ellipsis (...) crashes the ARM compiler and generates a warning in Sun Studio. The Nokia Symbian and the IBM XL C/C++ compiler try to instantiate a copy constructor for objects passed through ellipsis (...), failing for uncopyable objects. We define this to ensure that only POD is passed through ellipsis on these systems.

86894. Fired after all test activities have ended.

86895. Returns the standard listener responsible for the default XML output controlled by the --gtest_output=xml flag. Can be removed from the listeners list by users who want to shut down the default XML output controlled by this flag and substitute it with custom one. Note that removing this object from the listener list with Release transfers its ownership to the caller and makes this function return NULL the next time.

86896. GCC 4.5.3

86897. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. ParameterizedTestCaseInfo accumulates tests obtained from TEST_P macro invocations for a particular test case and generators obtained from INSTANTIATE_TEST_CASE_P macro invocations for that test case. It registers tests with all values generated by all generators when asked.

86898. Array.

86899. Gets the user supplied key.

86900. The type of basic IO manipulators (endl, ends, and flush) for narrow streams.

86901. We need to use cached value referenced by iterator_ because *iterator_ can return a temporary object (and of type other than T), so just having "return &*iterator_;" doesn't work. value_ is updated here and not in Advance() because Advance() can advance iterator_ beyond the end of the range, and we cannot

detect that fact. The client code, on the other hand, is responsible for not calling Current() on an out-of-range iterator.

86902. The user didn't tell us whether exceptions are enabled, so we need to figure it out.

86903. Next, recurses (at compile time) with the tail of the type list.

86904. This prevents the user from using TypeWithSize<N> with incorrect values of N.

86905. GTEST_OS_LINUX && !defined(__ia64__)

86906. file

86907. Thread annotations

86908. Returns true iff the path is "".

86909. Returns the TestResult that holds test properties recorded during execution of SetUpTestCase and TearDownTestCase.

86910. Returns the current working directory, or "" if unsuccessful.

86911. To visit static members of the fixture, add theTestFixture:: prefix.

86912. Ctor. TestProperty does NOT have a default constructor. Always use this constructor (with parameters) to create a TestProperty object.

86913. Fired before the test starts.

86914. Create returns false if there was an error determining the appropriate action to take for the current death test; for example, if the gtest_death_test_style flag is set to an invalid value. The LastMessage method will return a more detailed message in that case. Otherwise, the DeathTest pointer pointed to by the "test" argument is set. If the death test should be skipped, the pointer is set to NULL; otherwise, it is set to the address of a new concrete DeathTest object that controls the execution of the current test.

86915. **comment:** An enumeration of the three reasons that a test might be aborted.
label: test

86916. Prints the given number of bytes in the given object to the given ostream.

86917. Converts a wide C string to a String using the UTF-8 encoding. NULL will be converted to "(null)". If an error occurred during the conversion, "(failed to convert from wide string)" is returned.

86918. Windows CE does not define _snprintf_s and MSVC prior to 2005 doesn't complain about _snprintf.

86919. Macros for disabling Microsoft Visual C++ warnings. GTEST_DISABLE_MSC_WARNINGS_PUSH_(4800 4385) /* code that triggers warnings C4800 and C4385 */ GTEST_DISABLE_MSC_WARNINGS_POP_()

86920. NOLINT

86921. Returns a pointer to the last occurrence of a valid path separator in the FilePath. On Windows, for example, both '/' and '\' are valid path separators. Returns NULL if no path separator was found.

86922. TuplePolicy<TupleT> must provide: - tuple_size size of tuple TupleT. - get<size_t I>(const TupleT& t) static function extracting element I of tuple TupleT. - tuple_element<size_t I>::type type of element I of tuple TupleT.

86923. The usual test fixture members go here too.

86924. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Stores a parameter value and later creates tests parameterized with that value.

86925. The d'tor is virtual as we intend to inherit from Test.

86926. Increments the death test count, returning the new count.

86927. A function level attribute to disable checking for use of uninitialized memory when built with MemorySanitizer.

86928. dummy_ must not have a const type. Otherwise an overly eager compiler (e.g. MSVC 7.1 & 8.0) may try to merge TypeIdHelper<T>::dummy_ for different Ts as an "optimization".

86929. The compiler is required to allocate a different TypeIdHelper<T>::dummy_ variable for each T used to instantiate the template. Therefore, the address of dummy_ is guaranteed to be unique.

86930. Accessors for the implementation object.

86931. Constructs from a native array. Copies the source.

86932. The base case for the compile time recursion.

86933. The name of the source file where the test part took place, or "" if the source file is unknown.

86934. Returns the number of threads running in the process, or 0 to indicate that we cannot detect it.

86935. __BORLANDC__

86936. A unique struct template used as the default value for the arguments of class template Templates. This allows us to simulate variadic templates (e.g. Templates<int>, Templates<int, double>, and etc), which C++ doesn't support directly.

86937. We include tr1::tuple even if std::tuple is available to define printers for them.

86938. Returns true iff this is NAN (not a number).

86939. This helper template allows PrintTo() for tuples and UniversalTersePrintTupleFieldsToStrings() to be defined by induction on the number of tuple fields. The idea is that TuplePrefixPrinter<N>::PrintPrefixTo(t, os) prints the first N fields in tuple t, and can be defined in terms of TuplePrefixPrinter<N - 1>. The inductive case.

86940. expected(NULL)

86941. **comment:** The possible outcomes of a test part (i.e. an assertion or an explicit SUCCEED(), FAIL(), or ADD_FAILURE()).
label: test

86942. Helper function for *_STREQ on wide strings. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

86943. Helper for suppressing false warning from Clang on a const char* variable declared in a conditional expression always being NULL in the else branch.

86944. Implements printing an array type T[N].

86945. GTEST_HAS_PARAM_TEST

86946. Helper template function for comparing floating-points. Template parameter: RawType: the raw floating-point type (either float or double) INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

86947. GCC >= 4.6.0

86948. **comment:** Due to C++ preprocessor weirdness, we need double indirection to concatenate two tokens when one of them is __LINE__. Writing foo ## __LINE__ will result in the token foo__LINE__, instead of foo followed by the current line number. For more details, see <http://www.parashift.com/c++-faq-lite/misc-technical-issues.html#faq-39.6>
label: code-design

86949. Streams a custom failure message into this object.

86950. scripts/fuse_gtest.py depends on gtest's own header being #included *unconditionally*. Therefore these #includes cannot be moved inside #if GTEST_HAS_PARAM_TEST.

86951. The maximum number a BiggestInt can represent. This definition works no matter BiggestInt is represented in one's complement or two's complement. We cannot rely on numeric_limits in STL, as __int64 and long long are not part of standard C++ and numeric_limits doesn't need to be defined for them.

86952. The symbol "fail" here expands to something into which a message can be streamed.

86953. The raw floating-point number. The bits that represent the number.

86954. Gets the elapsed time, in milliseconds.

86955. **comment:** We use a const char* instead of an std::string, as Google Test used to be used where std::string is not available. TODO(wan@google.com): change to std::string.
label: code-design

86956. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. TestMetaFactoryBase is a base class for meta-factories that create test factories for passing into MakeAndRegisterTestInfo function.

86957. In order to catch the mistake of putting tests that use different test fixture classes in the same test case, we need to assign unique IDs to fixture classes and compare them. The TypeId type is used to hold such IDs. The user should treat TypeId as an opaque type: the only operation allowed on TypeId values is to compare them for equality using the == operator.

86958. GTEST_TUPLE_NAMESPACE_

86959. __clang__

86960. Formats a byte as "%02X".

86961. Fired before environment set-up for each iteration of tests starts.

86962. If a C string is compared with an STL string object, we know it's meant to point to a NUL-terminated string, and thus can print it as a string.

86963. !__BORLANDC__

86964. # of fraction bits in a number.

86965. **comment:** MSVC warns about implicitly converting from double to int for possible loss of data, so we need to temporarily disable the warning.

label: code-design

86966. class ValuesInIteratorRangeGenerator::Iterator

86967. Binary predicate assertion macros.

86968. Leave whatever circle we're part of. Returns true if we were the last member of the circle. Once this is done, you can join() another.

86969. Define this macro to 1 to omit the definition of FAIL(), which is a generic name and clashes with some other libraries.

86970. Runs the test after the test fixture has been set up. A sub-class must implement this to define the test logic. DO NOT OVERRIDE THIS FUNCTION DIRECTLY IN A USER PROGRAM. Instead, use the TEST or TEST_F macro.

86971. namespace edit_distance

86972. C++11 puts its tuple into the ::std namespace rather than ::std::tr1. gtest expects tuple to live in ::std::tr1, so put it there. This causes undefined behavior, but supported compilers react in the way we intend.

86973. Succeeded. Failed but the test can continue. Failed and the test should be terminated.

86974. Destroys the managed object for the current thread, if any.

86975. **comment:** True iff any test in this test case should run.

label: test

86976. Platform-indicating macros ----- Macros indicating the platform on which Google Test is being used (a macro is defined to 1 if compiled on the given platform; otherwise UNDEFINED -- it's never defined to 0.). Google Test defines these macros automatically. Code outside Google Test MUST NOT define them. GTEST_OS_AIX - IBM AIX GTEST_OS_CYGWIN - Cygwin GTEST_OS_FREEBSD - FreeBSD GTEST_OS_HPUX - HP-UX GTEST_OS_LINUX - Linux GTEST_OS_LINUX_ANDROID - Google Android GTEST_OS_MAC - Mac OS X GTEST_OS_IOS - iOS GTEST_OS_NACL - Google Native Client (NaCl) GTEST_OS_OPENBSD - OpenBSD GTEST_OS_QNX - QNX GTEST_OS_SOLARIS - Sun Solaris GTEST_OS_SYMBIAN - Symbian GTEST_OS_WINDOWS - Windows (Desktop, MinGW, or Mobile) GTEST_OS_WINDOWS_DESKTOP - Windows Desktop GTEST_OS_WINDOWS_MINGW - MinGW GTEST_OS_WINDOWS_MOBILE - Windows Mobile GTEST_OS_WINDOWS_PHONE - Windows Phone GTEST_OS_WINDOWS_RT - Windows Store App/WinRT GTEST_OS_ZOS - z/OS Among the platforms, Cygwin, Linux, Max OS X, and Windows have the most stable support. Since core members of the Google Test project don't have access to other platforms, support for them may be less stable. If you notice any problems on your platform, please notify googletestframework@googlegroups.com (patches for fixing them are even more welcome!). It is possible that none of the GTEST_OS_* macros are defined.

86977. Asserts that a given statement causes the program to exit, with an integer exit status that satisfies predicate, and emitting error output that matches regex.

86978. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. *

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan) The Google C++ Testing Framework (Google Test) This header file defines the public API for Google Test. It should be included by any test program that uses Google Test. IMPORTANT NOTE: Due to limitation of the C++ language, we have to leave some internal implementation details in this header file. They are clearly marked by comments like this: // INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM. Such code is NOT meant to be used by a user directly, and is subject to CHANGE WITHOUT NOTICE. Therefore DO NOT DEPEND ON IT in a user program! Acknowledgment: Google Test borrowed the idea of automatic test registration from Barthelemy Dagenais' (barthelemy@prologue.com) easyUnit framework.

86979. GTEST_ASSERT_ is the basic statement to which all of the assertions in this file reduce. Don't use this in your code.

86980. On some platforms, <regex.h> needs someone to define size_t, and won't compile otherwise. We can #include it here as we already included <stdlib.h>, which is guaranteed to define size_t through <stddef.h>. NOLINT

86981. GTEST_HAS_COMBINE

86982. Runs every test in this TestCase.

86983. Note: we deliberately don't call this PrintTo(), as that name conflicts with ::testing::internal::PrintTo in the body of the function.

86984. The mask for the fraction bits.

86985. Gets the (mutable) vector of TestInfos in this TestCase.

86986. Inspired by boost/config/stdlib/dinkumware.hpp

86987. Streams a value (either a pointer or not) to this object.

86988. GTEST_HAS_TYPED_TEST || GTEST_HAS_TYPED_TEST_P

86989. A workarond for the bug in VC++ 7.1 that prevents us from instantiating UniversalPrinter with T directly.

86990. begin[i]_ and end[i]_ define the i-th range that Iterator traverses. current[i]_ is the actual traversing iterator.

86991. Having the same base generator guarantees that the other iterator is of the same type and we can downcast.

86992. Most value-parameterized classes can ignore the existence of WithParamInterface, and can just inherit from ::testing::TestWithParam.

86993. Creates an empty UnitTest.

86994. enabler

86995. ByRef<T>::type is T if T is a reference; otherwise it's const T&.

86996. The helper function for {ASSERT|EXPECT}_STRCASEEQ. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

86997. GetParam works just the same here as if you inherit from TestWithParam.

86998. Removes const from a type if it is a const type, otherwise leaves it unchanged. This is the same as tr1::remove_const, which is not widely available yet.

86999. OS detection

87000. We cannot use std::numeric_limits<T>::max() as it clashes with the max() macro defined by <windows.h>.

87001. This overload is used when k >= 1.

87002. GTEST_INCLUDE_GTEST_GTEST_PARAM_TEST_H_ Copyright 2006, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan) Google C++ Testing Framework definitions useful in production code.

87003. **comment:** C++ doesn't allow casting from a function pointer to any object pointer. IsTrue() silences warnings: "Condition is always true", "unreachable code".

label: code-design

87004. Returns true iff the test fatally failed.

87005. Helper macro for defining tests.

87006. GTEST_HAS_TR1_TUPLE

87007. The biggest signed integer type the compiler supports.

87008. Generates a nonfatal failure with a generic message.

87009. At this point we are sure that the object we found is of the same type we are looking for, so we downcast it to that type without further checks.
87010. Pops a trace from the per-thread Google Test trace stack.
87011. class CartesianProductGenerator2
87012. gtest-port.h guarantees to #include <pthread.h> when GTEST_HAS_PTHREAD is true. NOLINT
87013. 6.1.3.4 Element access.
87014. GetTypeName<T>() returns a human-readable name of type T. NB: This function is also used in Google Mock, so don't move it inside of the typed-test-only section below.
87015. Returns true iff the current test has the same fixture class as the first test in the current test case.
87016. With this overloaded version, we allow anonymous enums to be used in {ASSERT|EXPECT}_EQ when compiled with gcc 4, as anonymous enums can be implicitly cast to BiggestInt.
87017. Invoked when a ThreadLocal instance is destroyed.
87018. Deletes self. We deliberately pick an unusual name for this internal method to avoid clashing with names used in user TESTs.
87019. In theory, defining stuff in the ::std namespace is undefined behavior. We can do this as we are playing the role of a standard library vendor.
87020. Initializes owner_thread_id_ and critical_section_ in static mutexes.
87021. Returns true iff the unit test passed (i.e. all test cases passed).
87022. Bool() allows generating tests with parameters in a set of (false, true). Synopsis: Bool() - returns a generator producing sequences with elements {false, true}. It is useful when testing code that depends on Boolean flags. Combinations of multiple flags can be tested when several Bool()'s are combined using Combine() function. In the following example all tests in the test case FlagDependentTest will be instantiated twice with parameters false and true. class FlagDependentTest : public testing::TestWithParam<bool> { virtual void SetUp() { external_flag = GetParam(); } } INSTANTIATE_TEST_CASE_P(BoolSequence, FlagDependentTest, Bool());
87023. The user didn't tell us explicitly, so we make reasonable assumptions about which platforms have pthreads support. To disable threading support in Google Test, add -DGTEST_HAS_PTHREAD=0 to your compiler flags.
87024. Forward-declares a static mutex.
87025. Initializes Google Test. This must be called before calling RUN_ALL_TESTS(). In particular, it parses a command line for the flags that Google Test recognizes. Whenever a Google Test flag is seen, it is removed from argv, and *argc is decremented. No value is returned. Instead, the Google Test flag variables are updated. Calling the function for the second time has no user-visible effect.
87026. <regex.h> is not available on Windows. Use our own simple regex implementation instead.
87027. Converts the given wide string to a narrow string using the UTF-8 encoding, and streams the result to this Message object.
87028. The user didn't tell us whether RTTI is enabled, so we need to figure it out.
87029. GetTypeId<T>() returns the ID of type T. Different values will be returned for different types. Calling the function twice with the same type argument is guaranteed to return the same ID.
87030. Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan) The Google C++ Testing Framework (Google Test) This header file defines the public API for death tests. It is #included by gtest.h so a user doesn't need to include this directly.
87031. Returns the number of the test properties.
87032. Returns the fraction bits of this number.
87033. C++Builder cannot use member overload resolution during template instantiation. The simplest workaround is to use its C++0x type traits functions (C++Builder 2009 and above only).
87034. When printing a plain char, we always treat it as unsigned. This way, the output won't be affected by whether the compiler thinks char is signed or not.
87035. Skips to the first non-space char after the first comma in 'str'; returns NULL if no comma is found in 'str'.
87036. **comment:** Helpers for suppressing warnings on unreachable code or constant condition.
 label: code-design
87037. GTEST_OS_SYMBIAN
87038. **comment:** A macro for implementing the helper functions needed to implement ASSERT_?? and EXPECT_?. It is here just to avoid copy-and-paste of similar code. For each templated helper function, we also define an overloaded version for BiggestInt in order to reduce code bloat and allow anonymous enums to be used with {ASSERT|EXPECT}_?? when compiled with gcc 4. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.
 label: code-design
87039. We don't support MSVC 7.1 with exceptions disabled now. Therefore all the compilers we care about are adequate for supporting value-parameterized tests.
87040. Copy constructor. Used in EXPECT_TRUE/FALSE(assertion_result).
87041. namespace testing
87042. iteration
87043. Sets up the stuff shared by all tests in this test case. Google Test will call Foo::SetUpTestCase() before running the first test in test case Foo. Hence a sub-class can define its own SetUpTestCase() method to shadow the one defined in the super class.
87044. namespace internal2 namespace testing
87045. This flag specifies the maximum number of stack frames to be printed in a failure message.
87046. Pointer to the function that sets up the test case.
87047. Returns true if pathname describes a root directory. (Windows has one root directory per disk drive.)
87048. Overload for std::pair.
87049. Same as above, but the input is represented as strings.
87050. **comment:** unused
 label: code-design
87051. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_LINKED_PTR_H_ Copyright 2007, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: wan@google.com (Zhanyong Wan)
87052. Type lists of length 1, 2, 3, and so on.
87053. Next, declare that you will define a type-parameterized test case (the _P suffix is for "parameterized" or "pattern", whichever you prefer).
87054. AssertTypeEq<T1, T2>::type is defined iff T1 and T2 are the same type. This can be used as a compile-time assertion to ensure that two types are equal.
87055. **comment:** Adds a test part result to the list.
 label: test

87056. Creates directories so that path exists. Returns true if successful or if the directories already exist; returns false if unable to create directories for any reason. Will also return false if the FilePath does not represent a directory (that is, it doesn't end with a path separator).
87057. `_WIN32_WCE`
87058. Returns the working directory when the first TEST() or TEST_F() was executed. The UnitTest object owns the string.
87059. An array of TestPartResult objects. Don't inherit from TestPartResultArray as its destructor is not virtual.
87060. Constructor.
87061. Postfix version of operator++.
87062. Type utilities needed for implementing typed and type-parameterized tests. This file is generated by a SCRIPT. DO NOT EDIT BY HAND! Currently we support at most 50 types in a list, and at most 50 type-parameterized tests in one type-parameterized test case. Please contact googletestframework@googlegroups.com if you need more.
87063. Utilities for native arrays.
87064. Fired after the test case ends.
87065. ValuesIn() function allows generation of tests with parameters coming from a container. Synopsis: ValuesIn(const T (&array)[N]) - returns a generator producing sequences with elements from a C-style array. ValuesIn(const Container& container) - returns a generator producing sequences with elements from an STL-style container. ValuesIn(Iterator begin, Iterator end) - returns a generator producing sequences with elements from a range [begin, end) defined by a pair of STL-style iterators. These iterators can also be plain C pointers. Please note that ValuesIn copies the values from the containers passed in and keeps them to generate tests in RUN_ALL_TESTS(). Examples: This instantiates tests from test case StringTest each with C-string values of "foo", "bar", and "baz": const char* strings[] = {"foo", "bar", "baz"}; INSTANTIATE_TEST_CASE_P(StringSequence, SrtngTest, ValuesIn(strings)); This instantiates tests from test case StlStringTest each with STL strings with values "a" and "b": ::std::vector< ::std::string> GetParameterStrings() { ::std::vector< ::std::string> v; v.push_back("a"); v.push_back("b"); return v; } INSTANTIATE_TEST_CASE_P(CharSequence, StlStringTest, ValuesIn(GetParameterStrings())); This will also instantiate tests from CharTest each with parameter values 'a' and 'b': ::std::list<char> GetParameterChars() { ::std::list<char> list; list.push_back('a'); list.push_back('b'); return list; } ::std::list<char> l = GetParameterChars(); INSTANTIATE_TEST_CASE_P(CharSequence2, CharTest, ValuesIn(l.begin(), l.end()));
87066. To avoid conditional compilation everywhere, we make it gtest-port.h's responsibility to #include the header implementing tuple.
87067. Template lists of length 1, 2, 3, and so on.
87068. !defined(GTEST_HAS_STD_STRING)
87069. Returns a copy of the FilePath with the case-insensitive extension removed. Example: FilePath("dir/file.exe").RemoveExtension("EXE") returns FilePath("dir/file"). If a case-insensitive extension is not found, returns a copy of the original FilePath.
87070. The helper class for {ASSERT|EXPECT}_EQ. The template argument `lhs_is_null_literal` is true iff the first argument to ASSERT_EQ() is a null pointer literal. The following default implementation is for `lhs_is_null_literal` being false.
87071. **comment:** INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Outputs a message explaining invalid registration of different fixture class for the same test case. This may happen when TEST_P macro is used to define two tests with the same name but in different namespaces.
label: code-design
87072. Determines whether to support death tests. Google Test does not support death tests for VC 7.1 and earlier as abort() in a VC 7.1 application compiled as GUI in debug config pops up a dialog window that cannot be suppressed programmatically.
87073. This flag sets how many times the tests are repeated. The default value is 1. If the value is -1 the tests are repeating forever.
87074. GTEST_INCLUDE_GTEST_GTEST_MESSAGE_H_ Copyright 2005, Google Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Authors: wan@google.com (Zhanyong Wan), eefacm@gmail.com (Sean McAfee) The Google C++ Testing Framework (Google Test) This header file declares the String class and functions used internally by Google Test. They are subject to change without notice. They should not be used by code external to Google Test. This header file is #included by <gtest/internal/gtest-internal.h>. It should not be #included by other files.
87075. **comment:** We cannot name this class MutexLock because the ctor declaration would conflict with a macro named MutexLock, which is defined on some platforms. That macro is used as a defensive measure to prevent against inadvertent misuses of MutexLock like "MutexLock(&mu)" rather than "MutexLock l(&mu)". Hence the typedef trick below.
label: code-design
87076. Returns the prefix of 'str' before the first comma in it; returns the entire string if it contains no comma.
87077. Smart pointer members.
87078. Prints the fields of a tuple tersely to a string vector, one element for each field. See the comment before UniversalTersePrint() for how we define "tersely".
87079. Creates a new ValueHolder<T> object holding a default value passed to this ThreadLocal<T>'s constructor and returns it. It is the caller's responsibility not to call this when the ThreadLocal<T> instance already has a value on the current thread.
87080. GTEST_OS_LINUX_ANDROID && __STLPORT_MAJOR && !_EXCEPTIONS
87081. Returns true iff the test part fatally failed.
87082. GTEST_n_TUPLE_(T) is the type of an n-tuple.
87083. Protects mutable state in *impl_. This is mutable as some const methods need to lock it too.
87084. 5-ary predicate assertion macros.
87085. Fired after each iteration of tests finishes.
87086. Returns the bits that represents this number.
87087. GTEST_HAS_STD_TUPLE_
87088. Sleeps for (roughly) n milliseconds. This function is only for testing Google Test's own constructs. Don't use it in user tests, either directly or indirectly.
87089. Adds a reference to const on top of T as necessary. For example, it transforms `char ==> const char&` `const char ==> const char&` `char& ==> const char&` The argument T must depend on some template parameters.
87090. The Nokia Symbian and IBM XL C/C++ compilers cannot decide between `const T&` and `const T*` in a function template. These compilers _can_ decide between class template specializations for T and T*, so a tr1::type_traits-like is_pointer works.
87091. Clang defines __GXX_RTTI starting with version 3.0, but its manual recommends using `has_feature` instead. `has_feature(cxx_rtti)` is supported since 2.7, the first version with C++ support.
87092. Values() allows generating tests from explicitly specified list of parameters. Synopsis: Values(T v1, T v2, ..., T vN) - returns a generator producing sequences with elements v1, v2, ..., vN. For example, this instantiates tests from test case BarTest each with values "one", "two", and "three": INSTANTIATE_TEST_CASE_P(NumSequence, BarTest, Values("one", "two", "three")); This instantiates tests from test case BazTest each with values 1, 2, 3.5. The exact type of values will depend on the type of parameter in BazTest. INSTANTIATE_TEST_CASE_P(FloatingNumbers, BazTest, Values(1, 2, 3.5)); Currently, Values() supports from 1 to 50 parameters.
87093. Creates a TestCase with the given name. TestCase does NOT have a default constructor. Always use this constructor to create a TestCase object. Arguments: name: name of the test case a_type_param: the name of the test's type parameter, or NULL if this is not a type-parameterized test. set_up_tc: pointer to the function that sets up the test case tear_down_tc: pointer to the function that tears down the test case
87094. We already know that 'expected' is a null pointer.
87095. Internal macro for implementing {EXPECT|ASSERT}_PRED_FORMAT4. Don't use this in your code.
87096. Helper function for implementing {EXPECT|ASSERT}_PRED5. Don't use this in your code.
87097. IsAProtocolMessage<T>::value is a compile-time bool constant that's true iff T is type ProtocolMessage, proto2::Message, or a subclass of those.
87098. Makes a successful assertion result.
87099. is a pointer

87100. GCC 4.5.4

87101. Clones a 0-terminated C string, allocating memory using new. The caller is responsible for deleting the return value using delete[]. Returns the cloned string, or NULL if the input is NULL. This is different from strdup() in string.h, which allocates memory using malloc().

87102. Assignment operator.

87103. **comment:** The current parameter value. Is also available in the test fixture's constructor. This member function is non-static, even though it only references static data, to reduce the opportunity for incorrect uses like writing 'WithParamInterface<bool>::GetParam()' for a test that uses a fixture whose parameter type is int.
label: code-design

87104. The Message class works like an ostream repeater. Typical usage: 1. You stream a bunch of values to a Message object. It will remember the text in a stringstream. 2. Then you stream the Message object to an ostream. This causes the text in the Message to be streamed to the ostream. For example; testing::Message foo; foo << 1 << " != " << 2; std::cout << foo; will print "1 != 2". Message is not intended to be inherited from. In particular, its destructor is not virtual. Note that stringstream behaves differently in gcc and in MSVC. You can stream a NULL char pointer to it in the former, but not in the latter (it causes an access violation if you do). The Message class hides this difference by treating a NULL char pointer as "(null)".

87105. **comment:** Gets the number of disabled tests in this test case.

label: test

87106. On Android, clone() is only available on ARM starting with Gingerbread.

87107. For timespec and nanosleep, used below. NOLINT

87108. MSVC can be configured to define wchar_t as a typedef of unsigned short. It defines _NATIVE_WCHAR_T_DEFINED when wchar_t is a native type. When wchar_t is a typedef, defining an overload for const wchar_t* would cause unsigned short* be printed as a wide string, possibly causing invalid memory accesses.

87109. for param_it for gen_it for test_it RegisterTests

87110. Compares two C strings, ignoring case. Returns true iff they have the same content. Unlike strcasecmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL C string, including the empty string.

87111. Returns the TestResult containing information on test failures and properties logged outside of individual test cases.

87112. Generates a success with a generic message.

87113. Next, recurses (at compile time) with the tail of the test list.

87114. The IEEE standard says that any comparison operation involving a NAN must return false.

87115. Then, use the TEST_P macro to define as many parameterized tests for this fixture as you want. The _P suffix is for "parameterized" or "pattern", whichever you prefer to think.

87116. **comment:** Gets the message associated with the test part.

label: test

87117. Returns true iff the current test has a fatal failure.

87118. Get the number of tests in this test case that should run.

87119. __has_feature(address_sanitizer)

87120. Starting with version 9.0 IBM Visual Age defines __RTTI_ALL__ to 1 if both the typeid and dynamic_cast features are present.

87121. ArrayEq() compares two k-dimensional native arrays using the elements' operator==, where k can be any integer ≥ 0 . When k is 0, ArrayEq() degenerates into comparing a single pair of values.

87122. Restores the test order to before the first shuffle.

87123. 4-ary predicate assertion macros.

87124. T is a function type, so '*os << p' doesn't do what we want (it just prints p as bool). We want to print p as a const void*. However, we cannot cast it to const void* directly, even using reinterpret_cast, as earlier versions of gcc (e.g. 3.4.5) cannot compile the cast when p is a function pointer. Casting to UInt64 first solves the problem.

87125. __WIN32_WCE

87126. Utilities for command line flags and environment variables.

87127. **comment:** The compiler used in Symbian has a bug that prevents us from declaring the tuple template as a friend (it complains that tuple is redefined). This hack bypasses the bug by declaring the members that should otherwise be private as public. Sun Studio versions < 12 also have the above bug.

label: code-design

87128. Windows CE has no C library. The abort() function is used in several places in Google Test. This implementation provides a reasonable imitation of standard behaviour.

87129. !GTEST_LANG_CXX11

87130. Returns true iff the test part failed.

87131. Controls whether events will be forwarded by the repeater to the listeners in the list.

87132. Constructs from a native array. References the source.

87133. Exception handling is in effect by default in HP aCC compiler. It has to be turned off by +noeh compiler option if desired.

87134. **comment:** This helper reduces code bloat. If we instead put its logic inside the previous CopyArray() function, arrays with different sizes would lead to different copies of the template code.

label: code-design

87135. Distinct from C++11 language support, some environments don't provide proper C++11 library support. Notably, it's possible to build in C++11 mode when targeting Mac OS X 10.6, which has an old libstdc++ with no C++11 support. libstdc++ has sufficient C++11 support as of GCC 4.6.0, __GLIBCXX__ 20110325, but maintenance releases in the 4.4 and 4.5 series followed this date, so check for those versions by their date stamps.
<https://gcc.gnu.org/onlinedocs/libstdc++/manual/abi.html#abi.versioning>

87136. Determine whether the compiler supports Microsoft's Structured Exception Handling. This is supported by several Windows compilers but generally does not exist on any other system.

87137. To distinguish different instances of the pattern, (yes, you can instantiate it more than once) the first argument to the INSTANTIATE_TEST_CASE_P macro is a prefix that will be added to the actual test case name. Remember to pick unique prefixes for different instantiations. The tests from the instantiation above will have these names: * InstantiationName/FooTest.DoesBlah/0 for "meeny" * InstantiationName/FooTest.DoesBlah/1 for "miny" * InstantiationName/FooTest.DoesBlah/2 for "moe" * InstantiationName/FooTest.HasBlahBlah/0 for "meeny" * InstantiationName/FooTest.HasBlahBlah/1 for "miny" * InstantiationName/FooTest.HasBlahBlah/2 for "moe" You can use these names in --gtest_filter. This statement will instantiate all tests from FooTest again, each with parameter values "cat" and "dog".

87138. Returns true iff the handle is a valid handle object that can be closed.

87139. First, register the first test in 'Test' for each type in 'Types'.

87140. For selecting which printer to use when a given type has neither << nor PrintTo().

87141. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_FILEPATH_H_ This file was GENERATED by command: pump.py gtest-type-util.h.pump DO NOT EDIT BY HAND!!!

87142. Returns the sign bit of this number.

87143. Tears down the stuff shared by all tests in this test case. Google Test will call Foo::TearDownTestCase() after running the last test in test case Foo. Hence a sub-class can define its own TearDownTestCase() method to shadow the one defined in the super class.

87144. Create a new circle that includes only this instance.

87145. EnableIf<condition>::type is void when 'Cond' is true, and undefined when 'Cond' is false. To use SFINAE to make a function overload only apply when a particular expression is true, add "typename EnableIf<expression>::type* = 0" as the last parameter.

87146. Macros for testing exceptions. * {ASSERT|EXPECT}_THROW(statement, expected_exception): Tests that the statement throws the expected exception. * {ASSERT|EXPECT}_NO_THROW(statement): Tests that the statement doesn't throw any exception. * {ASSERT|EXPECT}_ANY_THROW(statement): Tests that the statement throws an exception.

87147. Used to print a non-container, non-pointer value when the user doesn't define PrintTo() for it.

87148. Defines synchronization primitives.

87149. Returns the string representation of the regex.

87150. If you see an error about overriding the following function or about it being private, you have mis-spelled SetUp() as Setup().

87151. Fired before environment tear-down for each iteration of tests starts.

87152. Implements the helper function for {ASSERT|EXPECT}_GT

87153. A helper for implementing tuple_element<k, T>. kIndexValid is true iff k < the number of fields in tuple type T.

87154. A key pthreads uses for looking up per-thread values.
87155. A protobuf type a type implicitly convertible to BiggestInt (e.g. a named or unnamed enum type) anything else
87156. unit_test
87157. Constructs a Message from a C-string.
87158. class CartesianProductGenerator8::Iterator
87159. Returns the TestPartResult at the given index (0-based).
87160. Creates a new TestInfo object and registers it with Google Test; returns the created object. Arguments: test_case_name: name of the test case name; name of the test type_param the name of the test's type parameter, or NULL if this is not a typed or a type-parameterized test. value_param text representation of the test's value parameter, or NULL if this is not a type-parameterized test. fixture_class_id: ID of the test fixture class set_up_tc: pointer to the function that sets up the test case tear_down_tc: pointer to the function that tears down the test case factory: pointer to the factory that creates a test object. The newly created TestInfo instance will assume ownership of the factory object.
87161. Poor man's downcast.
87162. Flushes the buffers and, if severity is GTEST_FATAL, aborts the program.
87163. Join an existing circle.
87164. Formats log entry severity, provides a stream object for streaming the log message, and terminates the message with a newline when going out of scope.
87165. A pointer to the base generator instance. Used only for the purposes of iterator comparison to make sure that two iterators belong to the same generator.
87166. Base part of test case name for display purposes.
87167. Used to print a pointer that is neither a char pointer nor a member pointer, when the user doesn't define PrintTo() for it. (A member variable pointer or member function pointer doesn't really point to a location in the address space. Their representation is implementation-defined. Therefore they will be printed as raw bytes.)
87168. Cygwin 1.7 and below doesn't support ::std::wstring. Solaris' libc++ doesn't support it either. Android has no support for it at least as recent as Froyo (2.2).
87169. A handy wrapper around AddReference that works when the argument T depends on template parameters.
87170. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_TUPLE_H_
87171. Implements printing a reference type T&.
87172. C-string Comparisons. All tests treat NULL and any non-NULL string as different. Two NULLs are equal. * {ASSERT|EXPECT}_STREQ(s1, s2): Tests that s1 == s2 * {ASSERT|EXPECT}_STRNE(s1, s2): Tests that s1 != s2 * {ASSERT|EXPECT}_STRCASEEQ(s1, s2): Tests that s1 == s2, ignoring case * {ASSERT|EXPECT}_STRCASENE(s1, s2): Tests that s1 != s2, ignoring case For wide or narrow string objects, you can use the {ASSERT|EXPECT}_??() macros. Don't depend on the order in which the arguments are evaluated, which is undefined. These macros evaluate their arguments exactly once.
87173. Since the basic IO manipulators are overloaded for both narrow and wide streams, we have to provide this specialized definition of operator <<, even though its body is the same as the templated version above. Without this definition, streaming endl or other basic IO manipulators to Message will confuse the compiler.
87174. This header implements a family of generic predicate assertion macros: ASSERT_PRED_FORMAT1(pred_format, v1) ASSERT_PRED_FORMAT2(pred_format, v1, v2) ... where pred_format is a function or functor that takes n (in the case of ASSERT_PRED_FORMATn) values and their source expression text, and returns a testing::AssertionResult. See the definition of ASSERT_EQ in gtest.h for an example. If you don't care about formatting, you can use the more restrictive version: ASSERT_PRED1(pred, v1) ASSERT_PRED2(pred, v1, v2) ... where pred is an n-ary function or functor that returns bool, and the values v1, v2, ..., must support the << operator for streaming to std::ostream. We also define the EXPECT_* variations. For now we only support predicates whose arity is at most 5. Please email googletestframework@googlegroups.com if you need support for higher arities.
87175. xLC defines __EXCEPTIONS to 1 iff exceptions are enabled.
87176. class CartesianProductGenerator10::Iterator
87177. **comment:** MSVC defines this macro iff RTTI is enabled.
- label:** code-design
87178. Gets the number of successful tests.
87179. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_TYPE_UTIL_H_
87180. Finds the first element in the iterator range [begin, end) that equals elem. Element may be a native array type itself.
87181. The vector of TestInfos in their original order. It owns the elements in the vector.
87182. Overloads for ::string and ::std::string.
87183. Inspired by http://clang.llvm.org/docs/LanguageExtensions.html#_has_include
87184. unsigned int has size 4 in both gcc and MSVC. As base/basictypes.h doesn't compile on Windows, we cannot use uint32, uint64, and etc here.
87185. Gets the number of failed test cases.
87186. The helper function for {ASSERT|EXPECT}_STRCASENE. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.
87187. A copyable object representing a user specified test property which can be output as a key/value string pair. Don't inherit from TestProperty as its destructor is not virtual.
87188. pthread_create() accepts a pointer to a function type with the C linkage. According to the Standard (7.5/1), function types with different linkages are different even if they are otherwise identical. Some compilers (for example, SunStudio) treat them as different types. Since class methods cannot be defined with C-linkage we need to define a free C-function to pass into pthread_create().
87189. test_info
87190. NOLINT !GTEST_HAS_RTTI && GTEST_GCC_VER_ < 40302
87191. **comment:** Feature-indicating macros ----- Macros indicating which Google Test features are available (a macro is defined to 1 if the corresponding feature is supported; otherwise UNDEFINED -- it's never defined to 0.). Google Test defines these macros automatically. Code outside Google Test MUST NOT define them. These macros are public so that portable tests can be written. Such tests typically surround code using a feature with an #if which controls that code. For example: #if GTEST_HAS_DEATH_TEST EXPECT_DEATH(DoSomethingDeadly()); #endif GTEST_HAS_COMBINE - the Combine() function (for value-parameterized tests) GTEST_HAS_DEATH_TEST - death tests GTEST_HAS_PARAM_TEST - value-parameterized tests GTEST_HAS_TYPED_TEST - typed tests GTEST_HAS_TYPED_TEST_P - type-parameterized tests GTEST_IS_THREADSAFE - Google Test is thread-safe. GTEST_USES_POSIX_REGEX - enhanced POSIX regex is used. Do not confuse with GTEST_USES_POSIX_REGEX (see above) which users can define themselves. GTEST_USES_SIMPLE_REGEX - our own simple regex is used; the above two are mutually exclusive. GTEST_CAN_COMPARE_NULL - accepts untyped NULL in EXPECT_EQ().
- label:** code-design
87192. Provides a way for a thread to send notifications to a ThreadLocal regardless of its parameter type.
87193. Signals that the death test did not die as expected.
87194. We cannot call PrintTo(*it, os) here as PrintTo() doesn't handle *it being a native array.
87195. Prefix version of operator++.
87196. Test case base name for display purposes.
87197. Runs TearDownTestCase() for this TestCase. This wrapper is needed for catching exceptions thrown from TearDownTestCase().
87198. Generates a nonfatal failure at the given source file location with a generic message.
87199. Determines whether the given iterator and other point to the same element in the sequence generated by the generator. Used for implementing ParamGenerator<T>::operator==().
87200. **comment:** INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Use ImplicitCast_ as a safe version of static_cast for upcasting in the type hierarchy (e.g. casting a Foo* to a SuperclassOfFoo* or a const Foo*). When you use ImplicitCast_, the compiler checks that the cast is safe. Such explicit ImplicitCast_s are necessary in surprisingly many situations where C++ demands an exact type match instead of an argument type convertible to a target type. The syntax for using ImplicitCast_ is the same as for static_cast: ImplicitCast_<ToType>(expr) ImplicitCast_ would have been part of the C++ standard library, but the proposal was submitted too late. It will probably make its way into the language in the future. This relatively ugly name is intentional. It prevents clashes with similar functions users may have (e.g., implicit_cast). The internal namespace alone is not enough because the function can be found by ADL.
- label:** code-design
87201. Helper function for printing a tuple. Tuple must be either std::tr1::tuple or std::tuple type.
87202. GTEST_INCLUDE_GTEST_GTEST_PRED_IMPL_H_
87203. The value supplied by the user.
87204. MSVC warns about adding const to a function type, so we want to disable the warning.
87205. Creates an empty TestResult.
87206. This file was GENERATED by command: pump.py gtest-tuple.h.pump DO NOT EDIT BY HAND!!!
87207. This is an ordinary non-parameterized test.

87208. Until version 4.3.2, gcc has a bug that causes <tr1/functional>, which is #included by <tr1/tuple>, to not compile when RTTI is disabled. _TR1_FUNCTIONAL is the header guard for <tr1/functional>. Hence the following #define is a hack to prevent <tr1/functional> from being included.

87209. GTEST_HAS_CXXABI_H_ || __HP_aCC

87210. GTEST_HAS_EXCEPTIONS

87211. The specialization for size 4.

87212. **comment:** Prints the given number of elements in an array, without printing the curly braces.
label: code-design

87213. class CartesianProductGenerator3::Iterator

87214. Gets the i-th test case among all the test cases. i can range from 0 to total_test_case_count() - 1. If i is not in that range, returns NULL.

87215. For ::std::pair.

87216. We lose support for NULL detection where the compiler doesn't like passing non-POD classes through ellipsis (...).

87217. User-supplied thread function. User-supplied parameter to the thread function. When non-NULL, used to block execution until the controller thread notifies.

87218. Streams a Message to an ostream.

87219. Helper function for implementing {EXPECT|ASSERT}_PRED2. Don't use this in your code.

87220. Returns the death test count.

87221. UniversalPrintArray(begin, len, os) prints an array of 'len' elements, starting at address 'begin'.

87222. Functions with a different name on Windows.

87223. 0 seconds. And n ms.

87224. MakeFrom() is an expression whose type is From. We cannot simply use From(), as the type From may not have a public default constructor.

87225. TestClass must be a subclass of WithParamInterface<T> and Test.

87226. Internal macro for implementing {EXPECT|ASSERT}_PRED1. Don't use this in your code.

87227. The mask for the sign bit.

87228. Assume that Win32 HANDLE type is equivalent to void*. Doing so allows us to avoid including <windows.h> in this header file. Including <windows.h> is undesirable because it defines a lot of symbols and macros that tend to conflict with client code. This assumption is verified by WindowsTypesTest.HANDLEIsVoidStar.

87229. Always returns false.

87230. Ternary predicate assertion macros.

87231. Mutex implements mutex on Windows platforms. It is used in conjunction with class MutexLock: Mutex mutex; ... MutexLock lock(&mutex); // Acquires the mutex and releases it at the // end of the current scope. A static Mutex *must* be defined or declared using one of the following macros:
GTEST_DEFINE_STATIC_MUTEX_(g_some_mutex); GTEST_DECLARE_STATIC_MUTEX_(g_some_mutex); (A non-static Mutex is defined/declared in the usual way).

87232. Calculate the diff between 'left' and 'right' and return it in unified diff format. If not null, stores in 'total_line_count' the total number of lines found in left + right.

87233. Gets the number of successful tests in this test case.

87234. Note that we call GetTestId() instead of GetTypeId< ::testing::Test>() here to get the type ID of testing::Test. This is to work around a suspected linker bug when using Google Test as a framework on Mac OS X. The bug causes GetTypeId< ::testing::Test>() to return different values depending on whether the call is from the Google Test framework itself or from user test code. GetTestId() is guaranteed to always return the same value, as it always calls GetTypeId<>()

from the Google Test framework.

87235. Returns true iff the unit test failed (i.e. some test case failed or something outside of all tests failed).

87236. Helper function for implementing ASSERT_NEAR. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

87237. Misc public macros ----- GTEST_FLAG(flag_name) - references the variable corresponding to the given Google Test flag.

87238. **comment:** Defines a test that uses a test fixture. The first parameter is the name of the test fixture class, which also doubles as the test case name. The second parameter is the name of the test within the test case. A test fixture class must be declared earlier. The user should put his test code between braces after using this macro. Example: class FooTest : public testing::Test { protected: virtual void SetUp() { b_.AddElement(3); } Foo a_; Foo b_; }; TEST_F(FooTest, InitializesCorrectly) { EXPECT_TRUE(a_.StatusIsOK()); } TEST_F(FooTest, ReturnsElementCountCorrectly) { EXPECT_EQ(0, a_.size()); } EXPECT_EQ(1, b_.size()); }
label: code-design

87239. RTTI: debug mode only!

87240. Adds a TestInfo to this test case. Will delete the TestInfo upon destruction of the TestCase object.

87241. Gets the text streamed to this object so far as an std::string. Each '\0' character in the buffer is replaced with "\\0". INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

87242. The following family of struct and struct templates are used to represent type lists. In particular, TypesN<T1, T2, ..., TN> represents a type list with N types (T1, T2, ..., and TN) in it. Except for Types0, every struct in the family has two member types: Head for the first type in the list, and Tail for the rest of the list.

87243. 0

87244. Gets the name of the TestCase.

87245. Fired before each iteration of tests starts. There may be more than one iteration if GTEST_FLAG(repeat) is set. iteration is the iteration index, starting from 0.

87246. Returns true iff the test has a non-fatal failure.

87247. Sets the default_xml_generator attribute to the provided listener. The listener is also added to the listener list and previous default_xml_generator is removed from it and deleted. The listener can also be NULL in which case it will not be added to the list. Does nothing if the previous and the current listener objects are the same.

87248. Adds a TestProperty to the current TestResult object when invoked from inside a test, to current TestCase's ad_hoc_test_result_ when invoked from SetUpTestCase or TearDownTestCase, or to the global property set when invoked elsewhere. If the result already contains a property with the same key, the value will be updated.

87249. Defines some utility macros.

87250. dummy

87251. This helper class is used by {ASSERT|EXPECT}_NO_FATAL_FAILURE to check if a statement generates new fatal failures. To do so it registers itself as the current test part result reporter. Besides checking if fatal failures were reported, it only delegates the reporting to the former result reporter. The original result reporter is restored in the destructor. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

87252. A convenient wrapper for adding an environment for the test program. You should call this before RUN_ALL_TESTS() is called, probably in main(). If you use gtest_main, you need to call this before main() starts for it to take effect. For example, you can define a global variable like this: testing::Environment* const foo_env = testing::AddGlobalTestEnvironment(new FooEnvironment); However, we strongly recommend you to write your own main() and call AddGlobalTestEnvironment() there, as relying on initialization of global variables makes the code harder to read and may cause problems when you register multiple environments from different translation units and the environments have dependencies among them (remember that the compiler doesn't guarantee the order in which global variables from different translation units are initialized).

87253. Returns the assertion's negation. Used with EXPECT/ASSERT_FALSE.

87254. This header implements typed tests and type-parameterized tests.

87255. Copy an existing linked_ptr<>, adding ourselves to the list of references.

87256. Create the directory so that path exists. Returns true if successful or if the directory already exists; returns false if unable to create the directory for any reason, including if the parent directory does not exist. Not named "CreateDirectory" because that's a macro on Windows.

87257. Adds the given test name to defined_test_names_ and return true if the test case hasn't been registered; otherwise aborts the program.

87258. This overloaded version can be used in Windows programs compiled in UNICODE mode.

87259. AddRef<T>::type is T if T is a reference; otherwise it's T&. This is the same as tr1::add_reference<T>::type.

87260. Converts an integer from the sign-and-magnitude representation to the biased representation. More precisely, let N be 2 to the power of (kBitCount - 1), an integer x is represented by the unsigned number x + N. For instance, -N + 1 (the most negative number representable using sign-and-magnitude) is represented by 1; 0 is represented by N; and N - 1 (the biggest number representable using sign-and-magnitude) is represented by 2N - 1. Read http://en.wikipedia.org/wiki/Signed_number_representations for more details on signed number representations.

87261. Streams a non-pointer value to this object.

87262. TEST_P macro uses AddTestPattern() to record information about a single test in a LocalTestInfo structure. test_case_name is the base name of the test case (without invocation prefix). test_base_name is the name of an individual test without parameter index. For the test SequenceA/FooTest.DoBar/1 FooTest is test case base name and DoBar is test base name.

87263. Determines the version of gcc that is used to compile this.
87264. These two overloads allow streaming a wide C string to a Message using the UTF-8 encoding.
87265. Defines the unsigned integer type that has the same size as the floating point number.
87266. Asserts that a given statement causes the program to exit, either by explicitly exiting with a nonzero exit code or being killed by a signal, and emitting error output that matches regex.
87267. GTEST_INCLUDE_GTEST_INTERNAL_GTEST_PARAM_UTIL_H_ This file was GENERATED by command: pump.py gtest-param-util-generated.h.pump
DO NOT EDIT BY HAND!!!
87268. No implementation - assignment is unsupported.
87269. Clears the test part results.
87270. Sets parameter value. The caller is responsible for making sure the value remains alive and unchanged throughout the current test.
87271. GTEST_HAS_TR1_TUPLE || GTEST_HAS_STD_TUPLE_
87272. Creates a UTF-16 wide string from the given ANSI string, allocating memory using new. The caller is responsible for deleting the return value using delete[]. Returns the wide string, or NULL if the input is NULL. The wide string is created using the ANSI codepage (CP_ACP) to match the behaviour of the ANSI versions of Win32 calls and the C runtime.
87273. Given directory = "dir", base_name = "test", number = 0, extension = "xml", returns "dir/test.xml". If number is greater than zero (e.g., 12), returns "dir/test_12.xml". On Windows platform, uses \ as the separator rather than /.
87274. Boolean assertions. Condition can be either a Boolean expression or an AssertionResult. For more information on how to use AssertionResult with these macros see comments on that class.
87275. Maps a thread to a set of ThreadLocals that have values instantiated on that thread and notifies them when the thread exits. A ThreadLocal instance is expected to persist until all threads it has values on have terminated.
87276. The mask for the exponent bits.
87277. class CartesianProductGenerator9
87278. The upper limit for valid stack trace depths.
87279. true iff we know that the thread function has finished. The native thread object.
87280. Defines the stderr capturer: CaptureStdout - starts capturing stdout. GetCapturedStdout - stops capturing stdout and returns the captured string. CaptureStderr - starts capturing stderr. GetCapturedStderr - stops capturing stderr and returns the captured string.
87281. ImplicitlyConvertible<From, To>::value is a compile-time bool constant that's true iff type From can be implicitly converted to type To.
87282. Inside a test, access the test parameter with the GetParam() method of the TestWithParam<T> class:
87283. **comment:** Prints the given value using the << operator if it has one; otherwise prints the bytes in it. This is what UniversalPrinter<T>::Print() does when PrintTo() is not specialized or overloaded for type T. A user can override this behavior for a class type Foo by defining an overload of PrintTo() in the namespace where Foo is defined. We give the user this option as sometimes defining a << operator for Foo is not desirable (e.g. the coding style may prevent doing it, or there is already a << operator but it doesn't do what the user wants).
label: code-design
87284. Destroys a TestInfo object. This function is not virtual, so don't inherit from TestInfo.
87285. UnitTest class invokes this method to register tests in this test case test cases right before running tests in RUN_ALL_TESTS macro. This method should not be called more than once on any single instance of a ParameterizedTestCaseInfoBase derived class. UnitTest has a guard to prevent from calling this method more than once.
87286. Assuming T is defined in namespace foo, in the next statement, the compiler will consider all of: 1. foo::operator<< (thanks to Koenig look-up), 2. ::operator<< (as the current namespace is enclosed in ::), 3. testing::internal2::operator<< (thanks to the using statement above). The operator<< whose type matches T best will be picked. We deliberately allow #2 to be a candidate, as sometimes it's impossible to define #1 (e.g. when foo is ::std, defining anything in it is undefined behavior unless you are a compiler vendor.).
87287. Constructs a failure message for Boolean assertions such as EXPECT_TRUE.
87288. Returns the floating-point number that represent positive infinity.
87289. forcing value to bool
87290. DefaultPrintTo() is overloaded. The type of its first two arguments determine which version will be picked. If T is an STL-style container, the version for container will be called; if T is a pointer, the pointer version will be called; otherwise the generic version will be called. Note that we check for container types here, prior to we check for protocol message types in our operator<<. The rationale is: For protocol messages, we want to give people a chance to override Google Mock's format by defining a PrintTo() or operator<<. For STL containers, other formats can be incompatible with Google Mock's format for the container elements; therefore we check for container types here to ensure that our format is used. The second argument of DefaultPrintTo() is needed to bypass a bug in Symbian's C++ compiler that prevents it from picking the right overload between: PrintTo(const T& x, ...); PrintTo(T* x, ...);
87291. Does nothing if the current thread holds the mutex. Otherwise, crashes with high probability.
87292. Gets the number of all test parts. This is the sum of the number of successful test parts and the number of failed test parts.
87293. A helper for implementing get<k>().
87294. **comment:** On MinGW, we can have both GTEST_OS_WINDOWS and GTEST_HAS_PTHREAD defined, but we don't want to use MinGW's pthreads implementation, which has conformance problems with some versions of the POSIX standard.
label: code-design
87295. For FullMatch();
87296. A unique type used as the default value for the arguments of class template Types. This allows us to simulate variadic templates (e.g. Types<int>, Type<int, double>, and etc), which C++ doesn't support directly.
87297. Opaque implementation object. This field is never changed once the object is constructed. We don't mark it as const here, as doing so will cause a warning in the constructor of UnitTest. Mutable state in *impl_ is protected by mutex_.
87298. This field is mutable and needs to be reset before running the test for the second time.
87299. defined(GTEST_TUPLE_NAMESPACE_)
87300. __GNUC__
87301. Macros for declaring flags.
87302. Static value used for accessing parameter during a test lifetime.
87303. Typedefs the instances of the FloatingPoint template class that we care to use.
87304. Returns the name of the parameter type, or NULL if this is not a type-parameterized test case.
87305. Implements the helper function for {ASSERT|EXPECT}_LE
87306. Gets the vector of TestProperties.
87307. A string containing a description of the outcome of the last death test.
87308. How many times InitGoogleTest() has been called.
87309. The following list of PrintTo() overloads tells UniversalPrinter<T>::Print() how to print standard types (built-in types, strings, plain arrays, and pointers).
87310. Adds a TestPartResult to the current TestResult object. All Google Test assertion macros (e.g. ASSERT_TRUE, EXPECT_EQ, etc) eventually call this to report their results. The user code should use the assertion macros instead of calling this directly.
87311. An all-mode assert to verify that the given POSIX-style function call returns 0 (indicating success). Known limitation: this doesn't expand to a balanced 'if' statement, so enclose the macro in {} if you need to use it as the only statement in an 'if' branch.
87312. Returns the ParameterizedTestCaseRegistry object used to keep track of value-parameterized tests and instantiate and register them. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.
87313. We declare (but don't implement) this to prevent the compiler from implementing the assignment operator.
87314. Macros that test for HRESULT failure and success, these are only useful on Windows, and rely on Windows SDK macros and APIs to compile. * {ASSERT|EXPECT}_HRESULT_{SUCCEEDED|FAILED}(expr) When expr unexpectedly fails or succeeds, Google Test prints the expected result and the actual result with both a human-readable string representation of the error, if available, as well as the hex result code.
87315. Assignment releases the old value and acquires the new.
87316. Clears the results of all tests in the given test case.
87317. Per <http://blogs.msdn.com/b/oldnewthing/archive/2004/02/23/78395.aspx>, we assume that 0 is an invalid value for thread IDs.
87318. In describing the results of death tests, these terms are used with the corresponding definitions: exit status: The integer exit information in the format specified by wait(2) exit code: The integer code passed to exit(3), _exit(2), or returned from main()

87319. __has_feature(memory_sanitizer)
87320. To refer to typedefs in the fixture, add the "typename TestFixture::" prefix.
87321. GTEST_HAS_CXXABI_H
87322. signed/unsigned char is often used for representing binary data, so we print pointers to it as void* to be safe.
87323. MSVC 2005 and above support variadic macros.
87324. Defines and statically (i.e. at link time) initializes a static mutex. The initialization list here does not explicitly initialize each field, instead relying on default initialization for the unspecified fields. In particular, the owner_field (a pthread_t) is not explicitly initialized. This allows initialization to work whether pthread_t is a scalar or struct. The flag -Wmissing-field-initializers must not be specified for this to work.
87325. Overloaded PrintTo() for tuples of various arities. We support tuples of up-to 10 fields. The following implementation works regardless of whether tr1::tuple is implemented using the non-standard variadic template feature or not.
87326. class CartesianProductGenerator7
87327. This flag brings the debugger on an assertion failure.
87328. Returns the list of event listeners that can be used to track events inside Google Test.
87329. __LIBCPP_VERSION is defined by the libc++ library from the LLVM project.
87330. Looks up or creates and returns a structure containing information about tests and instantiations of a particular test case.
87331. GTEST_HAS_PARAM_TEST
87332. class CartesianProductGenerator6
87333. 1600 is Visual Studio 2010
87334. SameSizeTuplePrefixComparator<k, k>::Eq(t1, t2) returns true if the first k fields of t1 equals the first k fields of t2. SameSizeTuplePrefixComparator(k1, k2) would be a compiler error if k1 != k2.
87335. Test case id to verify identity.
87336. Gets the number of tests to be printed in the XML report.
87337. scripts/fuse_gtest.py depends on gtest's own header being #included *unconditionally*. Therefore these #includes cannot be moved inside #if GTEST_HAS_PARAM_TEST. Copyright 2008 Google Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. Author: vladl@google.com (Vlad Losev)
87338. Returns true if the death test passed; that is, the test process exited during the test, its exit status matches a user-supplied predicate, and its stderr output matches a user-supplied regular expression. The user-supplied predicate may be a macro expression rather than a function pointer or functor, or else Wait and Passed could be combined.
87339. C++11 specifies that <tuple> provides std::tuple. Use that if gtest is used in C++11 mode and libstdc++ isn't very old (binaries targeting OS X 10.6 can build with clang but need to use gcc4.2's libstdc++).
87340. Returns true iff test passed.
87341. Removes the given event listener from the list and returns it. It then becomes the caller's responsibility to delete the listener. Returns NULL if the listener is not found in the list.
87342. Internal macro for implementing {EXPECT|ASSERT}_PRED_FORMAT5. Don't use this in your code.
87343. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.
87344. For all other compilers, we assume RTTI is enabled.
87345. class CartesianProductGenerator6::Iterator
87346. Makes sure this header is not included before gtest.h.
87347. GTEST_HAS_SEH
87348. On Symbian, BOOST_HAS_TR1_TUPLE causes Boost's TR1 tuple library to use STLport's tuple implementation, which unfortunately doesn't work as the copy of STLport distributed with Symbian is incomplete. By making sure BOOST_HAS_TR1_TUPLE is undefined, we force Boost to use its own tuple implementation.
87349. We disallow copying TestEventListeners.
87350. Implements a subset of TR1 tuple needed by Google Test and Google Mock.
87351. Static utility methods
87352. This file is AUTOMATICALLY GENERATED on 10/31/2011 by command 'gen_gtest_pred_impl.py 5'. DO NOT EDIT BY HAND! Implements a family of generic predicate assertion macros.
87353. Helper function for implementing {EXPECT|ASSERT}_PRED1. Don't use this in your code.
87354. case_name
87355. # of exponent bits in a number.
87356. Returns the number of TestPartResult objects in the array.
87357. class CartesianProductHolder8
87358. GTEST_INCLUDE_GTEST_GTEST_DEATH_TEST_H_ This file was GENERATED by command: pump.py gtest-param-test.h.pump DO NOT EDIT BY HAND!!!
87359. **comment:** If the array has more than kThreshold elements, we'll have to omit some details by printing only the first and the last kChunkSize elements.
TODO(wan@google.com): let the user control the threshold using a flag.
label: code-design
87360. The index for the end() iterator. All the elements in the generated sequence are indexed (0-based) to aid iterator comparison.
87361. The d'tor pops the info pushed by the c'tor. Note that the d'tor is not virtual in order to be efficient. Don't inherit from ScopedTrace!
87362. We disallow copying TestResult.
87363. Constants.
87364. Prints the first N fields of a tuple.
87365. This templated version is for the general case.
87366. The GTEST_COMPILE_ASSERT_ macro can be used to verify that a compile time expression is true. For example, you could use it to verify the size of a static array: GTEST_COMPILE_ASSERT_(GTEST_ARRAY_SIZE_(names) == NUM_NAMES, names_incorrect_size); or to make sure a struct is smaller than a certain size: GTEST_COMPILE_ASSERT_(sizeof(foo) < 128, foo_too_large); The second argument to the macro is the name of the variable. If the expression is false, most compilers will issue a warning/error containing the name of the variable.
87367. Inside a test, refer to TypeParam to get the type parameter. Since we are inside a derived class template, C++ requires use to visit the members of FooTest via 'this'.
87368. Evaluates to the number of elements in 'array'.
87369. This exception is thrown by (and only by) a failed Google Test assertion when GTEST_FLAG(throw_on_failure) is true (if exceptions are enabled). We derive it from std::runtime_error, which is for errors presumably detectable only at run time. Since std::runtime_error inherits from std::exception, many testing frameworks know how to extract and print the message inside it.
87370. other_operand
87371. Finally, you can use INSTANTIATE_TEST_CASE_P to instantiate the test case with any set of parameters you want. Google Test defines a number of functions for generating test parameters. They return what we call (surprise!) parameter generators. Here is a summary of them, which are all in the testing namespace:
Range(begin, end [, step]) - Yields values {begin, begin+step, begin+2*step, ...}. The values do not include end. step defaults to 1. Values(v1, v2, ..., vN) - Yields values {v1, v2, ..., vN}. ValuesIn(container) - Yields values from a C-style array, an STL ValuesIn(begin,end) container, or an iterator range [begin, end).
Bool() - Yields sequence {false, true}. Combine(g1, g2, ..., gN) - Yields all combinations (the Cartesian product for the math savvy) of the values generated by the

N generators. For more details, see comments at the definitions of these functions below in this file. The following statement will instantiate tests from the FooTest test case each with parameter values "meeny", "miny", and "moe".

87372. First, registers the first type-parameterized test in the type list.

87373. Returns a Boolean value indicating whether the caller is currently executing in the context of the death test child process. Tools such as Valgrind heap checkers may need this to modify their behavior in death tests. IMPORTANT: This is an internal utility. Using it may break the implementation of death tests. User code MUST NOT use it.

87374. Forward declarations.

87375. **comment:** Traps C++ exceptions escaping statement and reports them as test failures. Note that trapping SEH exceptions is not implemented here.

label: code-design

87376. Creates a test instance to run. The instance is both created and destroyed within TestInfoImpl::Run()

87377. Causes a trace (including the source file path, the current line number, and the given message) to be included in every test failure message generated by code in the current scope. The effect is undone when the control leaves the current scope. The message argument can be anything streamable to std::ostream. In the implementation, we include the current line number as part of the dummy variable name, thus allowing multiple SCOPED_TRACE()s to appear in the same block - as long as they are on different lines.

87378. Implements type-parameterized tests.

87379. Internal macro for implementing {EXPECT|ASSERT}_PRED_FORMAT2. Don't use this in your code.

87380. Functions producing parameter generators. Google Test uses these generators to produce parameters for value- parameterized tests. When a parameterized test case is instantiated with a particular generator, Google Test creates and runs tests for each element in the sequence produced by the generator. In the following sample, tests from test case FooTest are instantiated each three times with parameter values 3, 5, and 8: class FooTest : public TestWithParam<int> { ... }; TEST_P(FooTest, TestThis) { } TEST_P(FooTest, TestThat) { } INSTANTIATE_TEST_CASE_P(TestSequence, FooTest, Values(3, 5, 8));

87381. The interface for tracing execution of tests. The methods are organized in the order the corresponding events are fired.

87382. These are needed as the Nokia Symbian Compiler cannot decide between const T& and const T* in a function template. The Nokia compiler _can_ decide between class template specializations for T and T*, so a tr1::type_traits-like is_pointer works, and we can overload on that.

87383. These classes and functions are friends as they need to access private members of UnitTest.

87384. **comment:** Environment-describing macros ----- Google Test can be used in many different environments. Macros in this section tell Google Test what kind of environment it is being used in, such that Google Test can provide environment-specific features and implementations. Google Test tries to automatically detect the properties of its environment, so users usually don't need to worry about these macros. However, the automatic detection is not perfect. Sometimes it's necessary for a user to define some of the following macros in the build script to override Google Test's decisions. If the user doesn't define a macro in the list, Google Test will provide a default definition. After this header is #included, all macros in this list will be defined to either 1 or 0. Notes to maintainers: - Each macro here is a user-tweakable knob; do not grow the list lightly. - Use #if to key off these macros. Don't use #ifdef or "#if defined(...)", which will not work as these macros are ALWAYS defined. GTEST_HAS_CLONE - Define it to 1/0 to indicate that clone(2) is/isn't available. GTEST_HAS_EXCEPTIONS - Define it to 1/0 to indicate that exceptions are enabled. GTEST_HAS_GLOBAL_STRING - Define it to 1/0 to indicate that ::string is/isn't available (some systems define ::string, which is different to std::string). GTEST_HAS_GLOBAL_WSTRING - Define it to 1/0 to indicate that ::wstring is/isn't available (some systems define ::wstring, which is different to std::wstring). GTEST_HAS_POSIX_REGEX - Define it to 1/0 to indicate that POSIX regular expressions are/aren't available. GTEST_HAS_PTHREAD - Define it to 1/0 to indicate that <pthread.h> is/isn't available. GTEST_HAS_RTTI - Define it to 1/0 to indicate that RTTI is/isn't enabled. GTEST_HAS_STD_WSTRING - Define it to 1/0 to indicate that std::wstring does/doesn't work (Google Test can be used where std::wstring is unavailable). GTEST_HAS_TR1_TUPLE - Define it to 1/0 to indicate tr1::tuple is/isn't available. GTEST_HAS_SEH - Define it to 1/0 to indicate whether the compiler supports Microsoft's "Structured Exception Handling". GTEST_HAS_STREAM_REDIRECTION - Define it to 1/0 to indicate whether the platform supports I/O stream redirection using dup() and dup2(). GTEST_USE_OWN_TR1_TUPLE - Define it to 1/0 to indicate whether Google Test's own tr1 tuple implementation should be used. Unused when the user sets GTEST_HAS_TR1_TUPLE to 0. GTEST_LANG_CXX11 - Define it to 1/0 to indicate that Google Test is building in C++11/C++98 mode. GTEST_LINKED_AS_SHARED_LIBRARY - Define to 1 when compiling tests that use Google Test as a shared library (known as DLL on Windows). GTEST_CREATE_SHARED_LIBRARY - Define to 1 when compiling Google Test itself as a shared library.

label: code-design

87385. Formats an int value as "%02d". "%02d" for width == 2

87386. Blocks until the controller thread notifies. Must be called from a test thread.

87387. **comment:** MSVC "deprecates" snprintf and issues warnings wherever it is used. In order to avoid these warnings, we need to use _snprintf or _snprintf_s on MSVC-based platforms. We map the GTEST_SNPRINTF_ macro to the appropriate function in order to achieve that. We use macro definition here because snprintf is a variadic function.

label: code-design

87388. It's this header's responsibility to #include <typeinfo> when RTTI is enabled.

87389. GTEST_OS_WINDOWS_MOBILE

87390. string.h is not guaranteed to provide strcpy on C++ Builder.

87391. index

87392. Windows CE does not have the 'ANSI' versions of Win32 APIs. To be able to pass strings to Win32 APIs on CE we need to convert them to 'Unicode', UTF-16.

87393. Compares two C strings. Returns true iff they have the same content. Unlike strcmp(), this function can handle NULL argument(s). A NULL C string is considered different to any non-NULL C string, including the empty string.

87394. Verifies that registered_tests match the test names in defined_test_names_; returns registered_tests if successful, or aborts the program otherwise.

87395. it

87396. This flag causes the Google Test to list tests. None of the tests listed are actually run if the flag is provided.

87397. Generates values from a pair of STL-style iterators. Used in the ValuesIn() function. The elements are copied from the source range since the source can be located on the stack, and the generator is likely to persist beyond that stack frame.

87398. Macros for testing equalities and inequalities. * {ASSERT|EXPECT}_EQ(expected, actual): Tests that expected == actual * {ASSERT|EXPECT}_NE(v1, v2): Tests that v1 != v2 * {ASSERT|EXPECT}_LT(v1, v2): Tests that v1 < v2 * {ASSERT|EXPECT}_LE(v1, v2): Tests that v1 <= v2 * {ASSERT|EXPECT}_GT(v1, v2): Tests that v1 > v2 * {ASSERT|EXPECT}_GE(v1, v2): Tests that v1 >= v2 When they are not, Google Test prints both the tested expressions and their actual values. The values must be compatible built-in types, or you will get a compiler error. By "compatible" we mean that the values can be compared by the respective operator. Note: 1. It is possible to make a user-defined type work with {ASSERT|EXPECT}_??(), but that requires overloading the comparison operators and is thus discouraged by the Google C++ Usage Guide. Therefore, you are advised to use the {ASSERT|EXPECT}_TRUE() macro to assert that two objects are equal. 2. The {ASSERT|EXPECT}_??() macros do pointer comparisons on pointers (in particular, C strings). Therefore, if you use it with two C strings, you are testing how their locations in memory are related, not how their content is related. To compare two C strings by content, use {ASSERT|EXPECT}_STR*(). 3.

{ASSERT|EXPECT}_EQ(expected, actual) is preferred to {ASSERT|EXPECT}_TRUE(expected == actual), as the former tells you what the actual value is when it fails, and similarly for the other comparisons. 4. Do not depend on the order in which {ASSERT|EXPECT}_??() evaluate their arguments, which is undefined. 5. These macros evaluate their arguments exactly once. Examples: EXPECT_NE(5, Foo()); EXPECT_EQ(NULL, a_pointer); ASSERT_LT(i, array_size); ASSERT_GT(records.size(), 0) << "There is no record left.";

87399. Information about a trace point. Implements scoped trace. Opaque implementation of TestInfo Opaque implementation of UnitTest

87400. Always returns true.

87401. Copy constructor. NOLINT

87402. ParamType and GeneratorCreationFunc are private types but are required for declarations of public methods AddTestPattern() and AddTestCaseInstantiation().

87403. MSVC 8.0, Sun C++, and IBM XL C++ have a bug which causes the above definition to fail to remove the const in 'const int[3]' and 'const char[3][4]'. The following specialization works around the bug.

87404. is not a pointer

87405. GTEST_HAS_TYPED_TEST

87406. Tests that an exit code describes an exit due to termination by a given signal.

87407. Streams a pointer value to this object. This function is an overload of the previous one. When you stream a pointer to a Message, this definition will be used as it is more specialized. (The C++ Standard, section [temp.func.order].) If you stream a non-pointer, then the previous definition will be used. The reason for this overload is that streaming a NULL pointer to ostream is undefined behavior. Depending on the compiler, you may get "0", "(nil)", "(null)", or an access violation. To ensure consistent result across compilers, we always treat NULL as "(null)".

87408. Then, use TYPED_TEST() instead of TEST_F() to define as many typed tests for this test case as you want.

87409. INTERNAL IMPLEMENTATION - DO NOT USE IN USER CODE. Expands to the name of the variable used to remember the names of the defined tests in the given test case.

87410. TypeParameterizedTestCase<Fixture, Tests, Types>::Register() registers *all combinations* of 'Tests' and 'Types' with Google Test. The return value is insignificant - we just need to return something such that we can call this function in a namespace scope.

87411. Adds reference to a type if it is not a reference type, otherwise leaves it unchanged. This is the same as tr1::add_reference, which is not widely available yet.

87412. Appends the user-supplied message to the Google-Test-generated message.

87413. With the following statement, during unqualified name lookup, testing::internal2::operator<< appears as if it was declared in the nearest enclosing namespace that contains both ::testing_internal and ::testing::internal2, i.e. the global namespace. For more details, refer to the C++ Standard section 7.3.4-1 [namespace.udir]. This allows us to fall back onto testing::internal2::operator<< in case T doesn't come with a << operator. We cannot write 'using ::testing::internal2::operator<<;', which gcc 3.3 fails to compile due to a compiler bug. NOLINT

87414. Overloads for various char types.

87415. Overload for C arrays. Multi-dimensional arrays are printed properly.

87416. class CartesianProductGenerator5::Iterator

87417. isspace(int ch) and friends accept an unsigned char or EOF. char may be signed, depending on the compiler (or compiler flags). Therefore we need to cast a char to unsigned char before calling isspace(), etc.

87418. You can inherit all the usual members for a non-parameterized test fixture here.

87419. An enumeration of possible roles that may be taken when a death test is encountered. EXECUTE means that the death test logic should be executed immediately. OVERSEE means that the program should prepare the appropriate environment for a child process to execute the death test, then wait for it to complete.

87420. Gets the name of the source file where the test part took place, or NULL if it's unknown.

87421. t1

87422. Macros for comparing floating-point numbers. * {ASSERT|EXPECT}_FLOAT_EQ(expected, actual): Tests that two float values are almost equal. * {ASSERT|EXPECT}_DOUBLE_EQ(expected, actual): Tests that two double values are almost equal. * {ASSERT|EXPECT}_NEAR(v1, v2, abs_error): Tests that v1 and v2 are within the given distance to each other. Google Test uses ULP-based comparison to automatically pick a default error bound that is appropriate for the operands. See the FloatingPoint template class in gtest-internal.h if you are interested in the implementation details.

87423. Macro for referencing flags.

87424. **comment:** Even though we don't use this macro any longer, we keep it in case some clients still depend on it.
label: code-design

87425. Runs all tests in this UnitTest object and prints the result. Returns 0 if successful, or 1 otherwise. This method can only be called from the main thread. INTERNAL IMPLEMENTATION - DO NOT USE IN A USER PROGRAM.

87426. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87427. DestroyDB(db_path_, Options());

87428. check with the default rocksdb options and expect failure

87429. override the following options as db_bench will not take these options from the options file

87430. namespace

87431. #ifdef GFLAGS

87432. namespace rocksdb

87433. normal run with universal compaction mode

87434. total time for this script to test db_stress

87435. normal run

87436. use large ops per thread since we will kill it anyway

87437. run with kill_random_test, with three modes. Mode 0 covers all kill points. Mode 1 covers less kill points but increases chance of triggering them. Mode 2 covers even less frequent kill points and further increases triggering chance.

87438. **comment:** normal run with FIFO compaction mode ops_per_thread is divided by 5 because FIFO compaction style is quite a bit slower on reads with lot of files
label: code-design

87439. params overwrite priority: for default: default_params < blackbox|whitebox_default_params < args for simple: simple_default_params < blackbox|whitebox_simple_default_params < args

87440. we need to clean up after ourselves -- only do this on test success

87441. time for one db_stress instance to run

87442. **comment:** TODO: May need to adjust random odds if kill_random_test is too small.
label: test

87443. This script runs and kills db_stress multiple times. It checks consistency in case of unsafe crashes in RocksDB.

87444. we need to clean up after ourselves -- only do this on test success

87445. time to stabilize before the next run

87446. since we will be killing anyway, use large value for ops_per_thread

87447. time to stabilize after a kill

87448. we expect negative retncode if kill option was given

87449. First half of the duration, keep doing kill test. For the next half, try different modes.

87450. This python script runs db_stress multiple times. Some runs with kill_random_test that causes rocksdb to crash at various points in code.

87451. ! /usr/bin/env python

87452. we expect zero retncode if no kill option

87453. Run kill mode 0, 1 and 2 by turn.

87454. **comment:** Suppress false positive clang static analyzer warnings.
label: code-design

87455. ROCKSDB_LITE

87456. __clang_analyzer__

87457. namespace

87458. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87459. namespace rocksdb

87460. map input to expected outputs. odd number of "hex" half bytes doesn't make sense

87461. ROCKSDB_LITE

87462. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87463. Test on valid dbPath.

87464. It is weird that GET and SCAN raise exception for non-existent key, while delete does not

87465. non-existing column family.

87466. Dump only a portion of the key range

87467. No param called he

87468. Delete the file

87469. Load into an existing db, create_if_missing is not specified

87470. **comment:** Dump command fails because of typo in params
label: documentation

87471. Running it with the copy we just created should pass.

87472. db not specified

87473. Pattern to expect from SST dump.

87474. Pattern to expect from WAL dump.

87475. max_keys is not applicable for put
87476. Pattern to expect from manifest_dump.
87477. Test with the default manifest file in dbPath.
87478. **comment:** These tests need to be improved; for example with asserts about whether compaction or level reduction actually took place.
 label: code-design
87479. Test with multiple manifest files in dbPath.
87480. No results => FAIL
87481. Dump with count_only
87482. DB should have atleast one value for scan to work
87483. Test on empty path.
87484. Dump and load with WAL disabled
87485. **comment:** TODO(dilip): Not sure what should be passed to WAL.Currently corrupted.
 label: code-design
87486. Just some paranoia
87487. hex has invalid boolean value
87488. Dump and load with lots of extra params specified
87489. fails because timestamp's length is greater than value's
87490. Make sure that using the dump with --path will result in identical output as just using manifest_dump.
87491. Given non-default column family to single CF DB.
87492. Dump upto max_keys rows
87493. Dump and load in hex
87494. Modify the file
87495. Dump and load without any additional params specified
87496. Delete
87497. **comment:** Clean up test database
 label: code-design
87498. Dump
87499. Get and Put
87500. WriteBatch
87501. !ROCKSDB_LITE
87502. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
87503. use `which time` to avoid using bash's internal time command
87504. # run the db_bench
87505. time
87506. \$1 --- name of the benchmark \$2 --- the filename of the output log of db_bench
87507. # make sure no db_bench is running The following statement is necessary make sure "eval \$ps_cmd" will success. Otherwise, if we simply check whether "\$(eval \$ps_cmd | grep db_bench)" is successful or not, then it will always be false since grep will return non-zero status when there's no matching output.
87508. ======
87509. perform the actual command to check whether db_bench is running
87510. Parse the output of the time command
87511. this will terminate all currently-running db_bench
87512. key range
87513. Obtain micros / op
87514. checkpoint cannot build in directory already exists
87515. Obtain percentile information
87516. \$1 --- benchmark name \$2 --- number of operations. Default: \$NUM_KEYS \$3 --- number of threads. Default \$NUM_THREADS \$4 --- use_existing_db. Default: 1 \$5 --- update_report. Default: 1
87517. #####
87518. !/usr/bin/env bash The RocksDB regression test script. REQUIREMENT: must be able to run make db_bench in the current directory This script will do the following things in order: 1. check out the specified rocksdb commit. 2. build db_bench using the specified commit 3. setup test directory \$TEST_PATH. If not specified, then the test directory will be "/tmp/rocksdb/regression_test" 4. run set of benchmarks on the specified host (can be either locally or remotely) 5. generate report in the \$RESULT_PATH. If RESULT_PATH is not specified, RESULT_PATH will be set to \$TEST_PATH/current_time = Examples = * Run the regression test using rocksdb commit abcdef that outputs results and temp files in "/my/output/dir" r TEST_PATH=/my/output/dir COMMIT_ID=abcdef ./tools/regression_test.sh * Run the regression test on a remot host under "/my/output/dir" directory and stores the result locally in "/my/benchmark/results" using commit abcdef and with the rocksdb options specified in /my/path/to/OPTIONS-012345 with 1000000000 keys in each benchmark in the regression test where each key and value are 100 and 900 bytes respectively: REMOTE_USER_AT_HOST=yhchiang@my.remote.host \ TEST_PATH=/my/output/dir \ RESULT_PATH=/my/benchmark/results \ COMMIT_ID=abcdef \ OPTIONS_FILE=/my/path/to/OPTIONS-012345 \ NUM_KEYS=1000000000 \ KEY_SIZE=100 \ VALUE_SIZE=900 \ ./tools/regression_test.sh = Regression test environmental parameters = DEBUG: If true, then the script will not checkout master and build db_bench if db_bench already exists Default: 0 TEST_MODE: If 1, run fillseqdeterministic and benchmarks both if 0, only run fillseqdeterministic if 2, only run benchmarks Default: 1 TEST_PATH: the root directory of the regression test. Default: "/tmp/rocksdb/regression_test" RESULT_PATH: the directory where the regression results will be generated. Default: "\$TEST_PATH/current_time" REMOTE_USER_AT_HOST: If set, then test will run on the specified host under TEST_PATH directory and outputs test results locally in RESULT_PATH The REMOTE_USER_AT_HOST should follow the format user-id@host.name DB_PATH: the path where the rocksdb database will be created during the regression test. Default: \$TEST_PATH/db WAL_PATH: the path where the rocksdb WAL will be outputed. Default: \$TEST_PATH/wal OPTIONS_FILE: If specified, then the regression test will use the specified file to initialize the RocksDB options in its benchmarks. Note that this feature only work for commits after 88acd93 or rocksdb version later than 4.9. DELETE_TEST_PATH: If true, then the test directory will be deleted after the script ends. Default: 0 = db_bench parameters = NUM_THREADS: The number of concurrent foreground threads that will issue database operations in the benchmark. Default: 16. NUM_KEYS: The key range that will be used in the entire regression test. Default: 1G. NUM_OPS: The number of operations (reads, writes, or deletes) that will be issued in EACH thread. Default: \$NUM_KEYS / \$NUM_THREADS KEY_SIZE: The size of each key in bytes in db_bench. Default: 100. VALUE_SIZE: The size of each value in bytes in db_bench. Default: 900. CACHE_SIZE: The size of RocksDB block cache used in db_bench. Default: 1G STATISTICS: If 1, then statistics is on in db_bench. Default: 0. COMPRESSION_RATIO: The compression ratio of the key generated in db_bench. Default: 0.5. HISTOGRAM: If 1, then the histogram feature on performance feature is on. STATS_PER_INTERVAL: If 1, then the statistics will be reported for every STATS_INTERVAL_SECONDS seconds. Default 1. STATS_INTERVAL_SECONDS: If STATS_PER_INTERVAL is set to 1, then statistics will be reported for every STATS_INTERVAL_SECONDS. Default 60. MAX_BACKGROUND_FLUSHES: The maximum number of concurrent flushes in db_bench. Default: 4. MAX_BACKGROUND_COMPACTION: The maximum number of concurrent compactions in db_bench. Default: 16. SEEK_NEXTS: Controls how many Next() will be called after seek. Default: 10. SEED: random seed that controls the randomness of the benchmark. Default: \$(date +%s)
87519. ====== CONSTANT ======
87520. compactall alone will not print ops or threads, which will fail update_report
87521. p50 p75 p99 p99.9 p99.99
87522. Verify that the sample dump file is undumpable and then redumpable.
87523. Test for sst dump tool "raw" mode
87524. !ROCKSDB_LITE return RUN_ALL_TESTS();
87525. Populate slightly more than 1K keys
87526. namespace
87527. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2012 The LevelDB Authors. All rights reserved. Use of

this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87528. namespace rocksdb

87529. The value returned by Arena::MemoryAllocatedBytes() may be greater than the requested memory. We choose a somewhat arbitrary upper bound of max_expected = expected * 1.1 to detect critical overallocation.

87530. Check the "i"th allocation for the known bit pattern

87531. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87532. Our arena disallows size 0 allocations.

87533. **comment:** requested size < quarter of a block: allocate a block with the default size, then try to use unused part of the block. So one new block will be allocated for the first Allocate(99) call. All the remaining calls won't lead to new allocation.

label: code-design

87534. requested size block size

87535. requested size > quarter of a block: allocate requested size separately

87536. Make sure we didn't count the allocate but not used memory space in Arena::ApproximateMemoryUsage()

87537. first allocation

87538. allocate inline bytes

87539. namespace

87540. namespace rocksdb

87541. requested size > size of a block: allocate requested size separately

87542. Fill the "i"th allocation with a known bit pattern

87543. Now the log file will be rolled

87544. Log some headers explicitly using Header()

87545. Also make sure the log size is increasing.

87546. start=

87547. reopens the log file and an empty log file will be created.

87548. OS_WIN

87549. -- Make the log file expire

87550. If only 'log_max_size' options is specified, then every time when rocksdb is restarted, a new empty log file will be created.

87551. Normal logger

87552. Only roll by size

87553. Log enough data to cause a roll over

87554. namespace rocksdb

87555. Make sure we always write to the same log file (by checking the create time);

87556. log_file_time_to_roll=

87557. Flush the log for the latest file

87558. namespace

87559. -- Test the existence of file during the server restart.

87560. measure the size of each message, which is supposed to be equal or greater than log_message.size()

87561. **comment:** TODO: does not build for Windows because of PosixLogger use below. Need to port

label: code-design

87562. Return the number of lines where a given pattern was found in the file

87563. again, messages with level smaller than log_level will not be logged.

87564. IROCKSDB_LITE

87565. **comment:** HEADER_LEVEL should make this behave like calling Header()

label: code-design

87566. Test the ability to roll by size

87567. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87568. **comment:** Since rotation is checked before actual logging, we need to trigger the rotation by logging another message.

label: code-design

87569. **comment:** Replace all slashes in the path so windows CompSpec does not become confused

label: code-design

87570. roll by both Time and size

87571. db_log_dir=

87572. **comment:** an extra-scope to force the AutoRollLogger to flush the log file when it becomes out of scope.

label: code-design

87573. Test the ability to roll by Time

87574. Test the logger Header function for roll over logs We expect the new logs creates as roll over to carry the headers specified

87575. verify that the old log contains all the header logs

87576. Test the cases when the log file will not be rolled.

87577. verify that the files rolled over

87578. Only roll by Time

87579. **comment:** In this test we only want to Log some simple log message with no format. LogMessage() provides such a simple interface and avoids the [format-security] warning which occurs when you call ROCKS_LOG_INFO(logger, log_message) directly.

label: code-design

87580. -- Write to the log for several times, which is supposed to be finished before time.

87581. **comment:** WORKAROUND: avoid compiler's complaint of "comparison between signed and unsigned integer expressions" because literal 0 is treated as "singed".

label: code-design

87582. Notes: (1) Need to pin the old logger before beginning the roll, as rolling grabs the mutex, which would prevent us from accessing the old logger. This also marks flush_thread with AutoRollLogger::Flush::PinnedLogger. (2) Need to reset logger during PosixLogger::Flush() to exercise a race condition case, which is executing the flush with the pinned (old) logger after auto-roll logger has cut over to a new logger. (3) PosixLogger::Flush() happens in both threads but its SyncPoints only are enabled in flush_thread (the one pinning the old logger).

87583. We know that the old log files are named [path]<something>. Return all entities that match the pattern

87584. log messages with log level smaller than log_level will not be logged.

87585. test_num == 0 -> standard call to Header() test_num == 1 -> call to Log() with InfoLogLevel::HEADER_LEVEL

87586. At this time, the new log file should be created.

87587. We run same operations for kOps times in order to get a more fair result.

87588. **comment:** This test case only reports the performance between std::vector<std::string> and autovector<std::string>. We chose string for comparison because in most of our use cases we used std::vector<std::string>.

label: code-design

87589. **comment:** Creation and insertion test Test the case when there is: * no element inserted: internal array of std::vector may not really get initialized. * one element inserted: internal array of std::vector must have initialized. * kSize elements inserted. This shows the most time we'll spend if we keep everything in stack. * 2 * kSize elements inserted. The internal vector of autovector must have been initialized.

label: code-design

87590. will always be in heap

87591. forward and backward

87592. Cannot use ASSERT_EQ since that macro depends on iostream serialization
87593. Test both heap-allocated and stack-allocated cases.
87594. pre-generated unique keys
comment: HACK: make sure -> works
label: code-design
87595. **comment:** HACK avoid compiler's optimization to ignore total
label: code-design
87597. namespace
87598. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
87599. Sequence Access Test
87600. const iterator
87601. namespace rocksdb
87602. non-const iterator
87603. basic operator test
87604. !ROCKSDB_LITE
87605. GFLAGS
87606. Must not be over 2% Allowed, but not too often
87607. Allowed, but not too often
87608. Check false positive rate
87609. All added keys must match
87610. Must not be over 2%
87611. Count number of filters that significantly exceed the false positive rate
87612. Empty filter is not match, at this level
87613. Different bits-per-byte
87614. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2012 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
87615. namespace rocksdb
87616. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
87617. Some special values
87618. Test values near powers of two
87619. Construct the list of values to check
87620. **comment:** Test that encoding routines generate little-endian encodings
label: code-design
87621. namespace rocksdb
87622. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
87623. From rfc3720 section B.4.
87624. namespace crc32c namespace rocksdb
87625. When we end up with 26 files in trash we will start deleting new files immediately
87626. Tests in this file are for DeleteScheduler component and dont create any DBs, so we need to use set this value to 100% (instead of default 25%)
87627. Deleting 100 files will need >28 hours to delete we will delete the DeleteScheduler while delete queue is not empty
87628. **comment:** 100 files 100 KB as a file size 1 byte per sec (very slow trash delete)
label: code-design
87629. Hold BackgroundEmptyTrash
87630. 1- Create 10 dummy files 2- Delete the 10 files using DeleteScheduler (move them to trsah) 3- Delete the 10 files directly (using env_->DeleteFile) --- Hold DeleteScheduler::BackgroundEmptyTrash --- 4- Make sure that DeleteScheduler failed to delete the 10 files and reported 10 background errors
87631. Generate 10 dummy files and move them to trash
87632. namespace rocksdb
87633. **comment:** Delete dummy files using 10 threads and measure time spent to empty trash
label: code-design
87634. Testing that moving files to trash with the same name is not a problem 1- Create 10 files with the same name "conflict.data" 2- Delete the 10 files using DeleteScheduler 3- Make sure that trash directory contain 10 files ("conflict.data" x 10) --- Hold DeleteScheduler::BackgroundEmptyTrash --- 4- Make sure that files are deleted from trash
87635. 1 Kb / sec
87636. Create "conflict.data" and move it to trash 10 times
87637. **comment:** Delete dummy files and measure time spent to empty trash
label: code-design
87638. 100 files every file is 1 kb
87639. 10 files every file is 1 kb
87640. 1 Byte / sec
87641. 10 files ("conflict.data" x 10) in trash
87642. 1 MB / sec
87643. Test the basic functionality of DeleteScheduler (Rate Limiting). 1- Create 100 dummy files 2- Delete the 100 dummy files using DeleteScheduler --- Hold DeleteScheduler::BackgroundEmptyTrash --- 3- Wait for DeleteScheduler to delete all files in trash 4- Verify that BackgroundEmptyTrash used to correct penlties for the files 5- Make sure that all created files were completely deleted
87644. 1- Delete the trash directory 2- Delete 10 files using DeleteScheduler 3- Make sure that the 10 files were deleted immediately since DeleteScheduler failed to move them to trash directory
87645. Same as the BasicRateLimiting test but delete files in multiple threads. 1- Create 100 dummy files 2- Delete the 100 dummy files using DeleteScheduler using 10 threads --- Hold DeleteScheduler::BackgroundEmptyTrash --- 3- Wait for DeleteScheduler to delete all files in queue 4- Verify that BackgroundEmptyTrash used to correct penlties for the files 5- Make sure that all created files were completely deleted
87646. ROCKSDB_LITE
87647. 10 files per thread every file is 1 kb
87648. **comment:** 1- Create a DeleteScheduler with very slow rate limit (1 Byte / sec) 2- Delete 100 files using DeleteScheduler 3- Delete the DeleteScheduler (call the destructor while queue is not empty) 4- Make sure that not all files were deleted from trash and that DeleteScheduler background thread did not delete all files
label: code-design
87649. Move files to trash, wait for empty trash, start again
87650. 1 Mb/sec
87651. 1- Create 10 dummy files 2- Delete 10 dummy files using DeleteScheduler 3- Wait for DeleteScheduler to delete all files in queue 4- Make sure all files in trash directory were deleted 5- Repeat previous steps 5 times
87652. Create 100 dummy files, every file is 1 Kb
87653. We will delete the trash directory, that mean that DeleteScheduler wont be able to move files to trash and will delete files them immediately.
87654. Delete 10 files from trash, this will cause background errors in BackgroundEmptyTrash since we already deleted the files it was goind to delete

87655. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87656. Disable rate limiting initially

87657. Disable rate limiting by setting `rate_bytes_per_sec_` to 0 and make sure that when DeleteScheduler delete a file it delete it immediately and dont move it to trash

87658. Ignore " " and ".."

87659. Every file we delete will be deleted immediately

87660. GFLAGS

87661. Allowed, but not too often

87662. Check false positive rate

87663. All added keys must match

87664. Count number of filters that significantly exceed the false positive rate

87665. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87666. Locality enabled version

87667. namespace rocksdb

87668. namespace rocksdb

87669. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87670. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright 2014 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87671. This test uses a custom Env to keep track of the state of a filesystem as of the last "sync". It then checks for data loss errors by purposely dropping file data (or entire files) not protected by a "sync".

87672. Return pair <parent directory name, file name> of a full path.

87673. WritableFileWriter* file is opened again then it will be truncated - so forget our saved state.

87674. Assume a filename, and not a directory name like "/foo/bar/"

87675. For every file that is not fully synced, make a call to `func` with FileState of the file as the parameter.

87676. A basic file truncation function suitable for this test.

87677. Not allow overwriting files

87678. Because DeleteFile access this container make a copy to avoid deadlock

87679. namespace rocksdb

87680. No need to actual sync.

87681. Trim the tailing "/" in the end of `str`

87682. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright 2014 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87683. This test uses a custom Env to keep track of the state of a filesystem as of the last "sync". It then checks for data loss errors by purposely dropping file data (or entire files) not protected by a "sync".

87684. A wrapper around WritableFileWriter* file is written to or sync'ed.

87685. For every file that is not fully synced, make a call to `func` with FileState of the file as the parameter.

87686. UTILFAULT_INJECTION_TEST_ENV_H_

87687. namespace rocksdb

87688. Setting the filesystem to inactive is the test equivalent to simulating a system reset. Setting to inactive will freeze our saved filesystem state so that it will stop being recorded. It can then be reset back to the state at the time of the reset.

87689. Record flushes, syncs, writes

87690. Next call to WritableFile::Append() should fail

87691. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87692. Make sure random writes generated enough writes.

87693. In some attempts, flush in a chance of 1/10.

87694. Flush in a chance of 1/10.

87695. namespace rocksdb

87696. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87697. acquire a lock on a file

87698. re-acquire the lock on the same file. This should fail.

87699. release the lock

87700. namespace rocksdb

87701. The hash algorithm is part of the file format, for example for the Bloom filters. Test that the hash values are stable for a set of random strings of varying lengths.

87702. Same as BloomHash.

87703. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2012 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87704. Basic test, `MAX_VALUE = 3*MAX_HEAP_SIZE` (occasional duplicates)

87705. namespace rocksdb

87706. This test performs the same pseudorandom sequence of operations on a BinaryHeap and an std::priority_queue, comparing output. The three possible operations are insert, replace top and pop. Insert is chosen slightly more often than the others so that the size of the heap slowly grows. Once the size heats the `MAX_HEAP_SIZE` limit, we disallow inserting until the heap becomes empty, testing the "draining" scenario.

87707. GFLAGS

87708. * Compares the custom heap implementation in util/heap.h against * std::priority_queue on a pseudo-random sequence of operations.

87709. After every operation, check that the public methods give the same results

87710. hit max size, draining until we empty the heap

87711. insert

87712. pop

87713. One-element heap

87714. Probabilities should be set up to occasionally hit the max heap size and drain it

87715. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87716. Mid-size heap with small values (many duplicates)

87717. Small heap, large value range (no duplicates)

87718. replace top

87719. Two-element heap

87720. stats

87721. fairness

87722. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87723. **comment:** The idea behind is to start a request first, then before it refills, update limit to a different value (2X/0.5X). No starvation should be guaranteed under any situation TODO(lightmark): more test cases are welcome.

label: test

87724. second iteration changes the target dynamically

87725. Test for 2 seconds

87726. refill per second

87727. starvation test when limit changes to a smaller value

87728. bytes

87729. refill_period_us

87730. rate_bytes_per_sec

87731. namespace rocksdb

87732. **comment:** TODO(yhchiang): the rate will not be accurate when we run test in parallel.

label: test

87733. Return the current option configuration.

87734. Destroy using last options

87735. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87736. namespace

87737. namespace rocksdb

87738. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87739. int options

87740. Duplicate the random data until we have filled "len" bytes

87741. Make sure to generate a wide variety of characters so we test the boundary conditions for short-key optimizations.

87742. size_t options

87743. uint64_t options

87744. vector int options

87745. custom typed options

87746. double options

87747. namespace test namespace rocksdb

87748. ' ' .. ' ~'

87749. namespace

87750. boolean options

87751. uint32_t options

87752. std::string options

87753. unsigned int options

87754. pointer typed options

87755. !ROCKSDB_LITE

87756. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

87757. Return a random key with the specified length that may contain interesting characters (e.g. \x00, \xff, etc.).

87758. A test comparator which compare two strings in this way: (1) first compare prefix of 8 bytes in alphabet order, (2) if two strings share the same prefix, sort the other part of the string in the reverse alphabet order. This helps simulate the case of compounded key of [entity][timestamp] and latest timestamp first.

87759. The following text is boilerplate that forwards all methods to target()

87760. A dummy merge operator which can change its name

87761. namespace test namespace rocksdb

87762. An internal comparator that just forward comparing results from the user comparator in it. Can be used to test entities that have no dependency on internal key structure but consumes InternalKeyComparator, like BlockBasedTable.

87763. Returns a dummy merge operator with random name.

87764. Iterator over a vector of keys/values

87765. A wrapper around a StringSink to give it a RandomRWFile interface

87766. Like StringSink, this writes into a string. Unlink StringSink, it has some initial content and overwrites it, just like a recycled log file.

87767. **comment:** TODO(yhchiang): Currently doesn't handle the overflow case.

label: code-design

87768. A wrapper that allows injection of errors.

87769. Signalled when background work finishes

87770. Store in *dst a string of length "len" that will compress to "N*compressed_fraction" bytes and return a Slice that references the generated data.

87771. Randomly initialize the given DBOptions

87772. Randomly initialize the given ColumnFamilyOptions Note that the caller is responsible for releasing non-null cf_opt->compaction_filter.

87773. Store in *dst a random string of length "len" and return a Slice that references the generated data.

87774. A dummy compaction filter factory which can change its name

87775. Returns a name that identifies this compaction filter factory.

87776. Filters merge operands and values that are equal to `num`.

87777. Corrupts key by changing the type

87778. Returns a dummy compaction filter with a random name.

87779. **comment:** Returns a user key comparator that can be used for comparing two uint64_t slices. Instead of comparing slices byte-wise, it compares all the 8 bytes at once. Assumes same endian-ness is used though the database's lifetime. Symantics of comparison would differ from Bytewise comparator in little endian machines.

label: code-design

87780. A dummy compaction filter which can change its name

87781. Verify none of the threads are running

87782. simulated tasks

87783. ROCKSDB_USING_THREAD_STATUS

87784. terminate compaction-write tasks and see if the thread-status reflects this update

87785. verify the global tables for operations and states are properly indexed.

87786. schedule the simulated tasks

87787. terminate compaction-wait tasks and see if the thread-status reflects this update

87788. setup right answers

87789. terminate flush-write tasks and see if the thread-status reflects this update

87790. namespace

87791. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87792. namespace rocksdb

87793. verify the thread-status

87794. Verify the number of running threads in each pool.

87795. Scrape all thread local data. No unref at thread exit or ThreadLocalPtr destruction
87796. Initiate 2 instances: one keeps writing and one keeps reading. The read instance should not see data from the write instance. Each thread local copy of the value are also different from each other.
87797. pick up 2
87798. additional N threads exit unref for the left instance
87799. next new id
87800. Waiting for instruction to exit thread
87801. Suppress false positive clang analyzer warnings.
87802. instance destroy for N threads
87803. Now we have 3, 1 in queue pick up 1
87804. The following test is disabled as it requires manual steps to run it correctly. Currently we have no way to access SyncPoint w/o ASAN error when the child thread dies after the main thread dies. So if you manually enable this test and only see an ASAN error on SyncPoint, it means you pass the test.
87805. **comment:** Id used 1
 label: code-design
87806. namespace rocksdb
87807. anonymous namespace
87808. pick up 1
87809. Fail Swap, still 2
87810. Swap in 3
87811. **comment:** Id used 2
 label: code-design
87812. pick up 3
87813. return up 1
87814. Triggers the initialization of singletons.
87815. **comment:** Id used 0
 label: code-design
87816. namespace
87817. N threads x 2 ThreadLocal instance cleanup on thread exit
87818. Signal to exit
87819. Case 1: unref triggered by thread exit
87820. Let write threads write a different value from the read threads
87821. Another new thread, read/write should not see value from previous thread
87822. ROCKSDB_LITE
87823. Wait for all threads to finish using Params
87824. Swap in 2
87825. __clang_analyzer__
87826. After exit, id sequence in queue: 3, 1, 2, 0
87827. Loop for 1 second
87828. Case 0: no unref triggered if ThreadLocalPtr is never accessed
87829. New ThreadLocal instance bumps id by 1
87830. global id list carries over 3, 1, 2, 0
87831. **comment:** Id used 3
 label: code-design
87832. Size_T switches size along with the ptr size we want to cast to.
87833. pick up 0
87834. Case 2: unref triggered by ThreadLocal instance destruction
87835. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
87836. Verify Fold() behavior
87837. id 3, 2, 1, 0 are in the free queue in order
87838. Now destroy one ThreadLocal instance
87839. The person who associated a work with this deed has dedicated the work to the public domain by waiving all of his or her rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law. You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission. See Other Information below.
87840. ////////////
87841. namespace Timing
87842. auto ret = q.cancel(id); assert(ret == 1); q.cancelAll();
87843. borrowed from <http://www.crazygaze.com/blog/2016/03/24/portable-c-timer-queue/> Timer Queue License The source code in this article is licensed under the CC0 license, so feel free to copy, modify, share, do whatever you want with it. No attribution is required, but I'll be happy if you do. CC0 license
87844. Portions Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
87845. namespace rocksdb
87846. Have not yet written to this key, so assume its value is 0
87847. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
87848. key format: [SET#][random#]
87849. Non-optimistic transactions should only fail due to expiration or write failures. For testing purposes, we do not expect any write failures.
87850. For each set, pick a key at random and increment it
87851. return success if we didn't get any unexpected errors
87852. Optimistic transactions can have write-conflict errors on commit. Any other error is unexpected.
87853. For each set of keys with the same prefix, sum all the values
87854. ROCKSDB_LITE
87855. stop when we reach a different prefix
87856. prefix_buf needs to be large enough to hold a uint16 in string form
87857. Found key, parse its value
87858. Optimistic transactions should never return non-ok status here. Non-optimistic transactions may return write-conflict/timeout errors.
87859. Since we did a GetForUpdate, Put should not fail.
87860. Pad prefix appropriately so we can iterate over each set
87861. pick a random number to use to increment a key in each set
87862. Increment key
87863. Returns the number of successfully written calls to TransactionDBInsert/OptimisticTransactionDBInsert/DBInsert
87864. Returns the number of calls to TransactionDBInsert/OptimisticTransactionDBInsert/DBInsert that did not write any data.
87865. Number of successful insert batches performed
87866. Increment a key in each set without using a transaction. If this function is called in parallel, then Verify() may fail. Returns true if the write succeeds. Error status may be obtained by calling GetLastStatus().
87867. optimization: re-use allocated transaction objects.
87868. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

87869. Returns OK if Invariant is true.
87870. Increment a key in each set using a Transaction on a TransactionDB. Returns true if the transaction succeeded OR if any error encountered was expected (eg a write-conflict). Error status may be obtained by calling GetLastStatus();
87871. Utility class for stress testing transactions. Can be used to write many transactions in parallel and then validate that the data written is logically consistent. This class assumes the input DB is initially empty. Each call to TransactionDBInsert() / OptimisticTransactionDBInsert() will increment the value of a key in #num_sets sets of keys. Regardless of whether the transaction succeeds, the total sum of values of keys in each set is an invariant that should remain equal. After calling TransactionDBInsert() / OptimisticTransactionDBInsert() many times, Verify() can be called to validate that the invariant holds. To test writing Transaction in parallel, multiple threads can create a RandomTransactionInserter with similar arguments using the same DB.
87872. Input options
87873. Returns the status of the previous Insert operation
87874. Status returned by most recent insert operation
87875. ROCKSDB_LITE
87876. Number of failed insert batches attempted
87877. namespace rocksdb
87878. Increment a key in each set using a Transaction on an OptimisticTransactionDB Returns true if the transaction succeeded OR if any error encountered was expected (eg a write-conflict). Error status may be obtained by calling GetLastStatus();
87879. num_keys is the number of keys in each set. num_sets is the number of sets of keys.
87880. Read from meta-file failure
87881. test various kind of corruptions that may happen: 1. Not able to write a file for backup - that backup should fail, everything else should work 2. Corrupted backup meta file or missing backedup file - we should not be able to open that backup, but all other backups should be fine 3. Corrupted checksum value - if the checksum is not a valid uint32_t, db open should fail, otherwise, it aborts during the restore process.
87882. should write 5 DB files + one meta file
87883. implicit
87884. pair<filename, alive?>
87885. test purge old backups when i == 4, purge to only 1 backup when i == 3, purge to 2 backups
87886. Verify manifest can roll while a backup is being created with the old manifest.
87887. set up backup db options
87888. set share_files_with_checksum to true and do some more backups
87889. has to be a big number, so that it triggers the memtable flush
87890. what business do you have calling this method?
87891. create backups 1, 2, 3, 4, 5
87892. Backup the LATEST options file to "<backup_dir>/private/<backup_id>/OPTIONS<number>"
87893. files
87894. 1GB
87895. Test that we properly propagate Env failures
87896. namespace rocksdb
87897. MANIFEST file size should be only 100
87898. The last manifest roll would've already been cleaned up by the full scan that happens when CreateNewBackup invokes EnableFileDeletions. We need to trigger another roll to verify non-full scan purges stale manifests.
87899. ----- case 2. corrupted backup meta or missing backedup file -----
87900. 128KB
87901. backup overwrites file 000007.sst
87902. delete old data
87903. utility functions
87904. ----- case 3. - corrupt a file -----
87905. Verify the specified max backups are opened, including skipping over corrupted backups. Setup: - backups 1, 2, and 4 are valid - backup 3 is corrupt - max_valid_backups_to_open == 2 Expectation: the engine opens backups 4 and 2 since those are latest two non-corrupt backups.
87906. now, the backup can succeed
87907. should never get here
87908. no failure
87909. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
87910. share_table_files
87911. !defined(ROCKSDB_LITE) && !defined(OS_WIN)
87912. ---- restore the DB ----
87913. TestEnv
87914. new backup should be 2!
87915. this will make sure that backup does not copy the same file twice
87916. checksum of backup 3 is an invalid value, this can be detected at db open time, and it reverts to the previous backup automatically
87917. This test verifies that the verifyBackup method correctly identifies invalid backups
87918. Create logger
87919. 00011.sst was only in backup 1, should be deleted
87920. delete old files in db
87921. should write 4 new DB files + one meta file should not write/copy 00010.sst, since it's already there!
87922. make sure that no corrupt backups have actually been deleted!
87923. To avoid FlushWAL called on stacked db which is nullptr
87924. ----- case 1. - valid backup -----
87925. size_bytes
87926. we should get consistent results with flush_before_backup set to both true and false
87927. open DB, write, backup, write, backup, close, restore
87928. Verify that you can backup and restore with share_files_with_checksum on
87929. should not open 00010.sst - it's already there
87930. logger_ must be above backup_engine_ such that the engine's destructor, which uses a raw pointer to the logger, executes first.
87931. set share_files_with_checksum to false
87932. all the dbs! BackupableDB owns dummy_db_
87933. ----- case 4. - invalid backup -----
87934. destroy_old_data
87935. checksum of the backup 2 appears to be valid, this can cause checksum mismatch and abort restore process
87936. every iteration -- 1. insert new data in the DB 2. backup the DB 3. destroy the db 4. restore the db, check everything is still there
87937. should fail creation
87938. 4 is corrupted, 3 is the latest backup now
87939. in last iteration, put smaller amount of data,
87940. latest backup should have all the keys
87941. Interrupt backup creation by failing new writes and failing cleanup of the partial state. Then verify a subsequent backup can still succeed.
87942. we deleted backup 2
87943. DummyDB
87944. delete backup number 2, online delete!

87945. create five backups
87946. single digit value, safe to insert one more digit
87947. check backup 5
87948. DummyLogFile
87949. Verify that you can backup and restore using share_files_with_checksum set to false and then transition this option to true
87950. close and destroy
87951. anon namespace
87952. Must reset() before reset(OpenDB()) again. Calling OpenDB() while *db_ is existing will cause LOCK issue
87953. first iter -- flush before backup second iter -- don't flush before backup
87954. ----- case 1. - fail a write ----- try creating backup 6, but fail a write
87955. ----- case 3. corrupted checksum value ----
87956. restores backup backup_id and asserts the existence of [start_exist, end_exist] and not-existence of [end_exist, end> if backup_id == 0, it means restore from latest if end == 0, don't check AssertEmpty
87957. delete the corrupt backups and then make sure they're actually deleted
87958. should fail
87959. ---- insert new data and back up ----
87960. check backup 3
87961. BackupableDBTest
87962. this seqnum guarantees the dummy file will be included in the backup as long as it is alive.
87963. the new backup fails because new table files clash with old table files from backups 4 and 5 (since write_buffer_size is huge, we can be sure that each backup will generate only one sst file and that a file generated by a new backup is the same as sst file generated by backup 4)
87964. see <https://github.com/facebook/rocksdb/issues/921>
87965. envs
87966. open DB, write, close DB, backup, restore, repeat
87967. all data should be there if we call with keep_log_files = true
87968. assert that data from backup 5 is still here (even though LATEST_BACKUP says 4 is latest)
87969. . and ..
87970. ---- restore every backup and verify all the data is there ----
87971. in last iteration, put smaller amount of data, so that backups can share sst files
87972. share_with_checksums
87973. FileManager
87974. new directory failure
87975. dummy
87976. **comment:** should also fail cleanup so the tmp directory stays behind
 label: code-design
87977. Behavior change: We now ignore LATEST_BACKUP contents. This means that we should have 5 backups, even if LATEST_BACKUP says 4.
87978. basically infinite
87979. since 5 meta is now corrupted, latest backup should be 4
87980. file 000007.sst was overwritten
87981. always rollover manifest for file add
87982. set up files
87983. write some data, backup, repeat
87984. get children failure
87985. ---- make sure it's empty ----
87986. set up envs
87987. iter 0 -- single threaded iter 1 -- multi threaded
87988. Some test cases do not actually create the test files (e.g., see DummyDB::live_files_) - for those cases, we mock those files' attributes so CreateNewBackup() can get their attributes.
87989. This test verifies we don't delete the latest backup when read-only option is set
87990. ----- case 2. - delete a file -----
87991. ---- make sure the data is there ---
87992. destroy old data
87993. reset all the defaults
87994. options
87995. this depends on the fact that 00007.sst is the first file created by the DB
87996. set up db options
87997. most tests will use multi-threaded backups
87998. restore 3
87999. Need to call this explicitly to delete tmp files
88000. created dir failure
88001. delete some backups -- this should leave only backups 3 and 5 alive
88002. context
88003. Filter expired blob indexes regardless of snapshots.
88004. new_value
88005. skip_until
88006. level
88007. namespace blob_db namespace rocksdb ROCKSDB_LITE
88008. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88009. Expired
88010. Unable to decode blob index. Keeping the value.
88011. CompactionFilter to delete expired blob index from base DB.
88012. namespace blob_db namespace rocksdb !ROCKSDB_LITE
88013. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88014. Iterator::Refresh() not supported.
88015. Test for data in SST
88016. Take a snapshot before compaction. Make sure expired blob indexes is filtered regardless of snapshot.
88017. First file
88018. The file shouln't be deleted
88019. Test for data in memtable
88020. The call simply pass through to base db. It should succeed.
88021. value_changed
88022. The default ttl extractor return no ttl for every key.
88023. Write to plain rocksdb.
88024. Issue manual compaction to trigger compaction filter.
88025. Second file
88026. Putting another blob should fail as ading it would exceed the blob_dir_size limit.

88027. namespace blob_db namespace rocksdb
88028. i = when to take snapshot
88029. The oldest simple blob file (i.e. blob_files[1]) has been selected for GC.
88030. The snapshot shouldn't see data in bfile
88031. Third file
88032. wait
88033. Test to verify that a NoSpace IOError Status is returned on reaching blob_dir_size limit.
88034. now
88035. The snapshot will see data in bfile, so the file shouldn't be deleted
88036. Fake blob index with TTL. See what it will do.
88037. Open as blob db. Verify it can read existing data.
88038. !ROCKSDB_LITE
88039. new_value
88040. very far in the future..
88041. DB should be empty.
88042. class BlobDBTest
88043. Verify blob db contain expected data and nothing more.
88044. The file is deleted this time
88045. Adding another 100 byte blob would take the total size to 264 bytes (2*132), which is more than 90% of blob_dir_size. So, the oldest file should be evicted and put in obsolete files list.
88046. Verify iterators
88047. VerifyDB use iterator to scan the DB.
88048. This test is no longer valid since we now return an error when we go over the configured blob_dir_size. The test needs to be re-written later in such a way that writes continue after a GC happens.
88049. context
88050. override all the keys
88051. Use mock env to stop wall clock.
88052. **comment:** Each stored blob has an overhead of about 42 bytes currently. So a small key + a 100 byte blob should take up ~150 bytes in the db.
 label: code-design
88053. run the same test for Get(), MultiGet() and Iterator each.
88054. **comment:** Each stored blob has an overhead of 32 bytes currently. So a 100 byte blob should take up 132 bytes.
 label: code-design
88055. Verify expired blob index are filtered.
88056. Verify normal Get
88057. Verify plain db return error for keys written by blob db.
88058. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88059. A black-box test for the t1l wrapper around rocksdb
88060. If true, the keys in this file all has TTL. Otherwise all keys don't have TTL.
88061. Mark file as obsolete by garbage collection. The file is not visible to snapshots with sequence greater or equal to the given sequence.
88062. The last sequence number by the time the file marked as obsolete. Data in this file is visible to a snapshot taken before the sequence.
88063. namespace blob_db namespace rocksdb ROCKSDB_LITE
88064. Compression type of blobs in the file
88065. has a pass of garbage collection successfully finished on this file obsolete_ still needs to do iterator/snapshot checks
88066. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88067. BlobIndex is a pointer to the blob and metadata of the blob. The index is stored in base DB as ValueType::kTypeBlobIndex. There are three types of blob index:
 kInlinedTTL: +-----+-----+-----+ | type | expiration | value | +-----+-----+-----+ | char | varint64 | variable size | +-----+-----+-----+
 kBlob: +-----+-----+-----+-----+ | type | file number | offset | size | compression | +-----+-----+-----+-----+
 +-----+ | char | varint64 | varint64 | varint64 | char | +-----+-----+-----+-----+-----+ | kBlobTTL: +-----+-----+-----+-----+-----+-----+-----+-----+ | type | expiration | file number | offset | size | compression | +-----+-----+-----+-----+-----+-----+-----+ | char | varint64 | varint64 | varint64 | char | +-----+-----+-----+-----+-----+-----+-----+
 There isn't a kInlined (without TTL) type since we can store it as a plain value (i.e. ValueType::kTypeValue).
88068. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88069. namespace blob_db namespace rocksdb ROCKSDB_LITE
88070. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88071. Verify the deserialization.
88072. Column1: Column
88073. Column2: Tombstone
88074. Verify the serialization.
88075. namespace cassandra namespace rocksdb
88076. expired not expired
88077. Verify the ColumnBase::Deserialization.
88078. Column0: ExpiringColumn
88079. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88080. Start each test with a fresh DB
88081. expired
88082. namespace cassandra namespace rocksdb
88083. The class for unit-testing
88084. expired not expired
88085. Path to the database on file system
88086. THE TEST CASES BEGIN HERE
88087. If the tombstone's timestamp is the latest, then it returns a row tombstone.
88088. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88089. namespace cassandra namespace rocksdb
88090. Some of the column's row is smaller, some is larger.
88091. All of the column's rows are larger than tombstone.
88092. This row's timestamp is smaller than tombstone.
88093. A row tombstone.
88094. namespace cassandra namespace rocksdb
88095. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88096. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

88097. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

88098. Return the current option configuration.

88099. Destroy using last options

88100. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

88101. Sequence of option configurations to try

88102. Roll the manifest during checkpointing right after live files are snapshotted.

88103. Writing prepare into middle of first WAL, then flush WALs many times

88104. Restore DB name

88105. namespace rocksdb

88106. Flush should never trigger.

88107. Open snapshot and verify contents

88108. always rollover manifest for file add

88109. Create a database

88110. Open snapshot and verify contents while DB is running

88111. allow_2pc is implicitly set with tx prepare options.allow_2pc = true;

88112. Destroy it for not alternative WAL dir is used.

88113. !ROCKSDB_LITE

88114. One flush after prepare + one outstanding file before checkpoint + one log file generated after checkpoint.

88115. Syncpoint prevents us building and running tests in release

88116. Insert something into CFB so lots of log files will be kept before creating the checkpoint.

88117. Destroy original DB

88118. Successful Open() implies that CURRENT pointed to the manifest in the checkpoint.

88119. Take a snapshot

88120. Get past the flush in the checkpoint thread before adding any keys to the db so the checkpoint thread won't hit the WriteManifest syncpoints.

88121. No more than two logs files should exist.

88122. Check correctness of decoded data

88123. Check correctness of encoded string length

88124. Check encoded string length: first value is original one (val - 0), the coming three are encoded as 1, -1, 1, so they should take 1 byte in varint.

88125. Check correctness of decoded value

88126. Check encoded string length

88127. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

88128. Check correctness of encoded value

88129. ROCKSDB_LITE

88130. namespace rocksdb

88131. Check correctness of decoded string length

88132. Database open and close should not affect

88133. Multiple column families for time series data

88134. Create key value pairs to insert

88135. check all insertions done

88136. Another time series column family is created

88137. Resets the timestamp of a set of kvs by updating them and checks that they are not deleted according to the old timestamp

88138. T=2, keys put into database are already obsolete Put data in database. Operations should not return OK

88139. Put data in database

88140. Explicitly drop obsolete column families

88141. Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

88142. T=2, keys should not be retrieved

88143. Iterator should be able to merge data from different column families

88144. T=0, open the database and insert data

88145. The first column family is deleted from database

88146. Populates and returns a kv-map

88147. T=5, keys should not be retrieved

88148. !ROCKSDB_LITE

88149. Empty iterator

88150. Put more data

88151. **comment:** A black-box test for the DateTieredDB around rocksdb
label: test

88152. namespace rocksdb

88153. Data should not be found in database

88154. Puts a set of values and checks its presence using Get during ttl

88155. Key should not be found

88156. Only default column family

88157. Puts a set of values and checks its presence using iterator during ttl

88158. Iterator should be able to merge data from multiple column families

88159. **comment:** Delete keys when they are not obsolete
label: code-design

88160. A time series column family is created

88161. **comment:** If the key in Put() is obsolete, the data should not be written into database
label: code-design

88162. T=1, keys should still reside in database

88163. inserting a document with existing primary key should return failure

88164. Delete all whose progress is bigger than 50%

88165. job_name == 'white' AND priority >= 2, index job_name

88166. 2 < priority <= 4, index priority

88167. 35.0 <= progress < 65.5, index progress

88168. add index on the fly!

88169. 2 < priority < 4 AND progress > 10.0, index priority

88170. 2 < priority < 6, index priority

88171. -2 < priority < 0

88172. 4 < priority

88173. 2 < priority < 4 AND progress > 10.0, index progress

88174. find all -- only "Two" left, everything else should be deleted

88175. update set priority to 10 where job_name is 'white'

88176. now there is index present

88177. !ROCKSDB_LITE
88178. find equal to "Two"
88179. priority < 0
88180. remove less or equal to "Three"
88181. namespace rocksdb
88182. find less than "Three"
88183. converts ' to ", so that we don't have to escape " all over the place
88184. -1 <= priority <= 1, index priority
88185. -2 <= priority < 0
88186. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88187. update twice: set priority to 15 where job_name is 'white'
88188. find less than "Three" without index
88189. update twice: set priority to 15 and progress to 40 where job_name is 'white'
88190. Copy constructor from owner -> owner
88191. Move constructor from owner -> owner
88192. test deep copying
88193. test iteration
88194. kNull
88195. !ROCKSDB_LITE
88196. namespace rocksdb
88197. kInt64
88198. Move constructor from non-owner -> non-owner
88199. Copy constructor from non-owner -> non-owner
88200. properties
88201. namespace
88202. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88203. deserialization failure
88204. kBool
88205. kString
88206. parse error
88207. kDouble
88208. Set a snapshot at start of transaction
88209. * next, we actually connect to the cluster
88210. open DB
88211. **comment:** Optimize Rocksdb. This is the easiest way to get RocksDB to perform well
 label: code-design
88212. Read "foo".
88213. **comment:** Optimize RocksDB. This is the easiest way to get RocksDB to perform well
 label: code-design
88214. Do some reads and writes to key "x"
88215. // "Read Committed" (Monotonic Atomic Views) Example --Using multiple Snapshots //////////////////////////////////
88216. Check that the file exists.
88217. Set a snapshot at start of transaction by setting set_snapshot=true
88218. Read "d".
88219. namespace rocksdb
88220. Read "hello".
88221. Check that the directory is empty.
88222. get value
88223. Attempt to read a key using the snapshot. This will fail since the previous write outside this txn conflicts with this read.
88224. Commit.
88225. Write to the file.
88226. Try to skip past end of file.
88227. Check that opening non-existent file fails.
88228. // Simple OptimisticTransaction Example ("Read Committed") //////////////////////////////////
88229. Random reads.
88230. Read a key OUTSIDE this transaction. Does not affect txn.
88231. Check that renaming works.
88232. Start a transaction
88233. Commit transaction
88234. **comment:** * And now we're done, so let's remove our pool and then * shut down the connection gracefully.
 label: code-design
88235. create the DB if it's not already present
88236. Read "world".
88237. Too high offset.
88238. ***** use default env *****
88239. Copyright (c) 2016, Red Hat, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88240. just use the client.admin keyring let's handle any error that might have come back
88241. ***** use librados env *****
88242. // "Repeatable Read" (Snapshot Isolation) Example -- Using a single Snapshot //////////////////////////////////
88243. Do some reads and writes to key "y" Since the snapshot was advanced, the write done outside of the transaction does not conflict.
88244. !ROCKSDB_LITE
88245. Check that deleting works.
88246. Read sequentially.
88247. atomically apply a set of updates
88248. These are no-ops, but we test they return success.
88249. Check for expected size.
88250. Do a write outside of the transaction to key "y"
88251. Set a new snapshot in the transaction
88252. Write a key OUTSIDE of transaction
88253. we will use all of these below
88254. Create a file.
88255. Write a key OUTSIDE of this transaction. Does not affect txn since this is an unrelated key. If we wrote key 'abc' here, the transaction would fail to commit.
88256. be careful not to
88257. Read a key in this transaction
88258. In this example, we set the snapshot multiple times. This is probably only necessary if you have very strict isolation requirements to implement.

88259. Decide we want to revert the last write from this transaction.
88260. Put key-value
88261. **comment:** Cleanup
 label: code-design
88262. Try reading past EOF.
88263. This could fail if the config file is malformed, but it'd be hard.
88264. Clear snapshot from read options since it is no longer valid
88265. Write a key in this transaction
88266. Copyright (c) 2015, Red Hat, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88267. Read "foo".
88268. namespace rocksdb
88269. Check that the file exists.
88270. Read "d".
88271. Read "hello".
88272. Check that the directory is empty.
88273. Write to the file.
88274. Try to skip past end of file.
88275. Check that opening non-existent file fails.
88276. Random reads.
88277. Check that renaming works.
88278. Read "world".
88279. Too high offset.
88280. !ROCKSDB_LITE
88281. Check that deleting works.
88282. Read sequentially.
88283. These are no-ops, but we test they return success.
88284. Check for expected size.
88285. Create a file.
88286. Try reading past EOF.
88287. namespace rocksdb
88288. Copyright (c) 2017-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88289. ROCKSDB_LITE
88290. search all objects centered at 46 degree latitude with a radius of 2 kilometers. There should be none.
88291. search all objects centered at 46 degree latitude with a radius of 200 kilometers. We should find the one object that we inserted earlier.
88292. insert first object into database
88293. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88294. Search. Verify distances via <http://www.stevemorse.org/nearest/distance.php>
88295. check that we can still find second object
88296. insert object at 45 degree latitude
88297. Insert, Get and Remove
88298. retrieve first object using position
88299. retrieve first object using id
88300. namespace rocksdb
88301. insert second object into database
88302. delete first object
88303. !ROCKSDB_LITE
88304. Performs the following logic: If key[0] < 'h' --> replace value by reverse key If key[0] >= 'r' --> keep the original key value Otherwise, filter the key value
88305. keeps all the key value pairs
88306. removes all keys whose initial is less than 'r'
88307. Issue full compaction and expect nothing is in the DB.
88308. If nothing is set in the LuaCompactionFilterOptions, then RocksDB will keep all the key / value pairs, but it will also print our error log indicating failure.
88309. removes all the key value pairs
88310. Issue full compaction and expect everything is in the DB.
88311. Copyright (c) 2016, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88312. Create DB with 10 files
88313. Filter() error when input key's initial >= 'r'
88314. defined(LUA)
88315. !defined(ROCKSDB_LITE)
88316. removes all values whose initial is less than 'r'
88317. namespace rocksdb
88318. Replace all values by their reversed key
88319. change the lua script to removes all the key value pairs
88320. Issue full compaction and expect all keys which initial is < 'r' will be deleted as we keep the key value when we hit an error.
88321. namespace rocksdb
88322. **comment:** Fill one memtable per Put to make memtable use more memory.
 label: code-design
88323. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88324. Cache from table factories
88325. Expect the usage history of the table readers increases as we flush tables.
88326. Create an iterator and flush all memtables for each db
88327. Expect the usage history from the "usage_decay_point" is monotonically decreasing.
88328. Cache from DBImpl
88329. Cache from DBOptions
88330. Expect EQ as we didn't flush more memtables.
88331. Expect the usage of all memtables decreasing as we delete iterators.
88332. Expect the usage history is monotonically increasing
88333. Since the memory usage of un-flushed memtables is only affected by Put and flush, we expect EQ here as we only delete iterators.
88334. Since memtables are pinned by iterators, we don't expect the memory usage of all the memtables decreases as they are pinned by iterators.
88335. !ROCKSDB_LITE
88336. Start each test with a fresh DB
88337. we should have a place to store the result
88338. implicit
88339. Perform the following operations in limited scope

88340. Get (Quick check for persistence of previous database)
88341. Apply the similar "Append" to the parallel copy
88342. start from k2 this time.
88343. Append, Flush, Append, Get
88344. !ROCKSDB_LITE namespace
88345. Allows user to open databases with different configurations. e.g.: Can open a DB or a TtlDB, etc.
88346. namespace rocksdb
88347. Path to the database on file system
88348. Should release the snapshot and be aware of the new stuff now
88349. deterministic seed
88350. THE TEST CASES BEGIN HERE
88351. Constructor: specifies the rocksdb db
88352. Generate a random query (Append or Get) and random parameters
88353. Apply the query and any checks.
88354. All changes should be on disk. This will test VersionSet Get()
88355. NotFound is okay; just return empty (similar to std::map) But network or db errors, etc, should fail the test (or at least yell)
88356. Assumes that a non-existent key just returns <empty>
88357. Append, Get
88358. OpenDb opens a (possibly new) rocksdb database with a StringAppendOperator
88359. /StringLists represents a set of string-lists, each with a key-index. / Supports Append(list, string) and Get(list)
88360. Construct the desired string. Default constructor doesn't like '\0' chars. Verify that the string is right size. Use null delimiter instead of comma.
88361. Will store a local copy of all data in order to verify correctness
88362. Either key does not exist, or there is some error. Always return empty string (just for convention)
88363. !ROCKSDB_LITE
88364. Append string val onto the list defined by key; return true on success
88365. Snapshot should still be the same. Should ignore a4 and v4.
88366. The class for unit-testing
88367. Generate semi random keys/words from a small distribution.
88368. TtlDb is not supported in Lite Open a TtlDB with a non-associative StringAppendTESTOperator
88369. I omit the "assert(success)" checks here.
88370. should not be hit
88371. Append, Compact, Get
88372. TtlDb is not supported in Lite Run with TTL
88373. Append, Flush, Get
88374. deterministic seed; always get same results!
88375. Append, Flush, Compact, Get
88376. Always return false if s.ok() was not true
88377. Apply the rocksdb test-harness Append defined above apply the rocksdb append
88378. Returns the list of strings associated with key (or "" if does not exist)
88379. * * An persistent map : key -> (list of strings), using rocksdb merge. * This file is a test-harness / use-case for the StringAppendOperator. * * @author Deon Nicholas (dnicholas@fb.com) * Copyright 2013 Facebook, Inc.
88380. Verify it is still the correct size
88381. Generate a bunch of random queries (Append and Get)!
88382. The previous changes should be on disk (L0) The most recent changes should be in memory (MemTable) Hence, this will test both Get() paths.
88383. Check that the rocksdb result string matches the desired string
88384. Run with regular database
88385. Reopen the database (the previous changes should persist / be remembered)
88386. Apply the query and any checks.
88387. All three keys should have been found
88388. Generate a list of random keys and values
88389. Compact, Get
88390. namespace rocksdb
88391. ROCKSDB_LITE
88392. Env::Default() is a singleton so we can't grant ownership directly to the caller - we must wrap it first.
88393. Copyright (c) 2016-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88394. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
88395. Make level target of L1, L2 to be 200KB and 600KB
88396. Required if inheriting from testing::WithParamInterface<>
88397. Wait for compaction to finish and make sure it can reopen
88398. Will make sure exactly those keys are in the DB after migration.
88399. ROCKSDB_LITE namespace rocksdb
88400. Generate at least 2MB of data
88401. Issue a full compaction to generate some zero-out files
88402. GFLAGS
88403. It's okay to change prefix_extractor from nullptr to non-nullptr
88404. prefix extractor
88405. close the db
88406. It's okay to set prefix_extractor to nullptr.
88407. namespace rocksdb
88408. perform sanity check
88409. prefix extractor nullptr case
88410. determine whether this is a valid src upon the function applies
88411. determine whether dst=Transform(src) for some src
88412. transform a src in domain to a dst in the range
88413. namespace
88414. !ROCKSDB_LITE
88415. Assign non-null values to prefix_extractors except the first cf.
88416. comparator
88417. table factory
88418. Return the name of this transformation.
88419. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88420. merge operator
88421. open and persist the options
88422. erase a few keys randomly

88423. Copyright (c) 2013, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).

88424. insert

88425. verify

88426. ret=

88427. namespace rocksdb

88428. unfortunately we can't predict eviction value since it is from any one of the lock stripe

88429. small write buffer

88430. page cache, block cache, compressed cache + KNoCompression both block cache and compressed cache, but DB is not compressed also, make block cache sizes bigger, to trigger block cache hits

88431. cleanup files with the patter :digi:.rc

88432. construct a tiered RAM+Block cache

88433. max_usecase=

88434. Copyright (c) 2013, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

88435. compressed=

88436. namespace rocksdb

88437. direct_writes=

88438. Simple logger that prints message on stdout

88439. remember kNoCompression

88440. nthreads=

88441. number of insertion interations

88442. create primary tier

88443. is_compressed

88444. page cache, no block cache, no compressed cache

88445. page cache, block cache, compressed cache + KNoCompression

88446. max_keys=

88447. max_size=

88448. Travis is unable to handle the normal version of the tests running out of fds, out of space and timeouts. This is an easier version of the test specifically written for Travis

88449. we assume the directory does not exist

88450. block_cache_size

88451. page cache, no block cache, no compressed cache Page cache caches compressed blocks

88452. check that we triggered the appropriate code paths in the cache

88453. create block cache

88454. test table with block page cache

88455. setup page cache

88456. insert data

88457. flush all data in cache to device

88458. max_keys

88459. Volatile cache tests

88460. page cache, block cache, no-compressed cache

88461. test template

88462. test table with volatile page cache

88463. page cache, block cache, compressed cache

88464. test table with tiered page cache

88465. size=

88466. create secondary tier

88467. create a new cache tier

88468. Block cache tests

88469. dbname

88470. verify data

88471. create tier out of the two caches

88472. Tiered cache tests

88473. memory_size=

88474. the tests causes a lot of file deletions which Travis limited testing environment cannot handle

88475. is_compressed=

88476. Wait for threads to join

88477. template for negative insert test

88478. assume that the key is evicted

88479. eviction_enabled=

88480. flush all data from memtable so that reads are from block cache

88481. Copyright (c) 2013, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.

88482. insert data to table

88483. pad 0 to numbers

88484. Unit tests for testing PersistentCacheTier

88485. template for insert with eviction test

88486. namespace rocksdb

88487. copy numbers

88488. Flush cache

88489. high compression ratio

88490. count=

88491. Run insert workload in threads

88492. post condition

88493. template for insert test

88494. Write 8MB (80 values, each 100K)

88495. create threaded workload

88496. default column family doesn't have block cache

88497. spawn threads

88498. test template

88499. Verification implementation

88500. copy 0s

88501. Run verification on the cache

88502. Insert workload implementation

88503. join with threads

88504. RocksDB tests
88505. verify data
88506. Size should not change
88507. Check Get the list
88508. Repopulate some stuff Results in: [x, x, x, x, newbegin, z, x, aftera, x, newend, a, x]
88509. Added by Deon
88510. Exhaustive test of the Range() function
88511. Partial / Broken indices
88512. Block one: populate a single key in the database
88513. Test Trim [z, a, aftera, x] (do nothing)
88514. PushRight, Length, Index, Range
88515. Persistent, non-destructive
88516. We have: [sad, nota, zz, happy, z, beforeba2, ba2, a, aftera, x, newend]
88517. Simple Pushes (will yield: [v6, v4, v4, v1, v2, v3])
88518. Test InsertAfter end
88519. Testing Remove, RemoveFirst, RemoveLast
88520. Empty The length should not change
88521. Test out of bounds (empty) trim
88522. Check valid indices
88523. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88524. Set() with negative indices
88525. Exhaustive test for InsertBefore(), and InsertAfter()
88526. Block two: make sure changes were saved and add some other key
88527. Test InsertAfter
88528. [sad, a, happy, z, a, aftera, ...]
88529. Test bad value on InsertAfter
88530. Test removal from end
88531. Check
88532. operator== and operator<< are defined below for vectors (lists) Needed for ASSERT_EQ
88533. Bad indices (just out-of-bounds / off-by-one check)
88534. InsertBefore start of list InsertAfter end of list
88535. Used below for all Index(), PopRight(), PopLeft()
88536. Test Inserting before/after non-existent values Ensure that the length doesn't change
88537. namespace
88538. Test Regular Set
88539. [z, a, aftera]
88540. A series of pushes and insertions Will result in [newbegin, z, a, aftera, x, newend, a, a]
88541. Testing Insert, Push, and Set, in a mixed environment
88542. **comment:** / THE manual REDIS TEST begins here / THIS WILL ONLY OCCUR IF YOU RUN: ./redis_test -m
 label: test
88543. Make sure nothing weird happened.
88544. Testing duplicate values/pivots (multiple occurrences of 'a') [a, z, a, aftera, x, newend] [a, happy, z, a, aftera, ...]
88545. Check Length and Index() functions Check length
88546. Set on empty list (return false, and do not crash)
88547. Check range function and vectors Get the list
88548. **comment:** Test over-shooting (removing more than there exists)
 label: code-design
88549. **comment:** USAGE: "./redis_test" for default (unit tests) "./redis_test -m" for manual testing (redis command api) "./redis_test -m -d" for destructive manual test (erase db before use)
 label: test
88550. Simple PushRight
88551. Testing Trim, Pop
88552. Simple Pushes (will yield: [v6, v4, v4, v1, v2, v3])
88553. Removal from an empty-list
88554. namespace rocksdb
88555. Simple, non-negative indices
88556. Try removing ALL! REMOVE 0 will remove all!
88557. Check RemoveFirst (Remove the first two 'a') Results in [newbegin, z, aftera, x, newend, a]
88558. Test InsertBefore
88559. Exhaustive test of Set function
88560. Test out of bounds Set
88561. One last check (to make sure nothing weird happened) Size should not change
88562. Check for "want" argument in the argument list
88563. Invalid indices
88564. Empty Index check (return empty and should not crash or edit tempv)
88565. Exhaustive Trim test (negative and invalid indices) Will start in [newbegin, z, a, aftera, x, newend]
88566. !ROCKSDB_LITE
88567. We now have: [x, x, x, x, newbegin, z, aftera, newend, a]
88568. Should do nothing
88569. Destructive
88570. Only really removed 4
88571. Popping with empty list (return empty without error)
88572. Negative indices (i.e.: measured from the end)
88573. **comment:** Will run unit tests. However, if -m is specified, it will do user manual/interactive testing -m -d is manual and destructive (will clear the database before use)
 label: test
88574. Push some preliminary stuff [g, f, e, d, c, b, a]
88575. Range matches Index
88576. Simple PopLeft/Right test
88577. Insert on empty list (return 0, and do not crash)
88578. A series of pushes and insertions Will result in [newbegin, z, a, aftera, x, newend] Also, check the return value sometimes (should return length)
88579. / Allows the user to enter in REDIS commands into the command-line. / This is useful for manual / interactive testing / debugging. / Use destructive=true to clean the database before use. / Use destructive=false to remember the previous state (i.e.: persistent) / Should be called from main function.
88580. InsertBefore beginning of list InsertAfter end of list
88581. [sad, nota, zz, happy, z, a, ...] [sad, nota, zz, happy, z, ba, a, ...]
88582. Exhaustive test of the Index() function
88583. Sanity check (check the length; make sure it's 6)

88584. Last check
88585. [sad, a, zz, happy, z, a, aftera, ...]
88586. Set with negative indices
88587. Block three: Verify the changes from block 2
88588. Ensure Set worked correctly
88589. Test Multiple keys and Persistence
88590. A series of pushes and insertions Will result in [newbegin, z, a, aftera, x, newend]
88591. Out of bounds (return empty, no crash)
88592. Simply Test the Set() function
88593. PushLeft, Length, Index, Range
88594. Simple PushRight (should return the new length each time)
88595. Test bad value on InsertBefore
88596. Negative indices
88597. Now have: [z, a, aftera, x]
88598. ** A test harness for the Redis API built on rocksdb. ** USAGE: Build with: "make redis_test" (in rocksdb directory). * Run unit tests with: "./redis_test" * Manual/Interactive user testing: "./redis_test -m" * Manual user testing + restart database: "./redis_test -m -d" ** TODO: Add LARGE random test cases to verify efficiency and scalability ** @author Deon Nicholas (dnicholas@fb.com)
88599. [a]
88600. Test InsertBefore beginning
88601. C-version defined in <ctype.h>
88602. We currently have: [sad, nota, zz, happy, z, ba, a, aftera, x, newend] redis.Print("k1"); // manually check
88603. Verify
88604. Simple range
88605. **comment:** TODO: Right now, please use spaces to separate each word. In actual redis, you can use quotes to specify compound values Example: RPUSH mylist "this is a compound value"
label: code-design
88606. error margin of 100 bytes
88607. We asked for every block twice
88608. namespace rocksdb
88609. count number of lookups
88610. count number of additions
88611. Log things again but stop logging automatically after reaching 512 bytes
88612. Release iterators and access cache again.
88613. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88614. Test with strict capacity limit.
88615. Set a small enough block size so that each key-value get its own block.
88616. options.compression = kNoCompression;
88617. Load blocks into cache.
88618. We added every block only once, since the cache can hold all blocks
88619. Add kNumBlocks again
88620. iter 0 -- not read only iter 1 -- read only
88621. namespace spatial namespace rocksdb
88622. null
88623. corrupted serialization
88624. serial
88625. parallel
88626. even though the bounding box doesn't intersect, we got "four" back because it's in the same tile
88627. namespace
88628. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88629. !ROCKSDB_LITE
88630. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
88631. TEST 3: Issues a lots of deletes, but their density is not high enough to trigger compaction.
88632. Only one section of a file satisfies the compaction trigger
88633. randomize tests
88634. deterministic tests
88635. Simple test
88636. Insert "kNumDeletionTrigger * 0.95" deletions for every "kWindowSize" and verify compaction is not needed.
88637. !ROCKSDB_LITE
88638. Txn should commit
88639. Rollback to beginning of txn
88640. This Put outside of a transaction will conflict with the previous read
88641. txn1 can commit since txn2's delete hasn't happened yet (it's just batched)
88642. have to open default column family
88643. This Put outside of a transaction will conflict with the previous write
88644. Create N threads that use RandomTransactionInserter to write many transactions.
88645. This should not conflict in txn since the snapshot is later than the previous write (spoiler alert: it will later conflict with txn2).
88646. txn2 cannot commit since txn1 changed "4"
88647. txn should commit since the flushed table is still in MemtableList History
88648. Put a key which will conflict with the next txn using the previous snapshot
88649. Write some data to the db
88650. Put a random key so we have a MemTable to flush
88651. Set a new snapshot
88652. namespace rocksdb
88653. Put a random key so we have a memtable to flush
88654. Modify BBB before snapshot is taken
88655. 1
88656. Wait for all threads to run
88657. key "C" was modified in the db after txn's snapshot. txn will not commit.
88658. Verify transaction rollback works for untracked keys.
88659. Should commit since read/write was done after data changed
88660. force a memtable flush
88661. open the new column families
88662. Write to the untracked key outside of the transaction and verify it doesn't prevent the transaction from committing.
88663. Txn should not commit

88664. Conflicts with write done after snapshot taken
88665. verify that we track multiple writes to the same key at different snapshots
88666. Modify key after transaction start
88667. Verify txn did not commit
88668. Write some keys in a txn
88669. 2
88670. This will conflict since the snapshot is earlier than another write to ZZZ
88671. Rollback to 3
88672. Verify that txn1 cannot commit since A will still be conflict checked
88673. namespace
88674. Verify that data is consistent
88675. !ROCKSDB_LITE
88676. Write some keys to the db
88677. These keys do no conflict with existing writes since they're in different column families
88678. Write some keys to the db after the snapshot
88679. unexpected failure
88680. open DB with three column families
88681. Write to tracked key outside of the transaction and verify that the untracked keys are not written when the commit fails.
88682. Test 2 transactions writing to the same key in multiple orders and with/without snapshots
88683. Potentially conflicting writes
88684. Verify Txn Did not Commit
88685. Make sure at least some of the transactions succeeded. It's ok if some failed due to write-conflicts.
88686. Read and write with snapshot
88687. 3 4
88688. should not commit since txn2 wrote a key txn has read
88689. This Put outside of a transaction will conflict with a later write
88690. Create 2 new column families
88691. Read all keys via iter and lock them all
88692. Verify that transaction did not write anything
88693. txn should not commit since MemTableList History is not large enough
88694. This write will cause a conflict with the earlier batch write
88695. Test to make sure transactions work when there are no other writes in an empty db.
88696. force a memtable flush Since our test db has max_write_buffer_number=2, this flush will cause the first memtable to get purged from the MemtableList history.
88697. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88698. Setting the key-space to be 100 keys should cause enough write-conflicts to make this test interesting.
88699. Read and write without a snapshot
88700. Rollback to 2
88701. Verify that txn1 can commit since A isn't conflict checked
88702. try rollback again
88703. Txn should commit
88704. This Put outside of a transaction will conflict with the previous read
88705. should show the first txn log
88706. verify foo is locked by txn
88707. Verify that only "A" is locked
88708. Snapshot is now set
88709. This should not conflict in txn since the snapshot is later than the previous write (spoiler alert: it will later conflict with txn2).
88710. wait for friends
88711. txn2 has a smaller lock timeout than txn1's expiration, so it will time out
88712. should succeed since key was written before txn started
88713. assure that all thread are in the same write group
88714. Put a key which will conflict with the next txn using the previous snapshot
88715. Write some data to the db
88716. Put a random key so we have a MemTable to flush
88717. Set a new snapshot
88718. Verify A,B,D,E,F are still locked and C is not.
88719. verify that txn has "A" locked
88720. flush only cfb memtable
88721. Locked keys exist in both column family.
88722. Reinitialize txn1 and verify that Z gets unlocked
88723. **comment:** TODO this test needs to be updated with serial commits
 label: test
88724. Create a txn and verify we can only lock up to 3 keys
88725. Verify that "A" and "C" is still locked while "B" is not
88726. Upsize the buffer and verify the 3 latest dealocks are preserved.
88727. fails due to non-empty commit-time batch
88728. valid name set
88729. sleep override
88730. Should commit since read/write was done after data changed
88731. mix transaction writes and regular writes
88732. force a memtable flush
88733. Rollback the leaf transaction.
88734. Test txn1 and txn2 sharing a lock and txn2 trying to upgrade lock.
88735. Verify B is locked and A and C are not
88736. Snapshot was created before change to '2'
88737. heap should not show any logs
88738. Verify "A" "C" "B" are no longer locked
88739. Should fail since column family was dropped.
88740. Test txn1 trying to downgrade its lock.
88741. after memtable flush we can now release the log
88742. verify foo2 is locked by txn
88743. valid set name
88744. The iterator points to an unordered_multimap thus the test can not assume any particular order.
88745. commit time put
88746. Calculate next buffer len, plateau at 5 when 5 records are inserted.
88747. Write some keys to the db after the snapshot
88748. **comment:** TODO(myabandeh): Instantiate the tests with other write policies
 label: code-design

88749. In this test, txn1 should succeed committing, as the callback is called after txn1 starts committing.
88750. unexpected failure
88751. We want the last transaction in the chain to block and hold everyone back.
88752. Transaction should only write the keys that succeeded.
88753. txn2 should be expired out since txn1 waiting until its timeout expired.
88754. now we pause background work so that imm()s are not flushed before we can check their status
88755. ROCKSDB_LITE
88756. deleting transaction should unregister transaction
88757. Complete the cycle Tlen -> T1
88758. Header (12 bytes) + NOOP (1 byte) + 2 * 8 bytes for data.
88759. 10ms
88760. verify data txn data
88761. Should fail after encountering a write to Z in SST file
88762. This Put outside of a transaction will conflict with a later write
88763. Committing txn should release its locks and allow txn2 to proceed
88764. transaction writes have 10ms timeout, db writes have infinite timeout
88765. 100 minutes 1ms
88766. rollback to 1
88767. Txn should commit, but only write foo2 and foo3
88768. our log should be in the heap
88769. name too long
88770. The locked key is "foo" and is locked by txn1
88771. Downsize to 0 and verify the size is consistent.
88772. flush only cfa memtable
88773. commit in LOG B
88774. Verify A,B,D,E,F,H are still locked and C,G are not.
88775. According to db.h, doing a SingleDelete on a key that has been overwritten will have undefined behavior. So it is unclear what the result of fetching "A" should be. The current implementation will return NotFound in this case.
88776. log cannot be flushed because txn2 has not been committed
88777. force a memtable flush Since our test db has max_write_buffer_number=2, this flush will cause the first memtable to get purged from the MemtableList history.
88778. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
88779. else commit it dummy put to inc db seq
88780. Rollback to 2
88781. but now our memtable should be referencing the prep section
88782. conflict
88783. commit old txn
88784. 100 minutes
88785. txn2 should be able to write to X since txn1 has expired
88786. We want the leaf transactions to block and hold everyone back.
88787. have to open default column family
88788. assert that cfa has a flush requested
88789. Conflict
88790. Create N threads that use RandomTransactionInserter to write many transactions.
88791. Verify A,B,F are still locked and C,D,E,G,H are not.
88792. Cannot lock B since it was written after the snapshot was set
88793. put txn2 prep section in this log
88794. make commit
88795. dummy put to inc db seq
88796. Should still fail after "downgrading".
88797. cfb should not be flushed becuze it has no data from LOG A
88798. make sure we are on a new log
88799. 1
88800. can't rename after prepare
88801. Set up a long wait for chain like this: T1 -> T2 -> T3 -> ... -> Tlen
88802. Now change "B".
88803. "C" was modified after txn's snapshot
88804. 1 ms
88805. shouldn't be able to prepare
88806. Test shared access between txns
88807. Force txn1 to expire
88808. We increase the db seq artificailly by a dummy Put. Check that this technique is effective and db seq is that same as ours.
88809. namespace
88810. verify that we track multiple writes to the same key at different snapshots
88811. Safety check to make sure the test actually ran
88812. Write some keys in a txn
88813. Use small commit cache to trigger lots of eviction and fast advance of max_evicted_seq_ will take effect after ReOpen
88814. This will conflict since the snapshot is earlier than another write to ZZZ
88815. find trans in list of prepared transactions
88816. value is readable from txn
88817. do prepare for a transaction dummy put to inc db seq
88818. All results should fail since there was a conflict
88819. Write some keys to the db
88820. Test txn1 holding an exclusive lock and txn2 trying to obtain shared access.
88821. nothing has been prepped yet
88822. Number of snapshots taken so far
88823. Conflict's with txn1's MultiGetForUpdate
88824. Upsize from 0 to verify the size is persistent.
88825. should show not dependency on logs
88826. Test 2 transactions writing to the same key in multiple orders and with/without snapshots
88827. Snapshot was set before D was written
88828. regular db read
88829. transaction should no longer be visible
88830. 3 4
88831. value is readable form txn
88832. 3 Rollback to 3
88833. Read all keys via iter and lock them all
88834. close

88835. Verify that "A", "B", "C" are still locked
88836. prepare
88837. We keep the list of txns committed before we take the last snapshot. These should be the only seq numbers that will be found in the snapshot
88838. Verify the size of the buffer and the single path.
88839. Set txn expiration timeout to 0 microseconds (expires instantly)
88840. Set up a wait for chain like this: Tn -> T(n*2) Tn -> T(n*2 + 1) So we have: T1 -> T2 -> T4 ... | > T5 ... | > T3 -> T6 ... | > T7 ... up to T31, then T[16 - 31] -> T1.
Note that Tn holds lock on floor(n / 2).
88841. Wait until all threads are waiting on each other.
88842. to restart the db
88843. Lock mostly for shared access, but exclusive 1/4 of the time.
88844. Rollback the last transaction.
88845. Column family is 1 or 0 (cfa).
88846. Verify A,B,C are locked
88847. Conflicts with previous GetForUpdate
88848. No snapshot created yet
88849. Should fail since column family was dropped
88850. roll to LOG B
88851. no longer is pprpared results
88852. txn1 has a lock timeout longer than txn2's expiration, so it will win
88853. Setting the key-space to be 100 keys should cause enough write-conflicts to make this test interesting.
88854. Take the final snapshot if it is not already taken
88855. Verify F is still locked and A,B,C,D,E,G,H are not.
88856. Complete the cycle T2 -> T1 with a shared lock.
88857. Take preliminary snapshots
88858. Rollback to beginning of txn
88859. kill and reopen
88860. To stress the data structure that maintain prepared txns, at each cycle we add a new prepare txn. These do not mean to be committed for snapshot inspection.
88861. Since we do not actually write to db, we mock the seq as it would be increased by the db. The only exception is that we need db seq to advance for our snapshots.
for which we apply a dummy put each time we increase our mock of seq.
88862. transaction put in two column families
88863. Create a cycle of size 2.
88864. regular put so that mem table can actually be flushed for log rolling
88865. Snapshot was set before C was written
88866. We compare whether the set of txns locking this key is the same. To do this, we need to sort both vectors so that the comparison is done correctly.
88867. In the above the latest change to AAAZZZ in handles[1] is delete.
88868. put new txn values
88869. The original value should still be read
88870. value is now available
88871. Should fail after encountering a write to S in SST file
88872. At each step we prepare a txn and then we commit it in the next txn. This emulates the consecutive transactions that write to the same key
88873. Drop first column family
88874. transaction expired!
88875. rollback to 2
88876. Test conflict checking against the very first write to a db. The transaction's snapshot is a seq 0 while the following write will have sequence 1.
88877. Conflicts with previous GetForUpdate. Since db writes do not have a timeout, this should eventually succeed when the transaction expires.
88878. Wait for some gap before taking the final snapshot
88879. According to db.h, doing a SingleDelete on a key that has been overwritten will have undefined behavior. So it is unclear what the result of fetching "F" should be.
The current implementation will return NotFound in this case.
88880. If the snapshot is taken, verify seq numbers visible to it. We redo it at each cycle to test that the system is still sound when max_evicted_seq_ advances.
88881. Verify that transaction wrote foo2 and foo3 but not foo
88882. namespace rocksdb
88883. re-locking same key shouldn't put us over the limit
88884. Verify transaction rollback works for untracked keys.
88885. cant prepare txn without name
88886. Reinitialize transaction to no long expire
88887. cant reset name
88888. Snapshot should not have changed yet.
88889. Conflicts with write done after snapshot taken
88890. Contrived case of shared lock of cycle size 2 to verify that a shared lock causing a deadlock is correctly reported as "shared" in the buffer.
88891. heap should still see first log
88892. new value is readable from txn
88893. * 1) use prepare to keep first log around to determine starting sequence * during recovery. * 2) insert many values, skipping wal, to increase seqid. * 3) insert final value into wal * 4) recover and see that final value was properly recovered - not * hidden behind improperly summed sequence ids
88894. Test WritePreparedTxnDB's IsInSnapshot against different ordering of snapshot, max_committed_seq_, prepared, and commit entries.
88895. Any other operation does
88896. Test conflict checking against the very first write to a db. The transaction's snapshot will have seq 1 and the following write will have sequence 1.
88897. Offset of the final entry in the dlock path from the root's txn id.
88898. open DB with three column families
88899. commit txn1
88900. **comment:** offset by 2 from the max depth to test edge case
label: test
88901. txn2 should not be able to write to X since txn1 has it locked
88902. **comment:** value is nolonger readable
label: code-design
88903. request a flush for all column families such that the earliest alive log file can be killed
88904. log has been marked
88905. Potentially conflicting writes
88906. transaction put
88907. Downsize the buffer and verify the 3 latest deadlocks are preserved.
88908. Make sure at least some of the transactions succeeded. It's ok if some failed due to write-conflicts.
88909. Same for snapshot cache size
88910. txn2 should now be able to write to X
88911. 3
88912. 4
88913. A Get does not generate the snapshot
88914. all data in LOG A resides in a memtable that has been requested for a flush
88915. Verify A,B,D,E,F are still locked and C,G,H are not.
88916. Complete the cycle T[16 - 31] -> T1

88917. create second cf
88918. crash
88919. we already committed
88920. This write will cause a conflict with the earlier batch write
88921. Special case for a deadlock path that exceeds the maximum depth.
88922. do all the writes
88923. write prep section to wal
88924. lock limit reached
88925. Should not conflict with txn2 since snapshot wasn't set until GetForUpdate was called.
88926. Verify that A and B are no longer locked
88927. If cur_txn is not started, do prepare for it.
88928. Lock keys in random order.
88929. issue rollback
88930. Rollback to 4
88931. heap should not care about prepared section anymore
88932. This Put outside of a transaction will conflict with the previous write
88933. Write a bunch of keys to db to force a compaction
88934. Verify snapshots get reinitialized correctly
88935. Open DB with a lock limit of 3
88936. Verify the exclusivity field of the transactions in the deadlock path.
88937. expired!
88938. txn should commit since the flushed table is still in MemtableList History
88939. Verify A,B,C,D,E,F are still locked
88940. Offset of each txn id from the root of the shared dlock tree's txn id.
88941. we should see txn1s log referenced by the memtables
88942. lock key in cfa
88943. verify non txn data
88944. Put a random key so we have a memtable to flush
88945. Modify BBB before snapshot is taken
88946. Untracked writes should succeed even though key was written after snapshot
88947. As an extra check, check if prepared set will be properly empty after they are committed.
88948. txn read
88949. Wait for all threads to run
88950. flush to next wal
88951. can't have duplicate name
88952. Column family is 0 (default) or 1.
88953. Test different table factories
88954. 500ms
88955. These keys do not conflict with existing writes since they're in different column families
88956. Snapshot has now been cleared
88957. open the new column families
88958. lock key in default cf
88959. Number of gaps applied so far
88960. 2
88961. heap should now show txn2s log
88962. Rollback to 3
88963. 100ms
88964. Modify key after transaction start
88965. We continue until max advances a bit beyond the snapshot.
88966. Verify that A is now unlocked
88967. exclusive
88968. Should succeed after verifying there is no write to X in SST file
88969. still not available to db
88970. transaction writes have an infinite timeout, but we will override this when we start a txn db writes have infinite timeout
88971. Verify that data is consistent
88972. Iterates backwards over path verifying decreasing txn_ids.
88973. cfb now has data from LOG A
88974. commit txn2
88975. Read and write without a snap
88976. flush default cf to create new log
88977. verify foo3 is locked by txn
88978. we should see that cfb now has a flush requested
88979. Leave some gap between the preliminary snapshots and the final snapshot that we check. This should test for also different overlapping scenarios between the last snapshot and the commits.
88980. The final snapshot that we will inspect
88981. Read and write with snapshot
88982. Test txn1 and txn2 sharing a lock and txn3 trying to obtain it.
88983. name too short
88984. issue rollback (marker written to WAL)
88985. Create 2 new column families
88986. Should commit since not expired
88987. 100ms txn timeout no longer infinite
88988. Verify A,B are locked and C is not
88989. Verify that A and B are still locked
88990. Test to make sure transactions work when there are no other writes in an empty db.
88991. txn1 should fail to commit since it is expired
88992. **comment:** Take some preliminary snapshots first. This is to stress the data structure that holds the old snapshots as it will be designed to be efficient when only a few snapshots are below the max_evicted_seq_.
label: code-design
88993. prepre transaction in LOG A
88994. "X" currently locked
88995. regular db put
88996. Rollback to 1
88997. Read and write without a snapshot
88998. data not in mem yet
88999. Verify that A is locked
89000. Should fail after encountering a write to S in memtable
89001. Verify A,B,E,F are still locked and C,D,G,H are not.

89002. T=0:Insert Set1.
89003. Puts num_entries starting from start_pos_map from kvmap_ into the database
89004. If TTL is non positive or not provided, the behaviour is TTL = infinity This test opens the db 3 times with such default behavior and inserts a bunch of kvs each time. All kvs should accumulate in the db till the end Partitions the sample-size provided into 3 sets over boundary1 and boundary2
89005. T=0:Open the db with ttl = 2 T=0:Insert Set1. Delete at t=2 T=1:Set1 should still be there
89006. Makes a write-batch with key-vals from kvmap_ and 'Write's it
89007. Choose carefully so that Put, Gets & Compaction complete in 1 second buffer
89008. T=0:Open the db with ttl = 2 T=0:Insert Set1. Delete at t=2 T=2:Set1 should not be there
89009. check all insertions done
89010. only column family 1 should be alive after 4 seconds
89011. T=2:Set1 should still be there
89012. Works on keys of the form "key<number>" Drops key if number at the end of key is in [0, kSampleSize_/3), Keeps key if it is in [kSampleSize_/3, 2*kSampleSize_/3), Change value if it is in [2*kSampleSize_/3, kSampleSize_) Eg. kSampleSize_=6. Drop:key0-1...Keep:key2-3...Change:key4-5...
89013. Resets the timestamp of a set of kvs by updating them and checks that they are not deleted according to the old timestamp
89014. dbiter should have found out kvmap_[st_pos]
89015. T=0>Delete at t=1 T=2: Set should not be there
89016. T=0:Open the db normally T=0:Insert Set1. Delete at t=1
89017. everything should be there after 1 second
89018. Sleeps for slp_tim then runs a manual compaction Checks span starting from st_pos from kvmap_ in the db and Gets should return true if check is true and false otherwise Also checks that value that we got is the same as inserted; and =knewValue if test_compaction_change is true
89019. Part dropped Part kept Part changed
89020. nothing should be there after 6 seconds
89021. Checks presence while opening the same db more than once with the same ttl Note: The second open will open the same db
89022. Similar to PresentDuringTTL but uses Iterator
89023. T=0: Insert. Delete at t=1 T=2: Should not be there
89024. Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
89025. Similar as SleepCompactCheck but uses TtlIterator to read from db
89026. db should be closed before opening again
89027. Open database with TTL support in read_only mode
89028. T=0: Insert. Delete at t=2
89029. Puts a set of values and checks its absence using Get after ttl
89030. class TtlTest
89031. T=0: Insert Set1. Delete at t=3 T=2 T=2: Insert Set1. Delete at t=5 T=4: Set1 should still be there
89032. Populates and returns a kv-map
89033. !ROCKSDB_LITE
89034. checks the whole kvmap_ to return correct values using KeyMayExist
89035. checks the whole kvmap_ to return correct values using MultiGet
89036. T=3: Set3 never deleted T=4: Sets 1,2,3 still there
89037. namespace rocksdb
89038. Keep keys not matching the format "key<NUMBER>"
89039. Open database with TTL support when TTL provided with db_ttl_ pointer
89040. Runs a manual compaction
89041. T=0. Delete at t=2 T=1: Set should be there
89042. T=0:Insert Set1. Delete at t=1 T=2: TTL logic takes precedence over TestFilter:-Set1 should not be there
89043. ensure that compaction is kicked in to always strip timestamp from kvs
89044. T=0: Insert. Delete at t=2 T=1: Set should be there
89045. T=0: Set1 never deleted T=1: Set1 still there
89046. T=0: Set deleted at t=3 T=2: Set should be there
89047. Insert some key-values which KeyMayExist should be able to get and check that values returned are fine
89048. Put a mock kv at the end because CompactionFilter doesn't delete last key
89049. A black-box test for the ttl wrapper around rocksdb
89050. Open with TestFilter compaction filter
89051. Similar to AbsentAfterTTL but uses Iterator
89052. Checks presence during ttl in read_only mode
89053. Checks user's compaction filter for correctness with TTL logic
89054. T=1: Set2 never deleted T=2: Sets1 & 2 still there
89055. compaction should take place always from level0 for determinism
89056. Checks whether WriteBatch works well with TTL Puts all kvs in kvmap_ in a batch and writes first, then deletes first half
89057. T=0: Insert. Delete at t=1
89058. Checks absence while opening the same db more than once with the same ttl Note: The second open will open the same db
89059. Puts a set of values and checks its presence using Get during ttl
89060. Checks presence while opening the same db more than once with bigger ttl
89061. Open database with TTL support when TTL not provided with db_ttl_ pointer
89062. namespace rocksdb
89063. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory).
89064. Copyright (c) 2011-present, Facebook, Inc. All rights reserved. This source code is licensed under both the GPLv2 (found in the COPYING file in the root directory) and Apache 2.0 License (found in the LICENSE.Apache file in the root directory). Copyright (c) 2011 The LevelDB Authors. All rights reserved. Use of this source code is governed by a BSD-style license that can be found in the LICENSE file. See the AUTHORS file for names of contributors.
89065. Verify index column family
89066. only delta has it
89067. Sort entries by value
89068. rollback to 6
89069. both has it. Delta should win
89070. rollback to 4
89071. Verify WriteBatch can be iterated
89072. 5
89073. namespace anonymous
89074. rollback to 5
89075. In this test, we insert <key, value> to column family `data`, and <value, key> to column family `index`. Then iterator them in order and seek them by key.
89076. Neither iterator has it.
89077. Seek to First
89078. Next
89079. Seek to random key
89080. Random walk and make sure iter and result_iter returns the same key and value
89081. new value
89082. 1

89083. only delta has it. Delta is delete
89084. Iterator all keys
89085. Prev
89086. Iterator all indexes
89087. Seek to every key
89088. 2
89089. namespace
89090. no savepoint found
89091. Seek the keys one by one in reverse order
89092. Tests that we can write to the WBWI while we iterate (from a single thread). iteration should see the newest writes
89093. Test batch with overwrite_key=true
89094. "mm" is deleted, so we're back at "m"
89095. IROCKSDB_LITE
89096. both has it. Delta is delete
89097. Seek to last
89098. 6
89099. still the same value
89100. stress testing mutations with IteratorWithBase
89101. 3 4
89102. only base has it
89103. Verify data column family
89104. overwrite_key
89105. rollback to 1
89106. Clear batch and re-run test with new values
89107. Sort entries by key
89108. Test the case that there is one element in the write batch
89109. Seek to every index
89110. rollback to 3
89111. same thing as above, but testing IteratorWithBase
89112. No merge operator specified.
89113. rollback to 2
89114. default column family
89115. **comment:** Do not drop data if compile with ASAN to suppress leak warning.
 label: code-design
89116. !__SANITIZE_ADDRESS__
89117. Write batch into transaction db Merge a key inside a transaction
89118. Put a key outside a transaction
89119. Read a key inside a transaction Read a key inside a transaction
89120. Merge a key outside a transaction
89121. Create an iterator outside a transaction
89122. Deletions obsoleted before bottom level due to file gap optimization.
89123. avoid underflow
89124. Whether there is an active snapshot in range [lower_bound, upper_bound).
89125. Set memtable_info for memtable sealed callback
89126. Blob DB only Blob DB only
89127. **comment:** TODO(yiwu): The property is currently available for fifo compaction with allow_compaction = false. This is because we don't propagate oldest_key_time on compaction.
 label: code-design
89128. version
89129. db
89130. If fail, the timestamp is already set.
89131. if user keeps adding entries that exceeds write_buffer_size, we need to flush earlier even though we still have much available memory left.
89132. Dynamically change the memtable's capacity. If set below the current usage, the next key added will trigger a flush. Can only increase size when memtable prefix bloom is disabled, since we can't easily allocate more space.
89133. Timestamp of oldest key
89134. Dynamically changeable memtable option
89135. Returns an estimate of the timestamp of the earliest key.
89136. retrieve all snapshot numbers up until max_seq. They are sorted in ascending order.
89137. Whether there is an active snapshot in range [lower_bound, upper_bound).
89138. Make sure there are no files on or beyond num_levels().
89139. **comment:** Store states of levels larger than num_levels_. We do this instead of storing them in levels_ to avoid regression in case there are no files on invalid levels. The version is not consistent if in the end the files on invalid levels don't cancel out.
 label: code-design
89140. Whether there are invalid new files or invalid deletion on levels larger than num_levels_.
89141. Creating an already existing file on invalid level.
89142. Deleting an non-existing file on invalid level.
89143. Same as PutCF except for value type.
89144. The Slice pointed by pinnable_val is not valid after this point
89145. Reset PinnableSlice after each use and before each reuse
89146. If the value is not pinned, the internal buffer must have the value.
89147. If it cannot pin the value, it copies the value to its internal buffer. The internal buffer could be set during construction.
89148. This snapshot should be freed using rocksdb_free
89149. Move constructor and move assignment is allowed.
89150. No copy constructor and copy assignment allowed.
89151. used internally by BlobDB.
89152. "rocksdb.estimate-oldest-key-time" - returns an estimation of oldest key timestamp in the DB. Currently only available for FIFO compaction with compaction_options_fifo.allow_compaction = false.
89153. "rocksdb.estimate-oldest-key-time"
89154. Determines whether the MergeOperator can be called with just a single merge operand. Override and return true for allowing a single operand. FullMergeV2 and PartialMerge/PartialMergeMulti should be implemented accordingly to handle a single operand.
89155. bytes for vals returned by Get bytes for vals returned by MultiGet bytes for keys/vals decoded by iterator
89156. No copy constructor and copy assignment allowed.
89157. Deletions obsoleted before bottom level due to file gap optimization.
89158. Timestamp of the earliest key. 0 means unknown.
89159. An overload of the the above method that receives a PinnableSlice For backward compatibility a default implementation is provided
89160. **comment:** TODO(myabandeh): Not implemented yet write data after the prepare phase of 2pc TODO(myabandeh): Not implemented yet write data before the prepare phase of 2pc
 label: requirement

89161. empty path, limit exceeded constructor and default constructor
89162. Stores the number of latest deadlocks to track
89163. An overload of the the above method that receives a PinnableSlice
89164. * Class: org_rocksdb_SstFileWriter * Method: merge * Signature: (J[B[B)V
89165. * Class: org_rocksdb_SstFileWriter * Method: delete * Signature: (JJ)V
89166. exception thrown: OutOfMemoryError
89167. * Class: org_rocksdb_SstFileWriter * Method: put * Signature: (JJ)V
89168. * * Add a Merge key with value to currently opened file. * * @param key the specified key to be merged. * @param value the value to be merged with the current value for * the specified key. * * @throws RocksDBException thrown if error happens in underlying * native library.
89169. * * Add a deletion key to currently opened file. * * @param key the specified key to be deleted. * * @throws RocksDBException thrown if error happens in underlying * native library.
89170. * * Add a Put key with value to currently opened file. * * @param key the specified key to be inserted. * @param value the value associated with the specified key. * * @throws RocksDBException thrown if error happens in underlying * native library.
89171. If you change this, you also need to change size of array buckets_ in HistogramImpl
89172. **comment:** Extracts two most significant digits to make histogram buckets more human-readable. E.g., 172 becomes 170.
label: code-design
89173. 109==BucketMapper::BucketCount() namespace rocksdb
89174. to PlainTableOptions just like bloom_bits_per_key
89175. **comment:** Separate inlines so they can be replaced if needed
label: code-design
89176. Before read partitions, prefetch them to avoid lots of IOs
89177. After prefetch, read the partitions one by one
89178. The partitions of partitioned index are always stored in cache. They are hence follow the configuration for pin and prefetch regardless of the value of cache_index_and_filter_blocks
89179. prefetch_buffer
89180. Read the last block's offset
89181. This is a possible scenario since block cache might not have had space for the partition
89182. on-stack BlockIter while the state is on heap. Currently it assumes the first level iter is always on heap and will attempt to delete it in its destructor.
89183. Index partitions are assumed to be consecutive. Prefetch them all. Read the first block offset
89184. pre-fetching of blocks is turned on
89185. Refer to the comment above about partitioned indexes always being cached
89186. **comment:** unused
label: code-design
89187. Prefetch all the blocks referenced by this index to the buffer
89188. checksum type (char, 1 byte)
89189. Blob value not supported. Stop.
89190. **comment:** TODO(myabandeh): if instead of filter object we store only the blocks in block cache, then we don't have to manually erase them from block cache here.
label: code-design
89191. Before read partitions, prefetch them to avoid lots of IOs
89192. After prefetch, read the partitions one by one
89193. **comment:** TODO(myabandeh): merge this with the same function in IndexReader
label: code-design
89194. prefetch_buffer
89195. Read the last block's offset
89196. This is a possible scenario since block cache might not have had space for the partition
89197. Index partitions are assumed to be consecutive. Prefetch them all. Read the first block offset
89198. **comment:** If enabled, blob DB periodically cleanup stale data by rewriting remaining live data in blob files to new files. If garbage collection is not enabled, blob files will be cleanup based on TTL.
label: code-design
89199. The smallest value to store in blob log. Value larger than this threshold will be inlined in base DB together with the key.
89200. Disable all background job. Used for test only.
89201. wopts
89202. options
89203. column_family
89204. Blob DB doesn't support non-default column family.
89205. **comment:** TODO(yiwu): Blob indexes will be remove by BlobIndexCompactionFilter. We can just drop the blob record.
label: requirement
89206. Write callback for garbage collection to check if key has been updated since last read. Similar to how OptimisticTransaction works. See inline comment in GCFfileAndUpdateLSM().
89207. Allocate the buffer. This is safe in C++11 Note that std::string::reserved() does not work, since previous value of the buffer can be larger than blob_index.size().
89208. Upper bound of sequence number to proceed.
89209. Error.
89210. error
89211. value_found
89212. Get a snapshot to avoid blob file get deleted between we fetch and index entry and reading from the file.
89213. Similar to OptimisticTransaction, we obtain latest_seq from base DB, which is guaranteed to be no smaller than the sequence of current key. We use a WriteCallback on write to check the key sequence on write. If the key sequence is larger than latest_seq, we know a new version is inserted and the old blob can be discarded. We cannot use OptimisticTransaction because we need to pass is_blob_index flag to GetImpl.
89214. allow_blob
89215. concrete at this time. also necessary to add to open_ttl_files_
89216. Key to check
89217. Hold a lock in the beginning to avoid updates to base DB during the call
89218. **comment:** TODO(yiwu): In case there are multiple writers the latest sequence would not be the actually sequence we are writing. Need to get the sequence from write batch after DB write instead.
label: code-design
89219. Inlined with TTL
89220. cache_only
89221. Double check the file is not obsolete by others
89222. Either the key is deleted or updated with a newer version which is inlined in LSM.
89223. **comment:** Get a snapshot to avoid blob file get deleted between we fetch and index entry and reading from the file. TODO(yiwu): For Get() retry if file not found would be a simpler strategy.
label: code-design
89224. reset oldest_file_evicted flag
89225. **comment:** TODO(yiwu): If index_entry is a PinnableSlice, we can also pin the same memory buffer to avoid extra copy.
label: code-design
89226. Release write_mutex_ before DB write to avoid race condition with flush begin listener, which also require write_mutex_ to sync blob files.
89227. Put as normal value

89228. **comment:** TODO(yiwu): correct the stats and expose it.
label: code-design
89229. **comment:** TODO(yiwu): Add an option to skip crc checking.
label: code-design
89230. Close a file if its size exceeds blob_file_size
89231. opened non-TTL blob file.
89232. Close a file by appending a footer, and removes file from open files list.
89233. when should oldest file be evicted: on reaching 90% of blob_dir_size
89234. Create a snapshot if there isn't one in read options. Return true if a snapshot is created.
89235. **comment:** TODO(yiwu): Should return column family id encoded in blob file after we add blob db column family support.
label: code-design
89236. Format of blob log file header (30 bytes): +-----+-----+-----+-----+-----+ | magic number | version | cf id | flags | compression | expiration range | +-----+-----+-----+-----+-----+ | Fixed32 | Fixed32 | Fixed32 | char | char | Fixed64 Fixed64 | +-----+-----+-----+-----+-----+ List of flags: has_ttl: Whether the file contain TTL data. Expiration range in the header is a rough range based on blob_db_options.ttl_range_secs.
89237. header include fields up to blob CRC
89238. Format of blob log file footer (48 bytes): +-----+-----+-----+-----+-----+ | magic number | blob count | expiration range | sequence range | footer CRC | +-----+-----+-----+-----+-----+ | Fixed32 | Fixed64 | Fixed64 + Fixed64 | Fixed64 + Fixed64 | Fixed32 | +-----+-----+-----+-----+-----+ The footer will be presented only when the blob file is properly closed. Unlike the same field in file header, expiration range in the footer is the range of smallest and largest expiration of the data in this file.
89239. Blob record format (32 bytes header + key + value): +-----+-----+-----+-----+-----+-----+ | key length | value length | expiration | header CRC | blob CRC | key | value | +-----+-----+-----+-----+-----+-----+ | Fixed64 | Fixed64 | Fixed64 | Fixed32 | Fixed32 | key len | value len | +-----+-----+-----+-----+-----+-----+ If file has has_ttl = false, expiration field is always 0, and the blob doesn't have expiration. Also note that if compression is used, value is compressed value and value length is compressed value length. Header CRC is the checksum of (key_len + val_len + expiration), while blob CRC is the checksum of (key + value). We could use variable length encoding (Varint64) to save more space, but it makes reader more complicated.
89240. 0x00248f37
89241. **comment:** A very rare event, in which the commit entry is updated before we do. Here we apply a very simple solution of retrying. TODO(myabandeh): do precautions to detect bugs that cause infinite loops
label: code-design
89242. We should not normally reach here
89243. It is committed and also not evicted from commit cache
89244. This is to avoid updating the snapshots_ if it already updated with a more recent version by a concurrent thread
89245. The list might get updated concurrently as we are reading from it. The reader should be able to read all the snapshots that are still valid after the update. Since the survived snapshots are written in a higher place before getting overwritten the reader that reads bottom-up will eventually see it.
89246. **comment:** TODO(myabandeh): implement lock-free commit_cache_
label: code-design
89247. then snapshot_seq < commit_seq overlapping range
89248. When advancing max_evicted_seq_, we move older entries from prepared to delayed_prepared_. Also we move evicted entries from commit cache to old_commit_map_ if it overlaps with any snapshot. Since prep_seq <= max_evicted_seq_, we have three cases: i) in delayed_prepared_, ii) in old_commit_map_, iii) committed with no conflict with any snapshot (i) delayed_prepared_ is checked above then (ii) cannot be the case only (iii) is the case: committed commit_seq <= max_evicted_seq_ < snapshot_seq => commit_seq < snapshot_seq
89249. initialized
89250. **comment:** Here we try to infer the return value without looking into prepare list. This would help avoiding synchronization over a shared map.
TODO(myabandeh): read your own writes TODO(myabandeh): optimize this. This sequence of checks must be correct but not necessarily efficient
label: code-design
89251. At this point we don't know if it was committed or it is still prepared
89252. else (ii) might be the case: check the commit data saved for this snapshot. If there was no overlapping commit entry, then it is committed with a commit_seq lower than any live snapshot, including snapshot_seq.
89253. When max_evicted_seq_ advances, move older entries from prepared_txns_ to delayed_prepared_. This guarantees that if a seq is lower than max, then it is not in prepared_txns_ and save an expensive, synchronized lookup from a shared set. delayed_prepared_ is expected to be empty in normal cases.
89254. Not evicted from cache and also not present, so must be still prepared
89255. continue the search if the next snapshot could be smaller than commit_seq
89256. We update the list concurrently with the readers. Both new and old lists are sorted and the new list is subset of the previous list plus some new items. Thus if a snapshot repeats in both new and old lists, it will appear upper in the new list. So if we simply insert the new snapshots in order, if an overwritten item is still valid in the new list is either written to the same place in the array or it is written in a higher place before it gets overwritten by another item. This guarantees a reader that reads the list bottom-up will eventually see a snapshot that repeats in the update, either before it gets overwritten by the writer or afterwards.
89257. We use this to identify how fresh are the snapshot list. Since this is done atomically with obtaining the snapshot list, the one with the larger seq is more fresh. If the seq is equal the full snapshot list could be different since taking snapshots does not increase the db seq. However since we only care about snapshots before the new max, such recent snapshots would not be included in the list anyway.
89258. snapshot_seq < prep_seq <= commit_seq => snapshot_seq < commit_seq
89259. If we do not store an entry in old_commit_map we assume it is committed in all snapshots. if commit_seq <= snapshot_seq, it is considered already in the snapshot so we need not to keep the entry around for this snapshot.
89260. **comment:** TODO(myabandeh) inc max in larger steps to avoid frequent updates
label: code-design
89261. Storing once is enough. No need to check it for other snapshots.
89262. continue the search if the next snapshot could be larger than prep_seq
89263. We only care about snapshots lower than max
89264. Returns true if commit_seq <= snapshot_seq
89265. After each eviction from commit cache, check if the commit entry should be kept around because it overlaps with a live snapshot. First check the snapshot cache that is efficient for concurrent access
89266. 10m entry, 80MB size
89267. Items could have moved from the snapshots_ to snapshot_cache_ before acquiring the lock. To make sure that we do not miss a valid snapshot, read snapshot_cache_ again while holding the lock.
89268. With each change to max_evicted_seq_ fetch the live snapshots behind it
89269. It is not committed, so it must be still prepared
89270. with any concurrent transactions. The easiest way to do this is to wrap all
89271. (ii) if it is the case: it is committed but after the snapshot_seq
89272. Update the size at the end. Otherwise a parallel reader might read items that are not set yet.
89273. **comment:** Then access the less efficient list of snapshots_
label: code-design
89274. **comment:** Insert them to a vector that is less efficient to access concurrently
label: code-design
89275. Then it is not committed yet
89276. Rewrite the entry with the index indexed_seq in the commit table with the commit entry <prep_seq, commit_seq>. If the rewrite results in eviction, sets the evicted_entry and returns true.
89277. Add the transaction with prepare sequence seq to the prepared list

89278. Add a new entry to `old_commit_map_` if `prep_seq <= snapshot_seq < commit_seq`. Return false if checking the next snapshot(s) is not needed. This is the case if the entry already added to `old_commit_map_` or none of the next snapshots could satisfy the condition. `next_is_larger`: the next snapshot will be a larger value
89279. **comment:** 2nd list for storing snapshots. The list sorted in ascending order. Thread-safety is provided with `snapshots_mutex_`.
label: requirement
89280. Rewrite the entry with the index `indexed_seq` in the commit table with the commit entry `new_entry` only if the existing entry matches the `expected_entry`. Returns false otherwise.
89281. Check whether the transaction that wrote the value with sequence number `seq` is visible to the snapshot with sequence number `snapshot_seq`
89282. A heap of prepared transactions. Thread-safety is provided with `prepared_mutex_`.
89283. **comment:** A set of long-running prepared transactions that are not finished by the time `max_evicted_seq` advances their sequence number. This is expected to be empty normally. Thread-safety is provided with `prepared_mutex_`.
label: requirement
89284. Already popped, ignore it.
89285. **comment:** The list sorted in ascending order. Thread-safety for writes is provided with `snapshots_mutex_` and concurrent reads are safe due to `std::atomic` for each entry. In x86_64 architecture such reads are compiled to simple read instructions. 128 entries TODO(myabandeh): avoid non-const static variables
label: requirement
89286. **comment:** unused
label: code-design
89287. Add the transaction with prepare sequence `prepare_seq` and commit sequence `commit_seq` to the commit map
89288. The largest evicted *commit* sequence number from the `commit_cache_`
89289. A map of the evicted entries from `commit_cache_` that has to be kept around to service the old snapshots. This is expected to be empty normally. Thread-safety is provided with `old_commit_map_mutex_`.
89290. Update when `delayed_prepared_.empty()` changes. Expected to be true normally.
89291. The version of the latest list of snapshots. This can be used to avoid rewriting a list that is concurrently updated with a more recent version.
89292. The list of live snapshots at the last time that `max_evicted_seq` advanced. The list stored into two data structures: in `snapshot_cache_` that is efficient for concurrent reads, and in `snapshots_` if the data does not fit into `snapshot_cache_`. The total number of snapshots in the two lists
89293. A heap with the amortized O(1) complexity for erase. It uses one extra heap to keep track of erased entries that are not yet on top of the main heap.
89294. `commit_cache_` must be initialized to zero to tell apart an empty index from a filled one. Thread-safety is provided with `commit_cache_mutex_`.
89295. Get the commit entry with index `indexed_seq` from the commit table. It returns true if such entry exists.
89296. (`heap_top() > seq`) Down the heap, remember to pop it later
89297. **comment:** TODO(myabandeh): avoid non-const static variables
label: code-design
89298. Update when `old_commit_map_.empty()` changes. Expected to be true normally.
89299. else value is already assigned
89300. `read_only`
89301. Resize the buffer to the target size and restore the buffer's idx
89302. Reversing the normalized vector returns the latest deadlocks first
89303. Next write occurs at a nonexistent path's slot
89304. Drop the deadlocks that will no longer be needed after the normalize
89305. callback
89306. **comment:** TODO(myabandeh): prevent the users from writing to txns after the prepare phase
label: code-design
89307. log ref
89308. any operations appended to this `working_batch` will be ignored from WAL
89309. We take the commit-time batch and append the Commit marker. The Memtable will ignore the Commit marker in non-recovery mode
89310. else value is already assigned
89311. Since the lifetime of the WriteBatch is the same as that of the transaction we cannot pin it as otherwise the returned value will not be available after the transaction finishes.

git_commits:

- summary:** MINIFICPP-303 Update to civet 1.10
message: MINIFICPP-303 Update to civet 1.10 MINIFICPP-303 Update to RocksDB 5.8.6 Use variables to store civet and rocksdb location Potential civetweb Xcode 9 fix This closes #185. Signed-off-by: Marc Parisi <phrocker@apache.org>

github_issues:

github_issues_comments:

github_pulls:

- title:** MINIFICPP-303 Upgrade civetweb and rocksdb
body: Thank you for submitting a contribution to Apache NiFi - MiNiFi C++. In order to streamline the review of the contribution we ask you to ensure the following steps have been taken: **### For all changes:** - [x] Is there a JIRA ticket associated with this PR? Is it referenced in the commit message? - [x] Does your PR title start with MINIFI-XXXX where XXXX is the JIRA number you are trying to resolve? Pay particular attention to the hyphen "-" character. - [x] Has your PR been rebased against the latest commit within the target branch (typically master)? - [] Is your initial contribution a single, squashed commit? **### For code changes:** - [] If adding new dependencies to the code, are these dependencies licensed in a way that is compatible for inclusion under [ASF 2.0] (<http://www.apache.org/legal/resolved.html#category-a>)? - [] If applicable, have you updated the LICENSE file? - [] If applicable, have you updated the NOTICE file? **### For documentation related changes:** - [] Have you ensured that format looks appropriate for the output in which it is rendered? **### Note:** Please ensure that once the PR is submitted, you check travis-ci for build issues and submit an update to your PR as soon as possible.

github_pulls_comments:

1. Looks good.
2. I can build & make test this on CentOS 7.4 with devtoolset-6 (GCC 6.3.1), but it fails when using devtoolset-7 (GCC 7.1.1). ``` In file included from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/write_batch_internal.h:12:0, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/column_family.h:20, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/version_set.h:31, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/compaction.h:11, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/compaction_iterator.h:12, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/builder.cc:16: /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/write_thread.h: In member function `std::mutex& rocksdb::WriteThread::Writer::StateMutex() const': /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/write_thread.h:225:78: error: dereferencing type-punned pointer will break strict-aliasing rules [-Werror=strict-aliasing] return *static_cast<std::mutex*>(static_cast<void*>(&state_mutex_bytes)); ``` I would still merge this because I wasn't able to build with devtoolset-7 before this change, either. It's an improvement to have the newer versions, and I can't find any regressions.
3. **body:** @calebj can you define a top level variable for the path to civet and rocksdb and use that instead of the hardcoded path?
label: code-design
4. @calebj did anything in the process change? I can't build "No VERSION specified for WRITE_BASIC_CONFIG_VERSION_FILE()" Usually this indicates a version issue but I have version 3.10 on mac.
5. Weird. Nothing in the process should have changed. Where is that happening?

6. CMake Error at /usr/local/Cellar/cmake/3.10.0/share/cmake/Modules/WriteBasicConfigVersionFile.cmake:40 (message): No VERSION specified for WRITE_BASIC_CONFIG_VERSION_FILE() Call Stack (most recent call first):


```
/usr/local/Cellar/cmake/3.10.0/share/cmake/Modules/CMakePackageConfigHelpers.cmake:207 (write_basic_config_version_file) CMakeLists.txt:644
(write_basic_package_version_file) I don't see this in a rocksdb 5.8 clean checkout. I'm on OSX.
```
7. See if it works after the next push. I'm putting 5.8.6 in afresh and disabling the tests in a cleaner fashion.
8. @calebj Re the next push, did you force push into that last commit?
9. I did. The new commit messages reflect the new version, too.
10. @calebj the travis failure shows you what we've been seeing on clang 9 <https://travis-ci.org/apache/nifi-minifi-cpp/jobs/309102064> It might be better to separate the upgrade of civet and rocksdb?
11. @calebj I see the change. I'll attempt to merge this into master and run the build before pushing. thanks!
12. @calebj I merged this already, but the sync hasn't occurred. Sometimes we need to perform another merge to break it free <https://git-wip-us.apache.org/repos/asf?p=nifi-minifi-cpp.git> for reference.
13. I see it in df353561c8a8a0a0d1178a76f03b966b44542834. Hopefully that commit hash will resolve once the sync is complete. Thanks!

github_pulls_reviews:

jira_issues:

1. **summary:** Upgrade civetweb and rocksdb

description: These upgrades to the latest stable releases of civetweb ([v1.10|<https://github.com/civetweb/civetweb/releases/tag/v1.10>]) and RocksDB ([v5.8|<https://github.com/facebook/rocksdb/releases/tag/v5.8>]) bring numerous improvements. They might solve any issues similar to those in MINIFICPP-262. For civet, the most notable is OpenSSL v1.1 support. It was partially backported to the version in the minifi tree (1.9.1), but still had issues matching some OpenSSL v1.1 interface changes. Civetweb release v1.10 has complete and official support. RocksDB v5.8 comes with the following bugfixes: * Fix wrong latencies in rocksdb.db.get.micros, rocksdb.db.write.micros, and rocksdb.sst.read.micros. * Fix incorrect dropping of deletions during intra-L0 compaction. * Fix transient reappearance of keys covered by range deletions when memtable prefix bloom filter is enabled. * Fix potentially wrong file smallest key when range deletions separated by snapshot are written together.

jira_issues_comments:

1. GitHub user calebj opened a pull request: <https://github.com/apache/nifi-minifi-cpp/pull/185> MINIFICPP-303 Upgrade civetweb and rocksdb Thank you for submitting a contribution to Apache NiFi - MiNiFi C++. In order to streamline the review of the contribution we ask you to ensure the following steps have been taken: **For all changes:** - [x] Is there a JIRA ticket associated with this PR? Is it referenced in the commit message? - [x] Does your PR title start with MINIFI-XXXX where XXXX is the JIRA number you are trying to resolve? Pay particular attention to the hyphen "-" character. - [x] Has your PR been rebased against the latest commit within the target branch (typically master)? - [] Is your initial contribution a single, squashed commit? **For code changes:** - [] If adding new dependencies to the code, are these dependencies licensed in a way that is compatible for inclusion under [ASF 2.0] (<http://www.apache.org/legal/resolved.html#category-a>)? - [] If applicable, have you updated the LICENSE file? - [] If applicable, have you updated the NOTICE file? **For documentation related changes:** - [] Have you ensured that format looks appropriate for the output in which it is rendered? **Note:** Please ensure that once the PR is submitted, you check travis-ci for build issues and submit an update to your PR as soon as possible. You can merge this pull request into a Git repository by running: \$ git pull <https://github.com/NiFiLocal/nifi-minifi-cpp> MINIFICPP-303 Alternatively you can review and apply these changes as the patch at: <https://github.com/apache/nifi-minifi-cpp/pull/185.patch> To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #185 -----
2. Caleb, did this solve your issues for MINIFICPP-262? I gathered that was no longer an issue and thus we are upgrading just to upgrade, correct?
3. Yes, it solved both of the problems I was having on Arch when I dropped the new versions in. I just didn't submit them as a patch until now.
4. [~calebj] Your comment in 262 was "I can't reproduce the issue either on a fresh Trusty VM," but yet you were having issues on Cloud9. Did this solve the Cloud9 issue for you ? I saw your comment, "It might solve." I don't want to close 262 until I can verify from [~christiansanson] that 262 was a problem he could/can replicate and is still a problem. Sorry I said, "before merging," but I don't plan to hold merging this to get an understanding. The plan is still to merge after some testing.
5. Github user achristianson commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> Looks good.
6. Github user achristianson commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> I can build & make test this on CentOS 7.4 with devtoolset-6 (GCC 6.3.1), but it fails when using devtoolset-7 (GCC 7.1.1). ` In file included from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/write_batch_internal.h:12:0, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/column_family.h:20, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/version_set.h:31, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/compaction.h:11, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/compaction_iterator.h:12, from /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/builder.cc:16: /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/write_thread.h: In member function `std::mutex& rocksdb::WriteThread::Writer::StateMutex()': /home/achristianson/workspace/nifi-minifi-cpp/thirdparty/rocksdb/db/write_thread.h:225:78: error: dereferencing type-punned pointer will break strict-aliasing rules [-Werror=strict-aliasing] return *static_cast<std::mutex*>(*static_cast<void*>(&state_mutex_bytes));` I would still merge this because I wasn't able to build with devtoolset-7 before this change, either. It's an improvement to have the newer versions, and I can't find any regressions.
7. Github user phrocker commented on the issue: [@calebj can you define a top level variable for the path to civet and rocksdb and use that instead of the hardcoded path?](https://github.com/apache/nifi-minifi-cpp/pull/185)
8. Github user phrocker commented on the issue: [@calebj did anything in the process change? I can't build "No VERSION specified for WRITE_BASIC_CONFIG_VERSION_FILE\(\)" Usually this indicates a version issue but I have version 3.10 on mac.](https://github.com/apache/nifi-minifi-cpp/pull/185)
9. Github user calebj commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> Weird. Nothing in the process should have changed. Where is that happening?
10. Github user phrocker commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> CMake Error at /usr/local/Cellar/cmake/3.10.0/share/cmake/Modules/WriteBasicConfigVersionFile.cmake:40 (message): No VERSION specified for WRITE_BASIC_CONFIG_VERSION_FILE() Call Stack (most recent call first):


```
/usr/local/Cellar/cmake/3.10.0/share/cmake/Modules/CMakePackageConfigHelpers.cmake:207 (write_basic_config_version_file) CMakeLists.txt:644
(write_basic_package_version_file) I don't see this in a rocksdb 5.8 clean checkout. I'm on OSX.
```
11. Github user calebj commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> See if it works after the next push. I'm putting 5.8.6 in afresh and disabling the tests in a cleaner fashion.
12. Github user phrocker commented on the issue: [@calebj Re the next push, did you force push into that last commit?](https://github.com/apache/nifi-minifi-cpp/pull/185)
13. Github user calebj commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> I did. The new commit messages reflect the new version, too.
14. Github user phrocker commented on the issue: [@calebj the travis failure shows you what we've been seeing on clang 9 <https://travis-ci.org/apache/nifi-minifi-cpp/jobs/309102064> It might be better to separate the upgrade of civet and rocksdb?](https://github.com/apache/nifi-minifi-cpp/pull/185)
15. Github user phrocker commented on the issue: [@calebj I see the change. I'll attempt to merge this into master and run the build before pushing, thanks!](https://github.com/apache/nifi-minifi-cpp/pull/185)
16. Github user phrocker commented on the issue: [@calebj I merged this already, but the sync hasn't occurred. Sometimes we need to perform another merge to break it free <https://git-wip-us.apache.org/repos/asf?p=nifi-minifi-cpp.git> for reference.](https://github.com/apache/nifi-minifi-cpp/pull/185)
17. Github user calebj commented on the issue: <https://github.com/apache/nifi-minifi-cpp/pull/185> I see it in df353561c8a8a0a0d1178a76f03b966b44542834. Hopefully that commit hash will resolve once the sync is complete. Thanks!
18. Github user calebj closed the pull request at: <https://github.com/apache/nifi-minifi-cpp/pull/185>